Mobile Link
サーバ管理

バージョン 16.0

2013 年 2 月

Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

バージョン 16.0
2013 年 2 月

© 2013 SAP AG or an SAP affiliate company.All rights reserved.

次の条件に従うかぎり、このマニュアルの全部または一部を使用、印刷、再生、配布することができます。1) マニュアルの全部または一部にかかわらず、すべてのコピーにこの情報またはマニュアル内のその他の版権と商標の表示を含めること。2) マニュアルに変更を加えないこと。3) SAP 以外の人間がマニュアルの著者または情報源であるかのように示す行為をしないこと。ここに記載された情報は事前の通知なしに変更されることがあります。

SAP AG およびディストリビュータが販売しているソフトウェア製品には、他のソフトウェアベンダー独自のソフトウェアコンポーネントが含まれているものがあります。国内製品の仕様は変わることがあります。

これらの資料は SAP AG および関連会社 (SAP グループ) が情報のみを目的として提供するものであり、いかなる種類の表明または保証も行うものではなく、SAP グループはこの資料に関する誤りまたは脱落について責任を負わないものとします。SAP グループの製品およびサービスに関する保証は、かかる製品およびサービスに付属している明確な保証文書がある場合、そこで明記されている保証に限定されます。ここに記載されているいかなる内容も、追加保証を構成するものとして解釈されるものではありません。

ここに記載された SAP および他の SAP 製品とサービス、ならびに対応するロゴは、ドイツおよび他の国における SAP AG の商標または登録商標です。http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark を参照してください。

目次

はじめに v

Mobile Link サーバテクノロジ 1

Mobile Link 統合データベース 1
Mobile Link サーバ 29
Mobile Link サーバオプション 43
同期の方法 107
Mobile Link のパフォーマンス 142
リモートデータベースの集中管理 152
Mobile Link プロファイラ 191
Relay Server 212
Mobile Link ファイルベースのダウンロード 212

Mobile Link イベント 229

同期スクリプトの作成 229
同期イベント 269

Mobile Link サーバ API 459

Java による同期スクリプトの作成 459
Mobile Link サーバ Java API リファレンス 473
.NET の同期スクリプト 533
Mobile Link サーバ .NET API リファレンス 548
ダイレクトローハンドリング 623

Mobile Link リファレンス 635

Mobile Link Replay C++ コールバック 635
Mobile Link サーバシステムプロシージャ 644
Mobile Link ユーティリティ 692
リモートデータベースと統合データベース間での Mobile Link データマッピング 708

文字セットの考慮事項 758
Mobile Link の ODBC ドライバ 760
Mobile Link アプリケーションの配備 765

索引 781

はじめに

このマニュアルでは、Mobile Link サーバ、統合データベース、および Mobile Link アプリケーションを設定および管理する方法について説明します。また、Mobile Link サーバの正常性や可用性に関する情報を示す Web ブラウザベースの管理ツールである Mobile Link 用の SQL Anywhere モニタ、および Web サーバを通じて通信する Mobile Link とモバイルデバイス間で安全な通信を実現する Relay Server についても説明します。

Mobile Link サーバテクノロジ

この項では、Mobile Link テクノロジの概要と、Mobile Link テクノロジを使用して 2 つ以上のデータソース間でデータを同期する方法について説明します。

Mobile Link 統合データベース

統合データベースには、Mobile Link で必要なシステムオブジェクトが格納されます。通常、統合データベースにはアプリケーションデータも格納されますが、アプリケーションデータのすべてまたは一部は、他の方法でも格納できます。

Mobile Link は、32 ビットおよび 64 ビットの環境にある Windows および Linux 用の統合データベースをサポートしています。統合データベースとして、ODBC に準拠した以下のいずれかの RDBMS を使用できます。

- Adaptive Server Enterprise (64 ビットの Linux のサポートは提供されていません)
- IBM DB2 LUW
- Microsoft SQL Server
- MySQL
- Oracle
- SAP HANA
- SAP Sybase IQ
- SQL Anywhere

サポートされているバージョンについては、http://www.sybase.com/detail?id=1002288 を参照してください。

Mobile Link で推奨される ODBC ドライバの詳細については、http://www.sybase.com/detail?id=1011880 を参照してください。

SQL Anywhere のインストール環境には、各タイプの RDBMS の Mobile Link 設定スクリプトが含まれています。その RDBMS を Mobile Link で使用するには、該当する設定スクリプトを実行する必要があります。設定スクリプトは、Mobile Link で必要なテーブルとストアドプロシージャを追加します。

各タイプの RDBMS を統合データベースとして設定する方法については、「統合データベースの設定」3 ページを参照してください。

特定の統合データベース用の同期スクリプトを記述する方法については、「RDBMS 依存の同期スクリプト」5 ページを参照してください。

他のデータソースとの同期

Mobile Link 環境には、統合データベースとして設定されたデータベースが必要です。しかし、統合データベース以外のデータソースとも同期できます。テキストファイル、Web サービス、非リレーショナルデータベース、スプレッドシートなど、その他のどのようなデータソースもほとんどは使用できます。次のことが可能です。

●統合データベースのみに同期します。

●別のデータソースのみに同期します。

●統合データベースとその他のデータソースの両方に同期するハイブリッドアプリケーションを作成します。

「ダイレクトローハンドリング」623 ページを参照してください。

統合データベースの変更に関する制限

統合データベースのスキーマを変更する機能が制限されている場合があります。このような場合に備えて、Mobile Link には、可能な場合に統合データベースへの変更を最小限に抑えるためのソリューションがあります。たとえば、Mobile Link には、ユニークなプライマリキーを同期システム全体で維持するためのさまざまなソリューションがあり、その中のいくつかは、統合データベースのスキーマへの影響を最小限に抑えます。

また、Mobile Link システムオブジェクトを別のデータベースに置くことによって、統合データベースへのほとんどすべての影響を避けることができます。 「Mobile Link システムデータベース」4 ページを参照してください。

リモートテーブルと統合データベースの関係

同期の設計では、リモートデータベースにあるテーブル／カラムと統合データベースにあるテーブル／カラムの間のマッピングが指定されます。通常、リモートデータベースのテーブルとカラムは、統合データベースのテーブルとカラムと完全に一致するか、それらのサブセットです。

許可されている任意の関係

リモートデータベースにあるテーブルは、統合データベースのテーブルと同じである必要はありません。1 つのリモートアプリケーションテーブルで同期されたデータは、異なるテーブルのカラムに配布できます。これらの関係を指定するには、同期スクリプトを使用します。

シンプルな直接関係

最も簡単で一般的な設計では、統合データベースのテーブル構造のサブセットであるリモートデータベースのテーブル構造を使用します。この設計を使うと、リモートデータベースの各テーブルが統合データベースに存在するようになります。対応するテーブルの構造体と外部キーの関係が、統合データベースのものと同じになります。

統合データベースには、同期されていないカラムとテーブルが含まれていることがよくあります。これらのカラムやテーブルの一部は、同期を支援するために使用されている可能性があります。たとえば、TIMESTAMP カラムは、統合データベース内の新しいローや更新されたローを識別できます。また、シャドウテーブルは、削除を追跡するのに使用できます。統合データベース内の同期されていないカラムやテーブルには、リモートサイトでは必要ない情報を保存できます。

リモートデータベースにも、同期されていないテーブルやカラムが含まれていることがよくあります。

参照

- 「リモートデータベースと統合データベース間での Mobile Link データマッピング」708 ページ

統合データベースの設定

設定スクリプト

Mobile Link 統合データベースとして使用できるようにデータベースを設定するには、設定スクリプトを実行します。SQL Anywhere のインストール環境には、サポートされている各 RDBMS のスクリプトが含まれています。これらのスクリプトは、すべて *%SQLANY16%¥MobiLink¥setup* にあります。Mobile Link システムの設定の詳細については、「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

Mobile Link 設定スクリプトは、データベースに Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加します。これらのテーブルとプロシージャは Mobile Link 同期に必要です。

インストールされるストアドプロシージャの詳細については、「Mobile Link サーバシステムプロシージャ」644 ページを参照してください。

> **注意**
> 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link を設定したりします。 「必要な権限」30 ページを参照してください。

設定スクリプトの実行方法については、使用する RDBMS についての以下の項を参照してください。

- 「Adaptive Server Enterprise 統合データベース」
- 「IBM DB2 LUW 統合データベース」
- 「Microsoft SQL Server 統合データベース」
- 「MySQL 統合データベース」
- 「Oracle 統合データベース」
- 「SAP HANA の統合データベース」24 ページ
- 「SAP Sybase IQ の統合データベース」
- 「SQL Anywhere 統合データベース」

ODBC 接続

Mobile Link サーバでは、統合データベースへの ODBC 接続が必要です。使用している RDBMS 用の適切な ODBC ドライバをインストールして、Mobile Link サーバを実行しているコンピュータのデータベース用に ODBC データソースを作成してください。

Mobile Link ODBC ドライバの詳細については、「Mobile Link の ODBC ドライバ」760 ページを参照してください。

Mobile Link で使用できる ODBC ドライバの最新の情報と完全な機能仕様については、http://www.sybase.com/detail?id=1011880 を参照してください。

Mobile Link システムデータベース

統合データベースをデータ用のデータベースと Mobile Link システム情報用のデータベースの 2 つに分割する必要がある場合もあります。このとき、統合データベースに Mobile Link システムオブジェクトを追加する必要はありません。Mobile Link システムデータベースという別のデータベースにすべての Mobile Link システムオブジェクトを格納できます。

注意
この設定には、分散トランザクションコーディネーターが必要です。現在、Windows でのみ動作する Microsoft Distributed Transaction Coordinator (MS DTC) のみが Mobile Link でサポートされています。

Mobile Link システムデータベースは、統合データベースとしてサポートされる任意のデータベースにすることができます。統合データベースと同じ RDBMS にする必要はありません。

Mobile Link システムデータベースは簡単に設定できます。統合データベース以外のデータベースに Mobile Link 設定スクリプトを適用します。Mobile Link サーバを起動する場合、-c を使用して統合データベースに接続し、-cs を使用してシステムデータベースに接続します。

注意

- 別のシステムデータベースを使用しているときに Microsoft Distributed Transaction Coordinator を使用する場合は、Windows でのみ Mobile Link を実行できます。
- Mobile Link システムデータベースは、Mobile Link の**同期モデル展開ウィザード**で使用することはできません。
- 別のデータベースに Mobile Link システムオブジェクトを格納するため、パフォーマンスの低下が発生します。

Mobile Link サーバのシステムテーブル

Mobile Link システムテーブルには、Mobile Link ユーザ、サブスクリプション、テーブル、スクリプト、スクリプトバージョン、その他の情報が保管されています。Mobile Link システムテーブルは Mobile Link 同期に必須です。Mobile Link システムテーブルの変更は可能ですが、通常は必要ありません。

Mobile Link システムテーブルは、使用している統合データベース用の Mobile Link 設定スクリプトを実行したときに作成されます。Mobile Link システムテーブルにはプレフィクス 「ml_」 を付け、統合データベースに格納してください。設定スクリプトを実行するデータベースユーザが、スクリプトによって作成される Mobile Link システムテーブルの所有者になります。

「統合データベースの設定」3 ページを参照してください。

説明

- この項では、SQL Anywhere 統合データベースの Mobile Link システムテーブルで使用されるデータ型について説明します。一部の RDBMS では、データ型が多少異なります。

RDBMS 依存の同期スクリプト

Mobile Link では、データを同期するときに使用する規則を定義するために、同期スクリプトを使用しています。同期スクリプトでは次の内容を定義します。

●リモートデータベースからアップロードしたデータを、統合データベースに適用する方法。

●統合データベースからリモートデータベースにダウンロードするデータ。

「同期スクリプトの作成」229 ページを参照してください。

スクリプトに記述できるイベントの一覧については、「同期イベント」269 ページを参照してください。

各タイプの統合データベースの固有の情報については、以下の項を参照してください。

- 「SQL Anywhere 統合データベース」
- 「Adaptive Server Enterprise 統合データベース」
- 「IBM DB2 LUW 統合データベース」
- 「Microsoft SQL Server 統合データベース」
- 「MySQL 統合データベース」
- 「Oracle 統合データベース」
- 「SAP HANA の統合データベース」24 ページ
- 「SAP Sybase IQ の統合データベース」

.NET 同期スクリプトと Java 同期スクリプト

データベースが使用している SQL 言語のバージョンで同期論理を記述できます。Java または .NET を使用して、より移植性が高く、強力なスクリプトを記述することもできます。Java と .NET は両方とも、各 RDBMS で SQL を使用した場合よりも高い柔軟性を提供し、SQL との完全な互換性も実現します。Java または .NET の同期論理を使用する場合は、セッション全体の変数の保持、ユーザ定義のプロシージャの作成、外部サーバに対する認証の統合などを行うことができます。

Java 同期論理については、「Java 同期論理」461 ページを参照してください。

.NET 同期論理については、「.NET の同期スクリプト」533 ページを参照してください。

スクリプトからのプロシージャの呼び出し

Microsoft SQL Server などのいくつかのデータベースでは、パラメータを持つプロシージャコールは、ODBC の構文を使用して記述する必要があります。

```
{ CALL procedure_name( {ml param1}, {ml param2}, ... ) }
```

プロシージャ定義の中でパラメータを OUT または INOUT として定義することで、戻り値を返すことができます。

CHAR カラム

他の多くの RDBMS では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモートデータベース (SQL Anywhere または Ultra Light) では、

CHAR は VARCHAR と同じです。固定幅に合わせて値にブランクが埋め込まれることはありません。SQL Anywhere を統合データベースとして使用していない場合は、統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv16 -b コマンドラインオプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

「-b mlsrv16 オプション」50 ページを参照してください。

データ変換

Mobile Link サーバが、SQL Anywhere 以外の統合データベースと通信するときに行われるデータ変換については、「リモートデータベースと統合データベース間での Mobile Link データマッピング」708 ページを参照してください。

空間データの同期

Mobile Link サーバでは、空間データ型のカラムを含むテーブルの同期がサポートされています。空間データの同期では、次のタイプの統合データベースがサポートされています。

- SQL Anywhere
- Oracle
- Microsoft SQL Server
- IBM DB2 LUW
- MySQL

Mobile Link サーバでは、空間データの Well Known Binary (WKB) 表現を Spatial Reference Identifier (SRID) とともに使用して、リモートデータベースから統合データベースにデータがアップロードされます。また、WKB と SRID は、統合データベースからリモートデータベースへの空間データのダウンロードにも使用されます。したがって、アップロードテーブルスクリプトとダウンロードテーブルスクリプトでは、WKB フォーマットの空間データを処理できる必要があります。

次元に関する制限

統合データベースが SQL Anywhere サーバで実行されている場合、Mobile Link サーバは、統合データベースとリモートデータベース間で、2 次元、3 次元、4 次元の空間データを同期できます。ただし、統合データベースが、サポートされている他のいずれかの RDBMS タイプである場合には、統合データベースとリモートデータベース間で 2 次元データのみを同期できます。

アップロードストリームに 3 次元または 4 次元の空間データが含まれており、統合データベースが SQL Anywhere 以外のデータベースサーバで実行されている場合は、Mobile Link サーバで警告メッセージが生成され、3 次元目と 4 次元目のデータが削除されてから、空間データが統合データベースに送信されます。

SRID 要件

Mobile Link サーバでは、統合データベースとリモートデータベースで使用される SRID 間のマッピングは自動的に検出されません。リモートデータベースで使用される SRID が統合データベースで定義されたものと一致していることを確認するか、ユーザ定義のアップロードテーブルスクリプトとダウンロードテーブルスクリプトで SRID を統合データベースとリモートデータベース間で変換できる必要があります。

名前付きパラメータ

アップロードテーブルスクリプトは、名前付きパラメータまたは疑問符 (?) を使用して作成できます。SQL スクリプトでの疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。疑問符を使用する場合は、それぞれの空間カラムに 2 つの疑問符を含めてください。最初の疑問符は WKB 値用であり、2 つ目の疑問符は SRID 値用です。

名前付きパラメータを使用する場合は、次の規則に従ってください。

```
{ml r.column_name:data}

{ml r.column_name:srid}
```

最初の名前付きパラメータは WKB 値を表し、2 つ目の名前付きパラメータは SRID 値用です。

Oracle の考慮事項

Oracle から空間データをダウンロードするときは、次の点を考慮してください。

- Oracle サーバでは、Oracle ネイティブフォーマットを使用してジオメトリ値が Oracle データベースに入力される場合は、Well-Known-Binary (WKB) 値が生成されません。次の例では、回避方法を示します。

  ネイティブ Oracle 空間フォーマットについては、次のようにします。

```
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape SDO_GEOMETRY);

INSERT INTO cola_markets VALUES(
  1,
  'cola_a',
  SDO_GEOMETRY(
    2003,  -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
        -- define rectangle (lower left and upper right) with
        -- Cartesian-coordinate data
  )
);
```

  ダウンロードカーソルが次のように書かれている必要があります。そうでない場合には、カーソルは実際のバイナリ値ではなく 1 を返します。

```
COLA_MARKETS (SCOTT): download_cursor
SELECT t.MKT_ID,
       t.NAME,
```

```
SDO_UTIL.TO_WKBGEOMETRY(SDO_UTIL.FROM_WKTGEOMETRY(SDO_UTIL.TO_WKTGEOM
ETRY(t.SHAPE))),
   t.SHAPE.sdo_srid
FROM SCOTT.COLA_MARKETS t
```

- 次のように、データが WKT フォーマットで入力されている場合は、デフォルトのダウンロードスクリプトは正しく動作します。

```
INSERT INTO cola_markets VALUES(
 1,
 'cola_a',
 SDO_GEOMETRY( 'polygon (( 1 1, 5 1, 5 7, 1 7, 1 1))',4326));
INSERT INTO SCOTT.cola_markets VALUES(
 2,
 'cola_b',
 SDO_GEOMETRY( 'polygon (( 5 1, 8 1, 8 6, 5 7, 5 1))',4326));
INSERT INTO SCOTT.cola_markets VALUES(
 3,
 'cola_c',
 SDO_GEOMETRY( 'polygon (( 3 3, 6 3, 6 5, 4 5, 3 3))',4326));
```

- Oracle には 0 の SRID はありません。そのため、すべての平坦な直交平面は 4326 の SRID を使用して入力する必要があります。

参照

- 「空間データ」『SQL Anywhere サーバ 空間データサポート』
- 「名前付きスクリプトパラメータ」234 ページ
- 「ユーザ定義の名前付きパラメータ」250 ページ

アップロードスクリプトとダウンロードスクリプト

テーブルに空間データが含まれている場合、使用している統合データベースのタイプによって、アップロードスクリプトとダウンロードスクリプトが大幅に異なることがあります。次の例では、Mobile Link サーバが現在サポートしている統合データベースの空間データに使用するアップロードスクリプトとダウンロードスクリプトのサンプルをいくつか示します。

この例では、同期テーブルを次のように想定しています。

```
create table test (c1 int not null primary key, c2 st_geometry )
```

各タイプの統合データベースの固有の情報については、以下の項を参照してください。

- 「SQL Anywhere 統合データベース」
- 「Microsoft SQL Server 統合データベース」
- 「MySQL 統合データベース」
- 「Oracle 統合データベース」
- 「SAP Sybase IQ の統合データベース」

SQL Anywhere サンプルスクリプト

次に、SQL Anywhere の upload_insert サンプルスクリプトを示します。

```
INSERT INTO test VALUES( {ml r.c1}, ST_Geometry::ST_GeomFromBinary({ml r.c2:data},{ml
r.c2:srid}) )
```

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

次に、SQL Anywhere の download_cursor サンプルスクリプトを示します。

```
SELECT c1, c2.ST_AsBinary(), c2.ST_SRID() FROM test
```

Microsoft SQL Server のサンプルスクリプト

次に、Microsoft SQL Server の upload_insert サンプルスクリプトを示します。

```
BEGIN
    DECLARE @c1 INTEGER
    DECLARE @v1 VARBINARY(max)
    DECLARE @s1 INTEGER
    DECLARE @g1 geometry
    SELECT @c1 = {ml r.c1}
    SELECT @v1 = {ml r.c2:data}
    SELECT @s1 = {ml r.c2:srid}
    IF @v1 IS NULL
        SELECT @g1 = NULL
    ELSE
        SELECT @g1 = Geometry::STGeomFromWKB(@v1,@s1)
    INSERT INTO test VALUES( @c1, @g1 )
END
```

次に、Microsoft SQL Server の download_cursor サンプルスクリプトを示します。

```
SELECT c1, c2.STAsBinary(), c2.STSrid FROM test
```

Oracle のサンプルスクリプト

次に、Oracle の upload_insert サンプルスクリプトを示します。

```
DECLARE
    v_c1 INTEGER;
    v_v1 sdo_geometry;
    v_s1 INTEGER;
    v_u1 blob;
BEGIN
    v_c1 := {ml r.c1};
    v_u1 := {ml r.c2:data};
    v_s1 := {ml r.c2:srid};
    IF v_u1 IS NULL THEN
        v_v1 := NULL;
    ELSE
        v_v1 := sdo_geometry( v_u1, v_s1 );
    END IF;
    INSERT INTO test VALUES( v_c1, v_v1 );
END;
```

次に、Oracle の download_cursor サンプルスクリプトを示します。

```
SELECT c1, g.c2.Get_WKB(), g.c2.sdo_srid FROM test g
```

IBM DB2 のサンプルスクリプト

次に、IBM DB2 の upload_insert サンプルスクリプトを示します。

```
BEGIN ATOMIC
    DECLARE v_c1 INTEGER;
    DECLARE v_v1 BLOB(10m);
    DECLARE v_s1 INTEGER;
    SET v_c1 = {ml r.c1}; SET v_v1 = {ml r.c2:data}; SET v_s1 = {ml r.c2:srid};
```

```
    INSERT INTO test VALUES( v_c1, ST_Geometry( v_v1, v_s1 ) );
END
```

次に、IBM DB2 の download_cursor サンプルスクリプトを示します。

```
SELECT c1, ST_AsBinary( c2 ), ST_SRID( c2) FROM test
```

MySQL のサンプルスクリプト

次に、MySQL の upload_insert サンプルスクリプトを示します。

```
INSERT INTO test VALUES( {ml r.c1}, GeometryFromWKB({ml r.c2:data},{ml r.c2:srid}) )
```

次に、MySQL の download_cursor サンプルスクリプトを示します。

```
SELECT c1, AsBinary( c2 ), SRID( c2 ) FROM test
```

Adaptive Server Enterprise 統合データベース

前提条件

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

- Mobile Link サーバのログイン ID には、MASTER..SYSTRANSACTIONS に対する SELECT 権限が必要です。

- mlsrv16 の -cs オプションを使用する場合は、Mobile Link サーバのログイン ID に dtm_tm_role ロールが必要です。

- sp_dboption オプションを使用して、SELECT INTO オプションを true に設定する必要があります。たとえば、**your-database-name** の SELECT INTO 権限を true に設定するには、Interactive SQL で次のスクリプトを実行します。

  ```
  sp_dboption your-database-name, "SELECT INTO", true
  go
  ```

Adaptive Server Enterprise を統合データベースとして設定する

Adaptive Server Enterprise を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *%SQLANY16%¥MobiLink¥setup* にある *syncase.sql* 設定スクリプトを実行します。

- Sybase Central で Mobile Link のシステム設定を確認し、更新します。「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

ODBC ドライバ

Adaptive Server Enterprise データベースで提供されている ODBC ドライバを使用して、Adaptive Server Enterprise 統合データベース用の ODBC DSN を設定してください。次の項を参照してください。

- http://www.sybase.com/detail?id=1011880
- Adaptive Server Enterprise のマニュアル

Adaptive Server Enterprise の考慮事項

- **Enable functionality group 設定パラメータ**　ASE 15.7 サーバで **enable functionality group** 設定パラメータが有効である場合、Mobile Link サーバは、同じリモート ID を使用する冗長な同時同期を防ぐために、「select ... for update」機能を使用してリモート ID をロックします。**enable functionality group** パラメータをオフにする場合は、現在 ASE サーバに接続されている任意の Mobile Link サーバを再起動して同期要求の失敗を回避してください。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモートデータベース間で完全に一致する必要があります。「Adaptive Server Enterprise データのマッピング」708 ページを参照してください。

- **CHAR カラム**　Adaptive Server Enterprise では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモートデータベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じです。固定幅に合わせて値にブランクが埋め込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv16 -b コマンドラインオプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  詳細については、「-b mlsrv16 オプション」50 ページを参照してください。

- **BLOB サイズ**　データサイズが 32 KB (デフォルト) より大きい BLOB データをダウンロードするには、次の操作を行います。

  - Windows の場合は、**[Adaptive Server Enterprise ODBC Driver Configuration]** ウィンドウの **[詳細]** ページにある **[テキスト・サイズ]** を想定される最大 BLOB サイズよりも大きい値に設定します。
  - Linux の場合は、*.obdc.ini* ファイル内の TextSize エントリを想定される最大 BLOB サイズよりも大きい値に設定します。

- **VARBIT の制限**　Mobile Link では、長さ 0 (空) の VARBIT または LONG VARBIT 値と Adaptive Server Enterprise 統合データベースとの同期はサポートされていません。Adaptive Server Enterprise は VARBIT 型をサポートしていないので、これらの型は通常、Adaptive Server Enterprise データベースの VARCHAR または TEXT カラムと同期されます。Adaptive Server Enterprise では、空の文字列値は 1 つのスペースに変換されます。SQL Anywhere の VARBIT

カラムではスペースを使用できないので、これらの値をダウンロードしようとすると、リモートデータベースでエラーとなります。

独立性レベル

「Mobile Link 独立性レベル」140 ページを参照してください。

IBM DB2 LUW 統合データベース

Mobile Link は、Linux と Windows 用の IBM DB2 LUW をサポートしています。

IBM DB2 LUW を統合データベースとして設定する

IBM DB2 を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。

前提条件

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

内容と備考

Sybase Central で Mobile Link のシステム設定を確認し、更新することもできます。「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

♦ タスク

1. 設定スクリプトを使用して Mobile Link システムテーブルをインストールするには、目的の IBM DB2 LUW テーブル領域は最低でも 8 KB ページを使用します。テーブル領域が 8 KB ページを使用しない場合は、次の手順を実行します。

   - 1 つ以上のバッファプールに 8 KB ページがあることを確認します。ない場合は、8 KB ページのバッファプールを作成してください。
   - 8 KB ページのバッファプールを使用する、新しいテーブル領域とテンポラリテーブル領域を作成します。

     詳細については、IBM DB2 LUW のマニュアルを参照してください。

2. 使用する接続情報を含むように、*syncdb2.sql* をカスタマイズします。

a. *syncdb2.sql* を変更と保存を行う新しいロケーションにコピーします。

b. *syncdb2.sql* スクリプトには、デフォルトの接続文 connect to DB2Database が含まれています。IBM DB2 データベースに接続するようにこの行を変更します。次の構文を使用します。

```
connect to DB2Database user userid using password ~
```

ここでは、*DB2Database*、*userid*、*password* に適切な名前を指定します。(*syncdb2.sql* スクリプトでは、チルダ (~) をコマンドデリミタとして使用します。)

3. *syncdb2.sql* を実行します。

```
db2 -c -ec -td~ +s -v -f syncdb2.sql
```

結果

Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューがインストールされ、IBM DB2 LUW を統合データベースとして使用できるようになります。

次の手順

IBM DB2 データベースで提供されている ODBC ドライバを使用して、IBM DB2 統合データベース用の ODBC DSN を設定してください。次の項を参照してください。

- http://www.sybase.com/detail?id=1011880
- IBM DB2 LUW のマニュアル

IBM DB2 LUW の考慮事項

- **ロックのエスカレーション**　Mobile Link サーバでは、統合データベースとリモートデータベースのデータ間の整合性を維持するために、すべての同期フェーズでリモート ID をロックするために、ml_lock_rid ストアドプロシージャを介して次のクエリを発行します。

  ```
  SELECT sync_key into p_sync_key FROM ml_database WHERE rid = a_given_remote_id WITH
  RR USE AND KEEP EXCLUSIVE LOCKS;
  ```

  このクエリにより、同じリモート ID を使用する同時同期が発生しないよう、リモート ID は排他的にロックされます。

  同じリモート ID を使用する同時同期がないにもかかわらず Mobile Link のリモート ID のロックエラーが発生する場合は (たとえば、Mobile Link サーバのログに Mobile Link エラーコード -10341 がある場合)、DB2 の maxlocks と locklist 設定パラメータを調整してロックのエスカレーションを回避する必要があります。詳細については、DB2 のマニュアルを参照してください。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモートデータベース間で完全に一致する必要があります。 詳細については、「IBM DB2 LUW データのマッピング」718 ページを参照してください。

- **CHAR カラム**　IBM DB2 LUW では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモートデータベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じです。固定幅に合わせて値にブランクが埋め込

まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv16 -b コマンドラインオプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

「-b mlsrv16 オプション」50 ページを参照してください。

- **テーブル領域の容量**　IBM DB2 LUW データベースを統合データベースとして使用する場合のテーブル領域とテンポラリテーブル領域は、最低でも 8 KB ページを使用します。

  また、LONG テーブル領域が必要なカラムもあります。次の例のように、デフォルトの LONG テーブル領域がない場合は、これらのカラムを含むテーブルの作成文を正しく設定します。

  ```
  CREATE TABLE ... ( ... )
  IN tablespace
  LONG IN long-tablespace
  ```

  サンプルアプリケーションの使用例については、「Mobile Link の CustDB サンプル」『Mobile Link クイックスタート』を参照してください。

- **システムプロシージャコール内の引用符を 2 つにする**　Mobile Link システムプロシージャを使用して、スクリプトを IBM DB2 統合データベースに追加する場合は、引用符を 2 つにする必要があります。たとえば、ml_add_table_script を使用して追加するスクリプトに、他の統合データベースに対する SET "DELETED"="Y" という行が含まれている場合、IBM DB2 では、これを SET "DELETED" = ""Y"" と記述する必要があります。

独立性レベル

「Mobile Link 独立性レベル」140 ページ を参照してください。

Microsoft SQL Server 統合データベース

前提条件

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、テーブル、トリガ、ストアドプロシージャを作成できる必要があるため、db_owner ロールを持っている必要があります。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

- Mobile Link サーバのログイン ID には、VIEW SERVER STATE パーミッションが必要です。このパーミッションは、Microsoft SQL Server の master データベース内で次の SQL 文を使用して付与できます。

```
grant view server state to user_name
```

- Mobile Link サーバのログイン ID には、sys.databases SYS.DM_TRAN_LOCKS、SYS.PARTITIONS、SYS.SYSPROCESSES に対する SELECT 権限が必要です。

Microsoft SQL Server を統合データベースとして設定する

Microsoft SQL Server を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *%SQLANY16%¥MobiLink¥setup* にある *syncmss.sql* 設定スクリプトを実行します。
- Sybase Central で Mobile Link のシステム設定を確認し、更新します。「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

ODBC ドライバ

SQL Server データベースで提供されている ODBC ドライバを使用して、SQL Server 統合データベース用の ODBC DSN を設定してください。次の項を参照してください。

- http://www.sybase.com/detail?id=1011880
- Microsoft SQL Server のマニュアル

SQL Server の考慮事項

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモートデータベース間で完全に一致する必要があります。 詳細については、「Microsoft SQL Server データのマッピング」727 ページを参照してください。
- **BLOB カラム**　SQL Server ODBC ドライバには制限があります。このため、同期テーブルを定義する場合、具体的には、テーブル用の download_cursor スクリプトをストアドプロシージャコールとしてまたは SQL 文のバッチとして作成する必要がある場合には、すべての BLOB カラムをカラムの末尾に配置してください。BLOB カラムが含まれる同期テーブル用の download_cursor スクリプトを単一の SELECT 文として作成する場合は、この制限を無視できます。http://support.microsoft.com/kb/238080 を参照してください。
- **CHAR カラム**　Microsoft SQL Server では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモートデータベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じです。固定幅に合わせて値にブランクが埋め込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することをおすすめします。CHAR を使用する必要がある場合は、mlsrv16 -b コマンドラインオプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  「-b mlsrv16 オプション」50 ページを参照してください。
- **SET NOCOUNT ON**　Microsoft SQL Server の場合には、すべてのストアドプロシージャまたは ODBC を使用して実行される SQL バッチの最初の文として SET NOCOUNT ON を指定してください。
- **プロシージャコール**　Microsoft SQL Server では、パラメータを持つプロシージャコールは、ODBC の構文を使用して記述する必要があります。

```
{ CALL procedure_name( {ml param1}, {ml param2}, ... ) }
```

- **サンプルデータベースでの計算カラムの使用**　SQL Server AdventureWorks サンプルデータベースには、計算カラムが含まれています。計算カラムはアップロードできません。カラムをダウンロードのみに設定することと、カラムを同期から除外することは可能です。

- **upload_update スクリプトでの競合検出の実装**　SQL Server では、upload_update スクリプトで競合の検出と解決を行う必要があります。 「upload_update スクリプトによる競合の解決」130 ページを参照してください。

独立性レベル

「Mobile Link 独立性レベル」140 ページ を参照してください。

MySQL 統合データベース

Mobile Link サーバでは、MySQL Community サーバと Enterprise サーバ 5.1.3 以降がサポートされています。

権限

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

記憶エンジン

Mobile Link サーバでは、デフォルトのストレージエンジンが ACID に準拠していることを必要としています。デフォルトのストレージエンジンが ACID に準拠していない場合は、Mobile Link サーバのすべてのテーブルが、InnoDB、Falcon など、ACID 準拠のストレージエンジンを使用して作成されていることを確認してください。その確認を行わないと、データの不整合が発生する場合があります。

必要な場合は、MySQL データベースに対してファイルを適用する前に、*syncmys.sql* スクリプトファイルを編集して次の 2 つの行を行 **delimiter //** の後に追加します。ここで、*engine_name* は ACID 準拠の記憶エンジンです。

```
set storage_engine = [engine_name]
  //
```

設定スクリプトを使用して MySQL を統合データベースとして設定する

MySQL を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。

前提条件

MySQL ユーザ ID には、テーブル、プロシージャ、ファンクション、ビュー、トリガを作成する権限が必要です。

内容と備考

Sybase Central 用の Mobile Link 16 プラグインを使用して、MySQL を統合データベースとして設定することもできます。

♦ タスク

- MySQL コマンドラインツールまたは MySQL Query Browser を使用して、*%SQLANY16%¥MobiLink¥setup* にある *syncmys.sql* 設定スクリプトを実行します。

結果

MySQL データベースは、Mobile Link 統合データベースとして使用できます。

Sybase Central を使用して MySQL を統合データベースとして設定する

MySQL を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。

前提条件

MySQL ユーザ ID には、テーブル、プロシージャ、ファンクション、ビュー、トリガを作成する権限が必要です。

内容と備考

syncmys.sql 設定スクリプトを使用して、MySQL を統合データベースとして設定することもできます。

♦ タスク

1. Sybase Central の Mobile Link 16 プラグインで、**[表示]** » **[フォルダ]** を選択します。
2. Mobile Link プロジェクトを開き、**[統合データベース]** を展開します。
3. データベース名を右クリックし、**[Mobile Link システム設定のチェック]** をクリックします。データベースの設定が必要な場合は、続行のプロンプトが表示されます。

   既存の Mobile Link システム設定を使用する場合、default_schema を Mobile Link システム設定のスキーマにしてください。

結果

MySQL データベースは、Mobile Link 統合データベースとして使用できます。

ODBC ドライバ

MySQL Web サイトで提供されている ODBC ドライバを使用して、MySQL 統合データベース用の ODBC DSN を設定してください。Mobile Link サーバは、MySQL ODBC ドライバ 5.1.3 以降をサポートしています。次の項を参照してください。

- http://www.sybase.com/detail?id=1011880

UINIX で ODBC 設定ファイルを指定するには、次のいずれかを行います。

- *ODBC.INI* ファイルを現在のユーザのホームディレクトリに配置します。
- **ODBCINI** 環境変数を作成し、*ODBC.INI* ファイルのディレクトリロケーションに設定します。

MySQL の考慮事項

- **データ型マッピング**　カラムの型は、統合データベースとリモートデータベース間で完全に一致する必要があります。 詳細については、「MySQL データのマッピング」734 ページを参照してください。

- **ストアドプロシージャ**　ストアドプロシージャコールでは、INOUT パラメータおよび OUT パラメータを使用することはできません。これらのパラメータが必要なプロシージャは、OUT 値を返す関数として実装してください。

  authenticate_user、modify_user などの INOUT パラメータが必要なサーバイベントは、関数として実装し、CALL 文ではなく SELECT 文を使用して実行してください。

- **名前付きパラメータ**　ユーザ定義の名前付きパラメータは、サポートされていません。

- **カーソルスクリプト**　イベント upload_fetch、download_cursor、download_delete_cursor は、SELECT 文を使用して呼び出してください。SELECT 文は、Mobile Link サーバによって、コミットされた読み出し独立性レベルを使用して実行されます。MySQL ODBC ドライバのバグによって、INSERT 文、UPDATE 文、DELETE 文など、コミットされない読み出し操作がサーバで発生し、統合データベースとリモートデータベースの間でデータの一貫性が失われます。

  この問題を回避するには、SELECT 文に LOCK IN SHARE MODE 句を追加します。次に例を示します。

  ```
  SELECT column1 FROM table1 WHERE id > 0 LOCK IN SHARE MODE
  ```

  この句は、コミットされない操作から SELECT 文を保護します。

- **バルクアップロード**　MySQL ODBC ドライバでは、現在、バルクアップロードが正しくサポートされていません。

- **MLSD**　MySQL ODBC ドライバでは、現在、MSDTC がサポートされていないため、別の MLSD はサポートされていません。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

- **MySQL サーバの設定**　Mobile Link 同期スクリプトは、ml_script テーブルに TEXT として格納され、必要に応じて取得されます。*my.ini* ファイルの max_allowed_packet は、16m 以上に設定することが必要な場合があります。

- **競合検出**　MySQL 統合データベースで競合検出用に生成されたスクリプトには、複数の文があります。競合検出を使用している場合は、Mobile Link サーバが MySQL データベースへの接続を作成するための DSN を設定するときに、**[MySQL Connector/ODBC Data Source Configuration]** ウィンドウの **[Flags 3]** ページで **[Allow Multiple Statements]** チェックボックスをオンにする必要があります。

- **複数の文**　いずれかの同期スクリプトに、セミコロンで区切られた、バッチによる SQL コマンドが含まれている場合は、Mobile Link サーバが MySQL データベースへの接続を作成するための DSN を設定するときに、**[MySQL Connector/ODBC Data Source Configuration]** ウィンドウの **[Flags 3]** ページで **[Allow Multiple Statements]** チェックボックスをオンにすることが必要な場合があります。

独立性レベル

「Mobile Link 独立性レベル」140 ページを参照してください。

Oracle 統合データベース

権限

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。

  また、RDBMS ユーザの場合は、GV$TRANSACTION、GV$SESSION、GV$LOCK、DBA_OBJECTS での SELECT 権限、および DBMS_UTILITY での EXECUTE 権限も必要です。同義語である GV$TRANSACTION、GV$SESSION、GV$LOCK には直接パーミッションを付与することができません。そのため、同義語の派生元である GV_$TRANSACTION、GV_$SESSION、および GV_$LOCK の動的パフォーマンスビューに対してパーミッションを付与する必要があります。このアクセス権を付与するには、SYS として接続する必要があります。このアクセス権を付与する Oracle の構文は次のとおりです。

```
grant select on SYS.GV_$TRANSACTION to user-name;

grant select on SYS.GV_$SESSION to user-name;

grant select on SYS.GV_$LOCK to user-name;

grant execute on SYS.DBMS_UTILITY to user-name;
```

Oracle を統合データベースとして設定する

Oracle を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

●*%SQLANY16%¥MobiLink¥setup* にある *syncora.sql* 設定スクリプトを実行します。

●Sybase Central で Mobile Link のシステム設定を確認し、更新します。「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

ODBC ドライバ

Oracle 統合データベースには、ODBC DSN を設定する必要があります。次の項を参照してください。

- 「SQL Anywhere 16 - Oracle ODBC ドライバ」
- http://www.sybase.com/detail?id=1011880

Oracle の考慮事項

- **Oracle Real Application Clusters 内での Mobile Link の同期とタイムスタンプベースのダウンロード**　Mobile Link サーバが次の最終ダウンロードタイムスタンプをフェッチする時刻と、ダウンロードされるローをフェッチする時刻との差を超える差異が、Oracle クラスタノードの各クロック間に存在する場合、Oracle RAC で実行されている統合データベースのローが欠落する場合があります。この問題は、ノードのクロックが同期される RAC システムではあまり起こりませんが、ノードのクロック間の差異が大きければ大きいほど、発生する可能性も大きくなります。対処方法としては、modify_next_last_download_timestamp または modify_last_download_timestamp スクリプトのいずれかを作成し、ノードのクロック間の最大差をなくします。

  Oracle では、少なくともバージョン 10i 以降では、クラスタ内のすべてのクロックの同期に Network Time Protocol (NTP) を使用することをすすめています。NTP は、UNIX と Linux では通常デフォルトで実行されます。クラスタのノードが NTP を使用するよう正しく構成されている場合、(NTP サーバとどれくらい近いかにより) それらのクロック間の時刻差は 200 マイクロ秒から 10 ミリ秒の範囲となります。Windows Server 2003 以降、Windows Time サービスには、デフォルトで実行される NTP バージョン 3 プロトコルが実装されています。また、バージョン 11gR2 以降の Oracle Clusterware には、クロックの同期をモニタする Oracle クラスタ時刻同期化サービス (CTSS) が含まれています。これでは、NTP または Windows Time サービスのいずれも実行されていない場合、クロックの同期がアクティブに維持されます。ただし、CTSS と Windows Time サービスは NTP と比較すると正確性に劣ります。

  next_last_download_timestamp をフェッチする時刻と、ダウンロードするローをフェッチする時刻との差が Oracle RAC のノードのクロックで最大 1 秒ある場合、ローの欠落を防ぐために、Mobile Link サーバでは、次の条件が該当する場合、統合データベースからフェッチした next_last_download_timestamp から 1 秒を減算します。

  ○Mobile Link サーバが使用する Oracle アカウントに、SYS.DBMS_UTILITY の実行権限があります

  ○統合データベースとは Oracle RAC システムです

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

○Mobile Link バージョン 12.0.0 以降で、generate_next_last_download_timestamp スクリプトがありません

Oracle RAC ノードのクロック間に大きな時間差がある場合、ノードのクロック間の最大時間差をなくす generate_next_last_download_timestamp、modify_next_last_download_timestamp または modify_last_download_timestamp スクリプトを定義してこの問題を回避できます。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモートデータベース間で完全に一致する必要があります。詳細については、「Oracle データのマッピング」740 ページを参照してください。

- **XMLTYPE データ型**　SQL Anywhere または Ultra Light で Oracle XMLTYPE データ型を使用する場合は、特に注意が必要です。 詳細については、Oracle の XMLTYPE データ型 22 ページを参照してください。

- **CHAR カラム**　Oracle では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモートデータベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じです。固定幅に合わせて値にブランクが埋め込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv16 -b コマンドラインオプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  「-b mlsrv16 オプション」50 ページを参照してください。

- **タイムスタンプ**　Mobile Link サーバは、gv$transaction を使用して、リモートデータベースが次の同期に使用するタイムスタンプを生成するため、Mobile Link サーバのログイン ID には gv_$transaction で SELECT 権限が必要です。Oracle では、gv$transaction に直接アクセス権を付与できません。そのため、その基底にある gv_$transaction ビューに対して SELECT 権限を付与する必要があります。 「権限」19 ページを参照してください。

- **ストアドプロシージャ**　ストアドプロシージャを使用して結果セットを返す場合、または VARRAY パラメータを受け入れる場合は、SQL Anywhere 16 - Oracle ODBC ドライバの **[プロシージャから結果が返される、または VARRAY パラメータを使用する]** オプションを選択してください。また、Sybase Central では、リモートデータベースの集中管理に使用するために、プロシージャが結果を返すことが必要となるため、集中管理を使用する場合には、このオプションを選択する必要があります。

  「SQL Anywhere 16 - Oracle ODBC ドライバ」761 ページを参照してください。

- **セッション全体の変数**　セッション全体の情報を保持するには、Oracle パッケージ内の変数を使用します。Oracle パッケージでは、変数の作成、修正、破棄が可能です。これらの変数は Oracle パッケージが現在のパッケージであるかぎり有効です。

- **オートインクリメントメソッド**　プライマリキーの一意性を維持するには、Oracle シーケンスを使用して、SQL Anywhere オートインクリメントフィールドのキーのリストに似た、キーのリストを生成します。CustDB サンプルデータベースの *Samples¥MobiLink¥CustDB¥custora.sql* に、サンプルコーディングが用意されています。ただし、オートインクリメントとは異なり、シーケンスを明示的に参照する必要があります。SQL Anywhere オートインクリメントは、INSERT 文でカラムが参照されていない場合は、自動的にカラム値を挿入します。

- **Oracle では空の文字列をサポートしていない**　Oracle では、空の文字列は NULL として処理されます。SQL Anywhere と Ultra Light では、空の文字列は NULL とは異なる意味を持ちます。したがって、Oracle 統合データベースがある場合には、クライアントデータベースで空の文字列を使用しないようにしてください。

Oracle の XMLTYPE データ型

Oracle XMLTYPE データ型は、SQL Anywhere の XML データ型または Ultra Light の LONG VARCHAR または VARCHAR(*n*) データ型にマッピングすることができます。Oracle データベースサーバではデータを XMLTYPE カラムに格納する前にデータの検証が行われますが、SQL Anywhere と Ultra Light ではそれが行われないので、アップロードする XML ドキュメントに有効な XML が含まれていることを確認する必要があります。

サイズが 32 KB 以下の小さな XML ドキュメントの場合は、Oracle PL/SQL 文を使用して Oracle データベースへのアップロードまたはこのデータベースからのダウンロードが可能です。XML ドキュメントのサイズが 32 KB を超える場合は、upload_insert スクリプトと upload_update スクリプトを使用してグローバルテンポラリテーブルにアップロード XML ドキュメントをアップロードすることが必要になる場合があります。end_upload_rows スクリプトまたは end_upload スクリプトを使用すれば、アップロードデータを変換して実際の同期テーブルに格納することができます。

次の例は、Oracle 統合データベースと SQL Anywhere リモートデータベースとの間で XMLTYPE オブジェクトをそれぞれアップロードおよびダウンロードするアップロードスクリプトとダウンロードスクリプトのサンプルを示します。これらの例では、アップロードテーブルは Oracle 統合データベースで次のように定義されます。

```
create table test (pk int not null primary key, c1 XMLTYPE)
```

SQL Anywhere リモートデータベースでは、アップロードテーブルが次のように定義されます。

```
create table test (pk int not null primary key, c1 XML)
```

- **すべての XML ドキュメントのサイズが 32KB 以下である場合は、アップロードスクリプトとダウンロードスクリプトを次のように作成できます。**　upload_insert

  ```
  declare v_pk integer; v_c1 clob; x_c1 xmltype;
     begin
        v_pk := {ml r.pk};
        v_c1 := {ml r.c1};
        x_c1 := XMLTYPE.createXML( v_c1 );
        insert into test values( v_pk, x_c1 );
     end;
  ```

  download_cursor

  ```
  select pk, XMLSERIALIZE( content c1 ) from test
  ```

  この upload_insert スクリプトは、XML データのサイズが 32 KB 以下である場合に適切に機能します。しかし、XML データのサイズが 32 KB を超えると、Oracle サーバでエラーが発生する場合があります。

- **サイズが 32 KB を超える XML ドキュメントが存在する場合は、アップロード XML データをグローバルテンポラリテーブルにアップロードする必要があります。**　XML ドキュメントは、

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

upload_insert スクリプトによって Oracle 統合データベース内のグローバルテンポラリテーブルにアップロードされます。グローバルテンポラリテーブルは、次のように定義されます。

```
create global temporary table tmp_test (pk int, c1 CLOB)
```

upload_insert スクリプトは、次のように作成します。

```
insert into tmp_test values( {ml r.pk}, {ml r.c1} )
```

注意
テンポラリテーブル内の c1 カラムは、CLOB データ型を持つ必要があります。

end_upload_rows スクリプトは、XML ドキュメントをグローバルテンポラリテーブルから取得し、それを XML ドキュメントに変換し、XML データをテストテーブルに保存します。次に示すのは、end_upload_rows スクリプトです。

```
insert into test (pk, c1) (select pk, XMLTYPE.createXML(c1) from tmp_test
```

独立性レベル

「Mobile Link 独立性レベル」 140 ページ を参照してください。

Oracle VARRAY

SQL Anywhere 16 - Oracle ODBC ドライバでは、ストアドプロシージャで Oracle VARRAY の使用がサポートされています。ストアドプロシージャ内に作成されたアップロードスクリプト (upload_insert、upload_update、upload_delete) で VARRAY を使用すると、ストアドプロシージャ内に作成されたアップロードスクリプトで VARRAY を使用しない場合と比べて、Mobile Link サーバのパフォーマンスが向上することがあります。通常、ストアドプロシージャを使用しない INSERT、UPDATE や DELETE などの単純な SQL 文が最高のパフォーマンスを発揮します。しかし、ストアドプロシージャを使用すると (VARRAY を使用する方法を含む)、単純な文では不可能なビジネスロジックの適用が可能になります。

VARRAY の例

次に、VARRAY を使用する簡単な例を示します。

1. 3 つのカラムが含まれる my_table という名前のテーブルを作成します。

   ```
   create table my_table ( pk integer primary key, c1 number(20), c2 varchar2(4000) )
   ```

2. VARRAY を使用するユーザ定義のコレクション型を作成します。

   ```
   create type my_integer is varray(100) of integer;
   create type my_number is varray(100) of number(20);
   create type my_varchar is varray(100) of varchar2(8000);
   ```

   my_varchar は、100 個の要素を含む VARRAY として定義されます。各要素のデータ型は varchar2 で、幅は 8000 です。この幅は、my_table に指定された幅の 2 倍の大きさにする必要があります。

3. 挿入用のストアドプロシージャを作成します。

```
create or replace procedure my_insert_proc( pk_v my_integer, c1_v my_number, c2_v
my_varchar )
is
c2_value my_varchar;
begin
    c2_value := c2_v;  -- Work around an Oracle bug
    FORALL i in 1 .. pk_v.COUNT
        insert into my_table ( pk, c1, c2 ) values( pk_v(i), c1_v(i), c2_value(i) );
end;
```

VARRAY の制限

ストアドプロシージャで VARRAY を使用するときは、次の制限があります。

- ODBC データソースでは、**[Microsoft 分散トランザクションを有効にする]** チェックボックスがクリアされている必要があります。
- BLOB と CLOB の VARRAY はサポートされません。
- VARRAY のデータ型が CHAR、VARCHAR、NCHAR、または NVARCHAR である場合、ユーザ定義の VARRAY 型は、テーブルのカラムに指定された長さの 2 倍の大きさでなければなりません。
- Mobile Link サーバから Oracle 統合データベースに送信される VARRAY のローの数は、VARRAY 型で宣言された VARRAY のサイズではなく、-s オプションで設定されます。-s オプションは、同期スクリプトで使用されている VARRAY 型のうち最小のサイズより大きくしてはなりません。大きくすると、Mobile Link サーバはエラーを発行します。 「-s mlsrv16 オプション」75 ページを参照してください。

SAP HANA の統合データベース

権限

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルにアクセスするユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

- RDBMS ユーザはまた、CATALOG READ 権限を持つ必要があります。

SAP HANA を統合データベースとして設定する

SAP HANA を Mobile Link 統合データベースとして動作するよう設定してするには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、ビューを追加する必要があります。次の方法を使用して、SAP HANA を設定します。

- SAP HANA Studio で *synchana.sql* 設定スクリプトを実行します。このスクリプトファイルは、*%SQLANY16%¥MobiLink¥Setup* にあります。

ODBC ドライバ

Mobile Link サーバでは、SAP HANA データベースの SAP HDBODBC からの ODBC ドライバを認識できるにすぎません。参照：

- http://www.sybase.com/detail?id=1011880
- SAP HANA ドキュメント

ストアドプロシージャ

SAP HANA では、スカラー入力パラメータに NULL を指定してストアドプロシージャを呼び出すことをサポートしていません。そのため、NULL 入力可のカラムを保持する同期テーブル用のストアドプロシージャ呼び出しとして、upload_insert スクリプトと upload_update スクリプトを実装しないでください。Mobile Link サーバは、NULL を、パラメータの長さ 0 の文字列 (CHAR 型) に置換するか、パラメータのバイナリ値 (BINARY 型) に置換し、その後、SAP HANA データベースによってユーザ認証スクリプトが呼び出されるときに、置換結果を SAP HANA データベースに渡します。ユーザ認証ストアドプロシージャを作成する場合は、この Mobile Link サーバの動作を考慮に入れてください。

ユーザ定義のパラメータはまた、最初に参照されたときに、長さ 0 の文字列に設定されます。

SAP HANA の統合

- **Mobile Link システムデータベース (MLSD)**　Mobile Link サーバでは、SAP HANA 用に MLSD をサポートしていません。

- **Mobile Link サーバシステムオブジェクト**　Mobile Link サーバシステムテーブル内のプライマリキーカラムは、ストアドプロシージャコールを介した SAP HANA SEQUENCEs によって維持されます。

  接続スクリプトとテーブルスクリプトを追加、変更、または削除するには、ml_add_connection_script および ml_add_table_script システムプロシージャをそれぞれ使用し、Mobile Link ユーザとリモートデータベースを追加するには、ml_add_user および ml_add_database システムプロシージャをそれぞれ使用します。Mobile Link サーバシステムテーブルには、いかなるローも直接挿入しないでください。

  参照：

  ○「ml_add_connection_script システムプロシージャ」648 ページ
  ○「ml_add_table_script システムプロシージャ」664 ページ
  ○「ml_add_user システムプロシージャ」665 ページ

- **タイムスタンプベースのダウンロード**　SAP HANA によってスナップショットアイソレーションがサポートされているので、Mobile Link サーバは現在のユーザの下で最初に開いたトランザクションの開始時刻を次の last_download_timestamp として使用します。この結果、download_cursor スクリプトと download_delete_cursor スクリプトでは、最後に変更したカラム情報を使用してタイムスタンプベースのダウンロードストリームを生成できます。次はその例です。

  1. SAP HANA で同期テーブルを作成します。

```
create COLUMN table test (pk int primary key, c1 int, last_modified timestamp generated
always as ( now() ) )
```

generated always as (now()) 句により、SAP HANA サーバは、このローが挿入または更新されるたびに、last_modified カラムを現在のタイムスタンプに更新します。

2. download_cursor スクリプトは、次のように作成できます。

```
select pk,c1 from test where last_modified > {ml s.last_table_download}
```

SAP HANA ではトリガがサポートされていないので、論理削除を使用して SAP HANA からリモートデータベースへのダウンロード削除を生成することが必要な場合があります。

あるいは、HISTORY COLUMN 句を使用してテーブルを作成すると、非表示のカラム $validto $ と $validfrom$ に基づいて download_cursor スクリプトと download_delete_cursor スクリプトを作成できます。次の last_download_timestamp は generate_next_last_download_timestamp スクリプトによって生成されます。このような非表示のカラムについては、SAP HANA のマニュアルを参照してください。

Mobile Link サーバが SAP HANA データベースにログインする場合に使用するユーザ以外のデータベースユーザにより所有される任意のテーブルに、download_cursor スクリプトおよび download_delete_cursor スクリプト、またはそのいずれかがアクセスする場合は、次の文を使用して Mobile Link サーバログインユーザに **CATALOG READ** 権限を付与する必要があります。

```
GRANT CATALOG READ TO MobiLink_server_login_user_name
```

こうすることで、Mobile Link サーバログインユーザは、データベースの最初に開いたトランザクションを取得しようとするときに、現在のユーザ用の開いているトランザクションだけでなく、データベース内で開いているすべてのトランザクションを確認できるようになります。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモートデータベース間で完全に一致する必要があります。 「SAP HANA データベースサーバのデータ型マッピング」751 ページを参照してください。

独立性レベル

「Mobile Link 独立性レベル」140 ページを参照してください。

SQL Anywhere 統合データベース

権限

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

SQL Anywhere を統合データベースとして設定する

SQL Anywhere を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *%SQLANY16%¥MobiLink¥setup* にある *syncsa.sql* 設定スクリプトを実行します。

- Sybase Central で Mobile Link のシステム設定を確認し、更新します。「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

ODBC ドライバの設定

SQL Anywhere 統合データベースには、ODBC DSN を設定する必要があります。SQL Anywhere 用の ODBC ドライバは、SQL Anywhere とともにインストールされます。

SQL Anywhere ODBC ドライバについては、「ODBC データソース」『SQL Anywhere サーバ データベース管理』を参照してください。

独立性レベル

「Mobile Link 独立性レベル」140 ページ を参照してください。

SAP Sybase IQ の統合データベース

権限

設定スクリプトを実行する前に、次の要件を確認してください。

- 設定スクリプトを実行するデータベースユーザは、同期中に Mobile Link システムテーブルを更新するユーザと同じである必要があります。このユーザを使用して Mobile Link サーバを起動したり、Mobile Link アプリケーションを設定したりします。 「必要な権限」30 ページを参照してください。

- Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システムテーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。 「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

- Mobile Link サーバのログイン ID には、SAP Sybase IQ の SP_IQTRANSACTION システムプロシージャに対する EXECUTE 権限が必要です。

SAP Sybase IQ を統合データベースとして設定する

SAP Sybase IQ を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システムテーブル、ストアドプロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *%SQLANY16%¥MobiLink¥setup* にある *synciq.sql* 設定スクリプトを実行します。
- Sybase Central で Mobile Link のシステム設定を確認し、更新します。「Mobile Link システム設定」『Mobile Link クイックスタート』を参照してください。

ODBC ドライバの設定

SAP Sybase IQ 統合データベースには、ODBC DSN を設定する必要があります。SAP Sybase IQ をインストールすると、Sybase IQ 用の ODBC ドライバもインストールされます。

SAP Sybase IQ 用の ODBC ドライバの詳細については、SAP Sybase IQ のマニュアルを参照してください。

SAP Sybase IQ の統合

- SAP Sybase IQ ストアに定義されている同期テーブルを変更するアップロードデータがアップロードに含まれている場合、および Mobile Link サーバが同時処理する複数のデータベースワーカスレッドで実行されている場合には、すべてのアップロードを直列化する必要があります。SAP Sybase IO 15 サーバでは、SAP Sybase IQ ストアの指定のテーブルを指定の時刻に修正する場合、それを 1 つの接続でしか行うことができないからです。

  次の SQL 文をインクルードまたは使用して begin_upload 接続を記述することによって、この要件が満たされます。

  ```
  LOCK TABLE table_name IN WRITE MODE WAIT time_string
  ```

  ここで、table_name は SAP Sybase IQ ストアに定義されたテーブルの名前を表しており、time_string はテーブルをロックする最長期間を表します。次のような簡単なテーブルを定義できます。

  ```
  create table coordinate_upload ( c1 int )
  ```

  テーブルにはデータがなくてもかまいません。

- (Mobile Link サーバ、またはその他の SAP Sybase IQ データベースへの接続で発生する) SAP Sybase IQ テーブルを変更するすべてのトランザクションはシリアル化する必要があります。Mobile Link サーバのトランザクションについて、前述したロジックを同じように使用できます。この方法は、それぞれのトランザクションについて Mobile Link サーバが自動的に再試行する場合より効率がよいため、パフォーマンスの向上につながります

- SAP Sybase IQ 統合データベースの同期モデルを作成する場合、SAP Sybase IQ テーブルのテーブルのマッピングのデフォルトはダウンロード専用となります。マッピングを双方向またはアップロード専用に変更した場合、それらの SAP Sybase IQ テーブルに対する変更がシリアル化されるようにする必要があります。たとえば、前述の説明のとおり begin_upload イベントを追加します。

独立性レベル

「Mobile Link 独立性レベル」140 ページを参照してください。

Mobile Link サーバ

すべての Mobile Link クライアントは、Mobile Link サーバを介して同期します。データベースサーバには、直接接続できません。Mobile Link サーバを起動してから、Mobile Link クライアントの同期を行います。

Mobile Link サーバは、統合データベースサーバとのデータベース接続を ODBC を介して開きます。その後、リモートアプリケーションからの接続を受け入れ、同期処理を制御します。

注意
mlsrv16 のオプションを指定すると、Mobile Link サーバの動作方法を指定できます。サーバの同期中の動作を制御するには、同期イベントで呼び出されるスクリプトを定義します。 「同期イベント」269 ページを参照してください。

Mobile Link サーバを起動するには、mlsrv16 を実行します。-c オプションを使用して統合データベースの ODBC 接続パラメータを指定します。 mlsrv16 コマンドラインオプションのリストは、「Mobile Link サーバオプション」43 ページを参照してください。

接続パラメータの指定は必須です。他のオプションは、必要に応じて使用します。これらのオプションを使用すると、サーバの動作方法を指定できます。たとえば、キャッシュサイズオプションとログオプションを指定できます。 接続パラメータの詳細については、「-c mlsrv16 オプション」52 ページを参照してください。

Mobile Link サーバには、統合データベースと通信するために、ODBC データソース名 (DSN) が必要です。DSN には、ODBC ドライバがロードされる ODBC ドライバマネージャに関する情報が含まれています。Windows では、Microsoft ODBC データソースアドミニストレータを使用して ODBC データソースを作成できます。Mobile Link サーバのビット設定と DSN のビット設定が一致している必要があります。具体的には、64 ビットの Mobile Link サーバでは、ODBC データソースアドミニストレータ (64 ビット) で作成された 64 ビットの DSN を使用する必要がります。 「レッスン 1 : Mobile Link 統合データベースの設定」『Mobile Link クイックスタート』を参照してください。

例

次のコマンドは、ODBC データソース *SQL Anywhere 16 CustDB* を使用して Mobile Link サーバを起動し、統合データベースを識別します。コマンド全体を 1 行に入力してください。

```
mlsrv16 -c "DSN=SQL Anywhere 16 CustDB;UID=ml_server;PWD=sql" -zs MyServer -o mlsrv.log -vcr -
x tcpip(port=3303)
```

この例では -c オプションで、ODBC データソース名 (DSN) と認証を含む接続文字列を指定しています。-zs オプションでサーバ名を指定しています。-o オプションは、ログファイル名を *mlsrv.log* と指定します。-vcr オプションが指定されているので、*mlsrv.log* の内容は冗長です。-x オプションを使用すると、バージョン 10 以降のクライアントのためのポートが開きます。

前の例で説明したオプションの詳細については、次の項を参照してください。

- 「-c mlsrv16 オプション」52 ページ
- 「-zs mlsrv16 オプション」104 ページ
- 「-o mlsrv16 オプション」65 ページ
- 「-v mlsrv16 オプション」85 ページ
- 「-x mlsrv16 オプション」92 ページ

Mobile Link サーバを Windows サービスまたは UNIX デーモンとして起動することもできます。「現在のセッション外での Mobile Link サーバの使用」33 ページを参照してください。

必要な権限

Mobile Link サーバをデータベースサーバに接続するには、データベースユーザを指定する必要があります。mlsrv16 -c オプションまたは ODBC データソースでデータベースユーザを指定します。 「-c mlsrv16 オプション」52 ページを参照してください。

このデータベースユーザには Mobile Link システムテーブルに対する完全な SELECT、INSERT、UPDATE、および DELETE 権限が必要です。また、Mobile Link システムプロシージャに対する EXECUTE ANY PROCEDURE 権限も必要です。デフォルトでは、Mobile Link 設定スクリプトを実行するデータベースユーザはこれらの権限を持っています。別のデータベースユーザを使用して Mobile Link サーバを実行する場合は、そのユーザに ml_* テーブルと ml_add_*_script システムプロシージャに対するこれらの権限を付与してください。

すべての Mobile Link システムプロシージャのリストについては、「Mobile Link サーバシステムプロシージャ」644 ページを参照してください。

このデータベースユーザは、Mobile Link スクリプトで参照されているすべてのテーブルに対する適切な権限と、Mobile Link スクリプトで参照されているプロシージャに対する EXECUTE 権限も必要です。

Mobile Link 統合データベースの中には、Mobile Link サーバによって使用されるデータベースユーザに、システムテーブルかビューまたはその両方に対する特定の権限を必要とするものもあります。特定の統合データベースの詳細については、次のトピックを参照してください。

- 「Adaptive Server Enterprise 統合データベース」
- 「IBM DB2 LUW 統合データベース」
- 「Microsoft SQL Server 統合データベース」
- 「MySQL 統合データベース」
- 「Oracle 統合データベース」

設定スクリプトの詳細については、「統合データベースの設定」3 ページを参照してください。

Mobile Link 接続

HTTP または HTTPS の使用時には、Relay Server の有無にかかわらず、Web ブラウザを使用して、Mobile Link サーバが要求を受信しているかどうかを確認できます。たとえば、Mobile Link サーバのコマンドラインが次のような場合：

```
mlsrv16 ... -x http(port=8080)
```

また、コンピュータが *ml1.mycorp.com* であれば、Web ブラウザを開いて **http://ml1.mycorp.com:8080** をポイントできます。

Mobile Link サーバは **404 Not Found** エラーで応答します。これには Mobile Link サーバの主要バージョンも記載されています。

「Relay Server の概要」『Relay Server』を参照してください。

Mobile Link サーバのシャットダウン

Mobile Link サーバは、サーバを起動したコンピュータから停止します。次の方法で、Mobile Link サーバを停止できます。

- Mobile Link 停止ユーティリティ (mlstop) を使用する
- Mobile Link サーバのメッセージウィンドウで **[シャットダウン]** をクリックする
- Windows のシステムトレイ内の Mobile Link サーバアイコンを右クリックして、**[シャットダウン]** をクリックする
- UNIX 上で実行中に Mobile Link サーバのメッセージウィンドウが開いていない状態で、Q と入力する
- Mobile Link サーバ API の shutdown メソッドを使用する

参照

- 「Mobile Link 停止ユーティリティ (mlstop)」692 ページ
- ServerContext.shutdown メソッド [Mobile Link サーバ Java]509 ページ
- ServerContext.Shutdown メソッド [Mobile Link サーバ .NET]603 ページ

Mobile Link サーバのロギング

サーバの動作をロギングすると、開発プロセスとトラブルシューティングのときに特に役立ちます。パフォーマンスが低下するため、運用環境の通常の操作には冗長出力を使用しないでください。

ファイルへのロギング結果の出力

選択したロギング結果は、Mobile Link サーバのメッセージウィンドウに送信されます。また、-o オプションを使用して結果をメッセージログファイルにも送信できます。次のコマンドでは、結果を *mlsrv.log* という名前のメッセージログファイルに送ります。

```
mlsrv16 -o mlsrv.log -c ...
```

次のオプションを使用して、ログファイルのサイズを制御したり、ファイルが最大サイズに達したときの処理を指定したりできます。

- ログファイルを使用することを指定する場合は -o オプションを使用します。

- -ot オプションを使用して、ログファイルを使用することを指定すると、メッセージが送信される前にログファイルの前の内容が削除されます。

- -o または -ot に加え、-on オプションを使用してサイズを指定すると、そのサイズに達したときに、拡張子 *.old* を付けてログファイルの名前が変更され、元の名前で新しいファイルが開始されます。このオプションは、メッセージログファイルが占める総ディスク容量を制限します。

- 古いログファイルが日付と連番に基づいた新しい名前を割り当てられたとき、-o または -ot の他に -os オプションを使用して古いログファイルのサイズを指定します。

次の項を参照してください。

- 「-o mlsrv16 オプション」65 ページ
- 「-on mlsrv16 オプション」66 ページ
- 「-os mlsrv16 オプション」67 ページ
- 「-ot mlsrv16 オプション」68 ページ

ロギング結果の出力容量の制御

-v オプションを使用すると、メッセージログファイルに記録され、Mobile Link サーバのメッセージウィンドウに表示される情報を制御できます。 「-v mlsrv16 オプション」85 ページを参照してください。

レポートされる警告メッセージの制御

レポートされる警告メッセージを制御できます。

詳細については、次の項を参照してください。

- 「-zw mlsrv16 オプション」105 ページ
- 「-zwd mlsrv16 オプション」106 ページ
- 「-zwe mlsrv16 オプション」106 ページ

Mobile Link サーバログの表示

Mobile Link ログは次の方法で表示できます。

- Mobile Link サーバのメッセージウィンドウの場合

- ログファイルを開く

- Sybase Central で **Mobile Link サーバログファイルビューア**を使用する

Mobile Link サーバメッセージウィンドウ外でログ情報を表示するには、情報をファイルに記録する必要があります。 「ファイルへのロギング結果の出力」31 ページを参照してください。

Mobile Link サーバログファイルビューア

Mobile Link サーバログを表示するには、Sybase Central を開き、**[ツール]** » **[Mobile Link 16]** » **[Mobile Link サーバログファイルビューア]** をクリックします。表示するログファイルを選択するよう要求されます。

Mobile Link サーバログファイルビューアは、Mobile Link ログファイルに格納された情報を読み取ります。Mobile Link サーバに接続したりログファイルの構成を変更したりすることはありません。

Mobile Link サーバログファイルビューアでは、表示する情報をフィルタリングできます。また、ログの情報に基づく統計情報も表示できます。

現在のセッション外での Mobile Link サーバの使用

Mobile Link サーバは、常時利用できるように設定できます。これを簡単に行うには、コンピュータをログオフしてもサーバは稼働し続けるように、Windows 版または UNIX 版の Mobile Link サーバを実行します。これを行う方法は、使用するオペレーティングシステムによって異なります。

- **UNIX デーモン**　mlsrv16 -ud オプションを使用すると、Mobile Link サーバをデーモンとして実行できます。これによって、Mobile Link サーバをバックグラウンドで実行し、ログオフ後も引き続き実行させることができます。

- **Windows サービス**　Windows Mobile Link サーバを、サービスとして実行させることができます。

  サービスとして実行されている Mobile Link サーバを停止するには、mlstop、dbsvc、または Windows サービスマネージャを使用できます。

参照

- 「-ud mlsrv16 オプション」84 ページ
- 「Windows でのサービスとしての Mobile Link サーバ」34 ページ
- 「Mobile Link 停止ユーティリティ (mlstop)」692 ページ
- 「Linux 用サービスユーティリティ (dbsvc)」『SQL Anywhere サーバ データベース管理』
- 「サービスまたはデーモンとしてデータベースサーバを実行する方法」『SQL Anywhere サーバ データベース管理』

デーモンとしての UNIX Mobile Link サーバの実行

UNIX において Mobile Link サーバをバックグラウンドで実行し、現在のセッションから独立して稼働させるには、Mobile Link サーバを「デーモン」として起動します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

- Mobile Link サーバの起動時に、-ud オプションを使用します。次に例を示します。

```
mlsrv16 -c "DSN=SQL Anywhere 16 Demo;UID=DBA;PWD=sql" -ud
```

結果

デーモンとして実行するような UNIX Mobile Link サーバ

参照

- 「-ud mlsrv16 オプション」84 ページ
- 「Linux 用サービスユーティリティ (dbsvc)」『SQL Anywhere サーバ データベース管理』

Windows でのサービスとしての Mobile Link サーバ

Windows Mobile Link サーバをバックグラウンドで実行し、現在のセッションから独立して稼働させるには、Mobile Link サーバを「サービス」として起動します。

コマンドラインから、または Sybase Central の **[サービス]** タブで、以下のサービス管理タスクを実行できます。

- サービスの追加、編集、削除。
- サービスの開始と停止。
- サービスを制御するパラメータの変更。

参照

- 「Windows 用サービスユーティリティ (dbsvc)」『SQL Anywhere サーバ データベース管理』

サービスの操作

Sybase Central を使用して新しいサービスを追加するか、既存のサービスを変更または削除します。サービス設定の変更は、次回のサービス実行時から有効となります。

前提条件

この作業を実行するための前提条件は、ありません。

内容と備考

Sybase Central のサービスアイコンは、サービスが実行中であるか停止されているかを示すアイコンを使用して、各サービスの現在の状況を表示します。

dbsvc ユーティリティを使用してサービスを作成することもできます。 「Windows 用サービスユーティリティ (dbsvc)」『SQL Anywhere サーバ データベース管理』を参照してください。

♦ タスク

1. Sybase Central の左ウィンドウ枠で [Mobile Link 16] をクリックします。
2. 右ウィンドウ枠で、**[サービス]** タブをクリックします。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

3. 右ウィンドウ枠で右クリックし、**[新規]** » **[サービス]** をクリックします。

   サービスを削除するには、サービスを選択し、**[編集]** » **[削除]** をクリックします。

   サービスのパラメータを変更するには、サービスを右クリックし、**[プロパティ]** をクリックします。

4. **サービス作成ウィザード**の指示に従います。

結果

サービスが追加、削除、または変更されます。

サービスの起動オプション

次のオプションは、Mobile Link サービスの起動時の動作を制御します。これらのオプションは、**[サービスのプロパティ]** ウィンドウの **[一般]** タブで設定できます。

- **[自動]**　**[自動]** を選択すると、サービスは Windows オペレーティングシステムが起動すると必ず起動します。この設定は、データベースサーバやその他の常に稼働しているアプリケーションに適しています。
- **[手動]**　**[手動]** を選択すると、サービスは Administrator 権限を持つユーザが起動したときにのみ起動します。Administrator 権限については、Windows のマニュアルを参照してください。
- **[無効]**　**[無効]** を選択すると、サービスは起動しません。

起動オプションは、次回 Windows を起動するときに適用されます。

コマンドラインオプション

[サービスのプロパティ] ウィンドウの **[設定]** タブには、実行プログラムのパスを入力する **[ファイル名]** テキストボックスと、サービスのコマンドラインオプションを入力する **[パラメータ]** テキストボックスがあります。**[パラメータ]** ボックスには、実行プログラムの名前は入力しないでください。

たとえば、冗長ロギングを指定して Mobile Link 同期サービスを開始するには、**[パラメータ]** ボックスに次のように入力します。

```
-c "DSN=SQL Anywhere 16 Demo;UID=DBA;PWD=sql"
-vc
```

サービスのコマンドラインオプションは、実行プログラムのものと同じです。「Mobile Link サーバオプション」43 ページを参照してください。

アカウントオプション

サービスが実行されるアカウントを選択できます。ほとんどのサービスは、特別なアカウントの LocalSystem で実行され、これがサービスのデフォルトオプションになっています。**[サービス**

のプロパティ] ウィンドウの **[アカウント]** タブを開き、アカウント情報を入力すると、別のアカウントでログオンするようにサービスを設定できます。

LocalSystem 以外のアカウントでサービスを実行するには、そのアカウントに「サービスとしてログオンする」権限が必要です。詳細なユーザ権限については、Microsoft Windows のマニュアルを参照してください。

サービスのアイコンがタスクバーまたはデスクトップに表示されるかどうかは、次のように、選択したアカウントと、**[デスクトップとの対話をサービスに許可]** チェックボックスが選択されているかどうかによって決まります。

- サービスが LocalSystem で実行されているときに、**[サービスのプロパティ]** ウィンドウの **[デスクトップとの対話をサービスに許可]** チェックボックスがオンになっている場合は、サービスを実行するコンピュータ上の Windows XP/2003 にどのユーザがログインしてもデスクトップにアイコンが表示されます。すべてのユーザがアプリケーションウィンドウを開き、サービスとして実行されているプログラムを停止できます。

- サービスが LocalSystem で実行されているときに、**[サービスのプロパティ]** ウィンドウの **[デスクトップとの対話をサービスに許可]** チェックボックスがクリアされている場合は、ユーザのデスクトップにアイコンは表示されません。サービスの状態を変更する権限を与えられているユーザのみ、サービスを停止できます。

- サービスが他のアカウントで実行されている場合、デスクトップにアイコンは表示されません。サービスの状態を変更する権限を与えられているユーザのみ、サービスを停止できます。

実行ファイルの変更

Sybase Central でサービスに関連付けられたプログラム実行ファイルを変更できます。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

実行ファイルを新しいディレクトリに移動する場合は、この内容も変更します。

♦ タスク

1. **[サービスプロパティ]** ウィンドウで **[設定]** タブをクリックします。
2. **[ファイル名]** ボックスに新しいパスとファイル名を入力します。

結果

サービスと関連付けられたプログラム実行ファイルが更新されます。

サービスの開始と停止

サービスの開始と停止

サービスを開始すると、停止するまで実行を続けます。Sybase Central を閉じたり、ログオフしてもサービスは停止しません。

前提条件

この作業を実行するための前提条件は、ありません。

内容と備考

サービスを停止すると、すべてのネットワーク接続が閉じられ、Mobile Link サーバは終了します。他のアプリケーションのプログラムは終了します。

♦ サービスの開始または停止

1. Sybase Central の左ウィンドウ枠で [Mobile Link 16] をクリックし、右ウィンドウ枠で **[サービス]** タブを開きます。
2. サービスを右クリックし、**[起動]** または **[停止]** をクリックします。

結果

サービスが開始または停止します。

複数のサービス

Windows の **[コントロール パネル]** にあるサービス マネージャを使用していくつかのタスクを実行することはできますが、Windows のサービス マネージャで Mobile Link サービスをインストールしたり設定したりすることはできません。Mobile Link のすべてのサービス管理は、Sybase Central から行うことができます。

Windows の [コントロール パネル] から [サービス マネージャ] を開くと、サービスのリストが表示されます。SQL Anywhere サービスの名前は、サービスをインストールしたときに入力したサービス名に、SQL Anywhere がプレフィクスを付けたものです。インストールされたすべてのサービスが、リストに表示されます。

サービスの依存性

状況によっては、複数の実行プログラムをサービスとして実行する必要があり、これらの実行プログラムが相互に依存する場合があります。たとえば、Mobile Link サーバを実行することと、同期する統合データベースサーバを実行することが必要な場合です。

サービスは正しい順序で開始する必要があります。Mobile Link 同期サービスは、統合データベースサーバの起動が完了する前に開始されると、統合データベースサーバを検出できないためエラーになります。Mobile Link サーバを起動するときにデータベースサーバが実行されているような順序にしてください。(これは、統合データベースサーバが別のコンピュータで実行されている場合は適用されません。)

サービスグループを使用すると、これらの問題を防ぐことができます。サービスグループは Sybase Central で管理します。 「サービスまたはグループの依存性のリストへの追加」38 ページを参照してください。

サービスグループ

システムの各サービスを、サービスグループの各メンバーに割り当てることができます。デフォルトでは、各サービスはグループに属しています。Mobile Link サーバのデフォルトのグループは、SQLANYMobiLink です。 「サービスが割り当てられているグループの調査と変更」38 ページを参照してください。

サービスが割り当てられているグループの調査と変更

サービスが適切なグループのメンバーであることをチェックしてから、サービスが正しい順序で開始するように設定してください。Sybase Central では、サービスが割り当てられているグループを調べ、このグループを変更できます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Sybase Central の左ウィンドウ枠で [Mobile Link 16] をクリックし、右ウィンドウ枠で **[サービス]** タブを開きます。
2. サービスを右クリックし、**[プロパティ]** をクリックします。
3. **[依存性]** タブをクリックします。最上部のテキストボックスにサービスが割り当てられているグループの名前が表示されます。
4. **[変更]** をクリックして、システムで使用可能なグループのリストを表示します。
5. いずれかのグループを選択するか、新しいグループの名前を入力します。
6. **[OK]** をクリックして、そのグループにサービスを割り当てます。

結果

サービスまたはグループが依存性リストに追加されます。

次の手順

サービスが確実に正しい順序で開始するように設定できます。 「サービスまたはグループの依存性のリストへの追加」38 ページを参照してください。

サービスまたはグループの依存性のリストへの追加

Sybase Central を使用して、サービスの依存性を指定できます。たとえば、1 つ以上のグループを開始してから現在のサービスを開始するようにできます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Sybase Central の左ウィンドウ枠で [Mobile Link 16] をクリックし、右ウィンドウ枠で **[サービス]** タブを開きます。
2. サービスを右クリックし、**[プロパティ]** をクリックします。
3. **[依存性]** タブをクリックします。
4. **[サービスの追加]** または **[サービスグループの追加]** をクリックして、サービスまたはグループを依存性リストに追加します。
5. リストからサービスまたはグループを 1 つ選択します。
6. **[OK]** をクリックして、そのサービスまたはグループを依存性リストに追加します。

結果

サービスまたはグループが依存性のリストに追加されます。

サーバファーム内の Mobile Link サーバ

注意

別途ライセンスが必要な必須コンポーネント　サーバファームでの Mobile Link サーバの実行は、Mobile Link の高可用性オプションの機能であり、別途ライセンスが必要です。 「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

Mobile Link サーバファームは、同じリモートデータベースのセットを同じ統合データベースと同期する複数の Mobile Link サーバがある環境です。これは多くの場合、大規模な配備やフェイルオーバ機能のために必要になります。このような Mobile Link サーバファーム配備では、HTTP 通信リンクを使用する場合は、Relay Server の使用が必要になることがあります。TCP ベースのストリームでは、TCP 負荷分散装置が機能している必要があります。複数のサーバを使用している場合、再起動可能なダウンロードは機能しません。

サーバによって開始された同期で Notifier を使用する場合は、-lsc オプションを使用してローカルサーバ接続設定を指定します。これらの設定はファーム内の他のサーバに渡されるため、サーバが相互に接続して通知の処理を共有できます。たとえば、ホスト **farm_host22** で実行している場合は、次のようになります。

```
mlsrv16 -x tcpip(port=3245) -zs server5 -lsc tcpip(host=farm_host22;port=3245) -c ...
```

Mobile Link 監視サーバ

サーバ起動同期で Mobile Link サーバファームを実行する場合は、Mobile Link 監視サーバを使用して、ファーム内に常に 1 台のプライマリサーバが存在するようにします。常にプライマリサーバが存在することで、冗長な通知やメッセージの消失が防止されます。

「mlarbiter」 コマンドを使用して Mobile Link 監視サーバを起動し、Mobile Link サーバの -ca オプションを -lsc オプションとともに使用して、監視サーバ情報を持つ Mobile Link サーバを起動します。

参照

- 「Windows 用 Mobile Link 監視サーバユーティリティ (mlarbiter)」705 ページ
- 「-lsc mlsrv16 オプション」63 ページ
- 「-ca mlsrv16 オプション」52 ページ

Mobile Link サーバでのメモリの使用

Mobile Link サーバは、主に次の方法でメモリを使用します。

- キャッシュ
- VM
- システムデータの格納

キャッシュ

標準的な Mobile Link サーバでは、メモリの大部分はメモリキャッシュによって使用されます。メモリキャッシュにはローデータ、関連するデータ構造、ネットワークバッファが格納されます。通常、サイズまたは量の面で無限のデータはキャッシュに格納されます。メモリキャッシュに格納するデータの量がキャッシュサイズを超える場合、余分なデータはディスクに転送されます。これは同期のパフォーマンスのボトルネックになる可能性があります。ディスクへのスワップがどの程度問題となるかは、ディスク I/O の量と頻度、およびディスクの速度によって異なります。この潜在的なボトルネックを排除するために、データがディスクに一切スワップされないようにすることをおすすめします。Mobile Link サーバログの警告 10082 または SQL Anywhere モニタの警告「キャッシュがいっぱいです」に注意してください。

デフォルトでは、Mobile Link サーバは利用できるプロセスアドレス領域の 60 % までキャッシュを自動的に拡大し、システムの他のプロセスがより多くのメモリを必要とした場合、および、サーバのキャッシュ以外のメモリが増大した場合に、キャッシュを縮小します。キャッシュの初期サイズ、最大サイズ、最小サイズは、-cinit, -cmax、-cmin の各 mlsrv16 サーバオプションを使用して制御できます。必要に応じて、-cmax と -cmin のオプション値に同じ値を指定して、動的キャッシュサイズ決定を無効にできます。

次の項を参照してください。

- 「-cmax mlsrv16 オプション」54 ページ
- 「-cmin mlsrv16 オプション」54 ページ
- 「-cinit mlsrv16 オプション」53 ページ

キャッシュに関する情報を確認するには、次の複数の方法があります。

- **[サーバキャッシュサイズ]** これは現在のキャッシュサイズを示す SQL Anywhere モニタのメトリックです。「Mobile Link サーバリソースのメトリックのリスト」『SQL Anywhere サーバ データベース管理』を参照してください。
- **[ロックされているページ (%)]** これはキャッシュ内のディスクへ転送できないページの割合を示す SQL Anywhere モニタのメトリックです。「Mobile Link サーバリソースのメトリックのリスト」『SQL Anywhere サーバ データベース管理』を参照してください。
- **[使用済みページ (%)]** これは有効なデータを含むページの割合を示す SQL Anywhere モニタのメトリックです。「Mobile Link サーバリソースのメトリックのリスト」『SQL Anywhere サーバ データベース管理』を参照してください。
- **PAGES_LOCKED_MAX** mlsrv16 の -ppv オプションでは、メモリキャッシュ内のページ数を出力できます。「-ppv mlsrv16 オプション」68 ページを参照してください。
- **PAGES_LOCKED** mlsrv16 の -ppv オプションでは、メモリにロードされたキャッシュページ数を出力できます。「-ppv mlsrv16 オプション」68 ページを参照してください。
- **PAGES_USED** mlsrv16 の -ppv オプションでは、使用されたキャッシュページ数を出力できます。「-ppv mlsrv16 オプション」68 ページを参照してください。
- **の -vk オプション mlsrv16** キャッシュが増大または縮小すると、サーバがログに情報を出力するようにします。「-ppv mlsrv16 オプション」68 ページを参照してください。

VM

Mobile Link サーバ内のメモリのもう 1 つの主な用途は、組み込みの Java VM と .NET VM です。ダイレクトロー API が多用される配備、または JBoss や Tomcat などのアプリケーションサーバを組み込む配備では、VM メモリの使用がメモリキャッシュを超える場合があります。

Java VM を使用する場合、通常は VM が使用するメモリの量を制御できます。ほとんどの Java VM に用意されている -Xms オプションと -Xmx オプションを使用すると、VM のヒープの最大サイズと初期サイズを指定できます。

[VM メモリ使用率] SQL Anywhere モニタメトリック、または **VM_MEM_USE** -ppv 値を使用して、組み込みの VM によって使用されているメモリ量を確認できます。

次の項を参照してください。

- 「Mobile Link サーバリソースのメトリックのリスト」『SQL Anywhere サーバ データベース管理』
- 「-ppv mlsrv16 オプション」68 ページ

その他のメモリ用途

残りのメモリは、各同期、スレッドスタック、スレッド間通信、ODBC ドライバに関するステータス情報の格納に使用されます。通常、これはメモリの数 10 メガバイトを占め、使用される総量は mlsrv16 の -nc、-sm、-w オプションによって制限されます。また、SQL Anywhere 以外の ODBC ドライバが、特に BLOB カラムの処理時に大量のメモリを使用する場合があることに注意してください。

次の項を参照してください。

- 「-nc mlsrv16 オプション」64 ページ
- 「-sm mlsrv16 オプション」79 ページ
- 「-w mlsrv16 オプション」89 ページ

Mobile Link サーバ起動時のトラブルシューティング

この項では、Mobile Link サーバの起動時に発生する一般的な問題について説明します。

ネットワーク通信ソフトウェアが実行されているかを確認する

適切なネットワーク通信ソフトウェアをインストールし、実行してから、Mobile Link サーバを実行します。信頼性の高いネットワークソフトウェアを 1 つのネットワークだけが導入された状態で実行している場合は、問題はありません。Mobile Link サーバを実行する前に、ネットワーク通信を必要とする他のソフトウェアが正しく動作していること確認してください。

ping と telnet が正しく動作することを確認することをおすすめします。ping アプリケーションと telnet アプリケーションは、多くの TCP/IP プロトコルスタックに付属します。

ネットワーク通信の起動時の問題をデバッグする

ネットワークで接続を確立するときに問題が生じた場合は、クライアントとサーバの両方でデバッグオプションを使用し、問題点を診断できます。起動情報は、Mobile Link サーバメッセージウィンドウに表示されます。-o オプションを使用して、結果を出力ファイルに記録することもできます。

「Mobile Link サーバのロギング」31 ページを参照してください。

Mobile Link サーバ接続の確認

HTTP または HTTPS の使用時には、Relay Server の有無にかかわらず、Web ブラウザを使用して、Mobile Link サーバが要求を受信しているかどうかを確認できます。たとえば、Mobile Link サーバのコマンドラインが次のような場合：

```
mlsrv16 ... -x http(port=8080)
```

また、コンピュータが *ml1.mycorp.com* であれば、Web ブラウザを開いて **http://ml1.mycorp.com:8080** をポイントできます。

Mobile Link サーバは **404 Not Found** エラーで応答します。これには Mobile Link サーバの主要バージョンも記載されています。

「Relay Server の概要」『Relay Server』を参照してください。

Mobile Link サーバオプション

mlsrv16 構文

Mobile Link サーバを起動します。

構文

mlsrv16 -c "*connection-string***"** [*options*]

オプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込みます。 「@data mlsrv16 オプション」49 ページを参照してください。
-a	統合データベース接続で同期エラーが発生した後もその接続を使用し続けます。 「-a mlsrv16 オプション」50 ページを参照してください。
-b	文字列のブランクの埋め込みを削除します。 「-b mlsrv16 オプション」50 ページを参照してください。
-bn *size*	競合の検出のため BLOB を比較するときに考慮する最大バイト数を指定します。 「-bn mlsrv16 オプション」51 ページを参照してください。
-c *"keyword=value; ..."*	統合データベース用の ODBC データベース接続パラメータを指定します。このオプションは必須です。「-c mlsrv16 オプション」52 ページを参照してください。
-ca *host_or_ip*	Mobile Link 監視サーバのホスト名または IP アドレスを設定します。「-ca mlsrv16 オプション」52 ページを参照してください。
-cinit *size*	サーバのメモリキャッシュの初期サイズを指定します。 「-cinit mlsrv16 オプション」53 ページを参照してください。
-cm *size*	サーバのメモリキャッシュのサイズを指定します。「-cm mlsrv16 オプション」54 ページを参照してください。
-cmax *size*	サーバのメモリキャッシュの最大サイズを指定します。 「-cmax mlsrv16 オプション」54 ページを参照してください。

オプション	説明
-cmin *size*	サーバのメモリキャッシュの最小サイズを指定します。 「-cmin mlsrv16 オプション」 54 ページを参照してください。
-cn *connections*	統合データベースサーバとの最大同時接続数を設定します。 「-cn mlsrv16 オプション」 55 ページを参照してください。
-cr *count*	データベース接続リトライの最大回数を設定します。 「-cr mlsrv16 オプション」 56 ページを参照してください。
-cs *"keyword=value; ..."*	Mobile Link システムデータベース (MLSD) 用のシステムデータベース接続パラメータを指定します。 「-cs mlsrv16 オプション」 56 ページを参照してください。
-ct *connection-timeout*	接続が使用されなくなってからタイムアウトになるまでの時間を設定します。 「-ct mlsrv16 オプション」 57 ページを参照してください。
-dl	Mobile Link サーバのメッセージウィンドウにすべてのログメッセージを表示します。 「-dl mlsrv16 オプション」 57 ページを参照してください。
-dr	Adaptive Server Enterprise の場合のみ。同期に関係するテーブルで、DataRow ロックスキームを使用しないようにします。 「-dr mlsrv16 オプション」 58 ページを参照してください。
-ds *size*	すべての再起動可能なダウンロードで使用される保存できるデータの最大量を指定します。 「-ds mlsrv16 オプション」 58 ページを参照してください。
-dsd	SQL Anywhere と Microsoft SQL Server 統合データベースのデフォルトのダウンロード独立性レベルであるスナップショットアイソレーションを無効にします。 「-dsd mlsrv16 オプション」 59 ページを参照してください。
-dt	現在のデータベース内のみでトランザクションを検出します。 「-dt mlsrv16 オプション」 59 ページを参照してください。

オプション	説明
-e *filename*	送信されたリモートデータベースのエラーログを指定のファイルに格納します。 「-e mlsrv16 オプション」 60 ページを参照してください。
-esu	アップロードにスナップショットアイソレーションを使用します。 「-esu mlsrv16 オプション」 61 ページを参照してください。
-et *filename*	指定のファイルが存在する場合は、最初に内容を削除してから、送信されたリモートデータベースのエラーログをそのファイルに格納します。 「-et mlsrv16 オプション」 61 ページを参照してください。
-fips	すべての Mobile Link セキュアストリームを強制的に FIPS 認定とします。 「-fips mlsrv16 オプション」 62 ページを参照してください。
-ftr *path*	Mobile Link 転送ユーティリティ (mlfiletransfer) によってダウンロードされるファイル用のロケーションを指定します。 「-ftr mlsrv16 オプション」 62 ページを参照してください。
-ftru	Mobile Link ファイル転送ユーティリティ (mlfiletransfer) によってアップロードされたファイル用のロケーションを指定します。「-ftru mlsrv16 オプション」 63 ページを参照してください。
-lsc *protocol*[*protocol-options*]	ローカルサーバの接続情報を指定します。 「-lsc mlsrv16 オプション」 63 ページを参照してください。
-nc *connections*	同時ネットワーク接続の最大数を設定します。 「-nc mlsrv16 オプション」 64 ページを参照してください。
-notifier	サーバ起動同期用に Notifier を起動します。 「-notifier mlsrv16 オプション」 65 ページを参照してください。
-o *logfile*	ファイルにメッセージのログを取ります。 「-o mlsrv16 オプション」 65 ページを参照してください。

オプション	説明
-on *size*	ログファイルの最大サイズを設定します。 「-on mlsrv16 オプション」66 ページを参照してください。
-oq	起動エラーの発生時にポップアップウィンドウを表示しません。 「-oq mlsrv16 オプション」67 ページを参照してください。
-os *size*	古いログファイルの最大サイズ。 「-os mlsrv16 オプション」67 ページを参照してください。
-ot *logfile*	最初に内容を削除してから、ファイルにメッセージのログを取ります。 「-ot mlsrv16 オプション」68 ページを参照してください。
-ppv *period*	指定された期間に従って、Mobile Link が新しい定期モニタリングの値を出力するようにします。 「-ppv mlsrv16 オプション」68 ページを参照してください。
-q	Mobile Link サーバのメッセージウィンドウを最小化します。 「-q mlsrv16 オプション」73 ページを参照してください。
-r *retries*	デッドロックされたアップロードを、この回数までリトライします。 「-r mlsrv16 オプション」73 ページを参照してください。
-rd *delay*	デッドロックされたトランザクションをリトライするまでの最大遅延秒数を設定します。 「-rd mlsrv16 オプション」74 ページを参照してください。
-rp *directory*	mlreplay ユーティリティでプレイバックする同期の記録先ディレクトリを指定します。 「-rp mlsrv16 オプション」75 ページを参照してください。
-rrp *directory*	Mobile Link サーバが起動時に、指定されたディレクトリで mlreplay ユーティリティを実行するようにします。 「-rrp mlsrv16 オプション」74 ページを参照してください。
-s *count*	一度に統合データベースにアップロードされるロー、または統合データベースからフェッチされるローの最大数を指定します。 「-s mlsrv16 オプション」75 ページを参照してください。

オプション	説明
-sl dnet *script-options*	.NET Common Language Runtime (CLR) オプションを設定し、起動時に CLR を強制的にロードします。「-sl dnet mlsrv16 オプション」76 ページを参照してください。
-sl java *script-options*	Java VM のオプションを設定し、起動時に Java VM を強制的にロードします。「-sl java mlsrv16 オプション」78 ページを参照してください。
-sm *number*	同時に動作できる同期の最大数を設定します。「-sm mlsrv16 オプション」79 ページを参照してください。
-tc *minutes*	SQL スクリプト実行のタイムアウトスレッショルドを設定します。「-tc mlsrv16 オプション」80 ページを参照してください。
-tf	タイムアウトスレッショルドに達したら、SQL スクリプトを実行できないようにします (Oracle は対象外)。「-tf mlsrv16 オプション」81 ページを参照してください。
-ts *session-name*(*session-option*=[*option-value*;...])	Mobile Link サーバのトレースセッションを設定します。「-ts mlsrv16 オプション」82 ページを参照してください。
-tx *count*	アップロードのトランザクションで、トランザクションのグループをバッチにし、まとめてコミットします。「-tx mlsrv16 オプション」83 ページを参照してください。
-ud	UNIX プラットフォーム上でデーモンとして実行します。「-ud mlsrv16 オプション」84 ページを参照してください。
-ui	X-Window がサポートされている Linux で、使用可能な表示がない場合にシェルモードで Mobile Link サーバを起動します。「-ui mlsrv16 オプション」84 ページを参照してください。
-ux	X-Window がサポートされている Linux で、Mobile Link サーバのメッセージウィンドウを開きます。「-ux mlsrv16 オプション」85 ページを参照してください。

オプション	説明
-v [*levels*]	ログファイルに書き込まれるメッセージのタイプを制御します。 「-v mlsrv16 オプション」 85 ページを参照してください。
-w *count*	データベースワーカスレッドの初期数を設定します。 「-w mlsrv16 オプション」 89 ページを参照してください。
-wm *count*	データベースワーカスレッドの最大数を設定します。 「-wm mlsrv16 オプション」 90 ページを参照してください。
-wn *count*	ネットワークストリームの同時処理のためにネットワークワーカスレッドの数を設定します。 「-wn mlsrv16 オプション」 80 ページを参照してください。
-wu *count*	アップロード処理の同時実行を許可されるデータベースワーカスレッドの最大数を設定します。「-wu mlsrv16 オプション」 91 ページを参照してください。
-x *protocol"options; ..."*	通信プロトコルを指定します。必要に応じて、*parameter=value* の形式でネットワークパラメータを指定します。複数のパラメータを指定する場合はセミコロンで区切ります。 「-x mlsrv16 オプション」 92 ページを参照してください。
-zf	各同期の始めに Mobile Link サーバがスクリプトの変更をチェックするように指定します。 「-zf mlsrv16 オプション」 103 ページを参照してください。
-zp	リモートデータベースと統合データベースの精度が異なる場合、タイムスタンプ値間の明らかな差異を無視します。 「-zp mlsrv16 オプション」 103 ページを参照してください。
-zs *name*	サーバ名を指定します。 「-zs mlsrv16 オプション」 104 ページを参照してください。
-zt *number*	Mobile Link サーバを実行するのに使用されるプロセッサの最大数を指定します。 「-zt mlsrv16 オプション」 104 ページを参照してください。

オプション	説明
-zu { + | - }	authenticate_user スクリプトが未定義の場合に、ユーザの自動的な追加を制御します。 「-zu mlsrv16 オプション」104 ページを参照してください。
-zus	Mobile Link がアップロードのないテーブルのアップロードスクリプトを呼び出すようにします。 「-zus mlsrv16 オプション」105 ページを参照してください。
-zw 1,...5	表示する警告メッセージのレベルを制御します。 「-zw mlsrv16 オプション」105 ページを参照してください。
-zwd *code*	特定の警告コードを無効にします。 「-zwd mlsrv16 オプション」106 ページを参照してください。
-zwe *code*	特定の警告コードを有効にします。 「-zwe mlsrv16 オプション」106 ページを参照してください。

備考

Mobile Link サーバは、統合データベースサーバとの接続を ODBC を介して開きます。その後、クライアントアプリケーションからの接続を受け入れ、同期処理を制御します。

-c オプションを使用して統合データベースの接続パラメータを指定してください。コマンドラインオプションは、任意の順序で指定できます。ここでは、便宜上、-c オプションをコマンド文字列の最初の項目としています。接続文字列の前であれば、オプションリストのどこにでも指定できます。

ODBC データソースが統合データベースを自動的に起動するように設定していない場合には、統合データベースを実行してから Mobile Link サーバを起動してください。

参照

- 「Mobile Link サーバ」29 ページ

@data mlsrv16 オプション

指定された環境変数または設定ファイルからオプションを読み込みます。

構文

mlsrv16 -c "*connection-string***" @***data* ...

備考

このオプションを使用すると、指定された環境変数または設定ファイルから mlsrv16 コマンドラインオプションを読み込むことができます。指定された名前と同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。

設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化します。

参照

- 「設定ファイル」『SQL Anywhere サーバ データベース管理』
- 「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』

-a mlsrv16 オプション

Mobile Link サーバに対して、統合データベース接続で同期エラーが発生した後もその接続を使用し続けるように指示します。

構文

mlsrv16 -c "*connection-string***" -a** ...

備考

デフォルトでは、同期中にエラーが発生すると、Mobile Link サーバは自動的に統合データベースとの接続を切断してから、接続を再確立します。再接続が行われると、認識されている状態から確実に次の同期が開始します。このような動作が不要な場合は、このオプションを使用して無効にしてください。ステータス情報の管理は、プログラマーの要求や、RDBMS を処理する Mobile Link スクリプトの設定方法に応じて異なります。このことは、Oracle や SQL Anywhere のデータベース、またはサポートされている他の製品にもあてはまります。クライアントアプリケーションによっては、一部のステータス情報を初期化し直す操作が必要になることがあります。

-b mlsrv16 オプション

VARCHAR、CHAR、LONG VARCHAR、または LONG CHAR 型のカラムの場合、同期中に文字列から後続ブランクを削除します。

構文

mlsrv16 -c "*connection-string***" -b** ...

備考

> **注意**
> この問題が発生しないように、統合データベースで CHAR の代わりに VARCHAR を使用することをおすすめします。

このオプションを使用すると、SQL Anywhere の CHAR データ型と統合データベースが使用する CHAR または VARCHAR データ型の違いを解決できます。SQL Anywhere の CHAR データ型

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

は VARCHAR と同じです。しかし、SQL Anywhere ではないほとんどの統合データベースで、CHAR(n) データ型は n 文字までブランクが埋め込まれます。

-b が指定されていると、Mobile Link サーバは、リモートデータベースのカラムが文字列の場合に CHAR、VARCHAR、LONG CHAR、または LONG VARCHAR 型のカラムに対して文字列から後続ブランクを削除します。トリムされたデータは、次にリモートデータベースへダウンロードされます。

また、このオプションを使用すると、upload_fetch または upload_fetch_column_conflict スクリプトが使用されるとき、競合する更新を適切に検出することもできます。各アップロード更新ローに対して、Mobile Link サーバは統合データベースから指定されたプライマリキーのローをフェッチし、そのローを更新前イメージと比較して、この更新が競合する更新であるかどうかを判断します。-b を使用する場合、Mobile Link は CHAR、VARCHAR、LONG CHAR、または LONG VARCHAR 型のカラムから後続ブランクを消してから、この比較を行います。

参照

- 「CHAR カラム」5 ページ
- 「NVARCHAR データ型」『SQL Anywhere サーバ SQL リファレンス』
- 「upload_fetch テーブルイベント」435 ページ
- 「upload_fetch_column_conflict テーブルイベント」437 ページ

例

-b オプションを使用しない場合、SQL Anywhere または Ultra Light リモートデータベースから統合データベースの CHAR(10) カラムにアップロードされるプライマリキー値 'abc' は、'abc' の後に 7 個のブランクスペースが続く値になります。同じローがダウンロードされると、リモートデータベースでは 'abc' の後に 7 個のブランクスペースが続く値として扱われます。リモートデータベースがブランクを埋め込まない場合、リモートデータベースは 2 つのローを格納します。それは、「abc」というローと「abc」の後に 7 個のブランクスペースが続くローです。そのため、リモートで重複するローが存在することになります。

-b オプションを使用する場合、SQL Anywhere または Ultra Light リモートデータベースから統合データベースの CHAR(10) カラムにアップロードされるプライマリキー値 'abc' は、'abc' の後に 7 個のスペースが続く値になります。7 個のスペースが埋め込まれた値は 10 個の文字になりますが、同じローがダウンロードされるときに Mobile Link サーバは後続のスペースを削除するため、その値はリモートデータベースでは 'abc' として扱われます。このように、-b オプションは重複するローの問題を解決します。

-bn mlsrv16 オプション

競合検出中に比較する BLOB の最大バイト数を設定します。

構文

mlsrv16 -c "*connection-string***" -bn** *size* ...

備考

2 つの BLOB が類似する値または同じ値で構成されている場合は、処理対象のデータの量が多くなるため、フィルタや競合検出などで両者の比較操作が高負荷になることがあります。このオプ

ションを指定すると、Mobile Link サーバは、*size* で指定された最初のバイト数だけを比較対象とします。デフォルトでは、2 つの BLOB 全体を比較します。

比較するデータの最大量を制限することで、同期の速度をかなり上げることができますが、エラーが発生する可能性もあります。たとえば、2 つの大きな BLOB の最後の数バイトだけが異なる場合、Mobile Link サーバは、実際には異なるのに、同一であるとみなす可能性があります。

-c mlsrv16 オプション

統合データベースの接続パラメータを指定します。

構文

mlsrv16 -c "*connection-string***"** ...

備考

接続文字列には、Mobile Link サーバが統合データベースに接続するために十分な情報を指定します。接続文字列は必須です。

接続文字列では、接続パラメータを *keyword=value* の形式で指定します。パラメータとパラメータの間はセミコロンで区切り、スペースは入れないでください。

接続パラメータをコマンドラインで指定しない場合は、ODBC データソースの設定に含めてください。RDBMS と ODBC データソースを調べて、必要な接続データを判断してください。

SQL Anywhere 接続パラメータの完全なリストについては、「接続パラメータ」『SQL Anywhere サーバ データベース管理』を参照してください。

パスワードを非表示にする方法については、「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。

例

```
mlsrv16 -c "DSN=odbcname;UID=DBA;PWD=sql"
```

-ca mlsrv16 オプション

Mobile Link 監視サーバが実行されている場所を Mobile Link サーバが認識できるように Mobile Link 監視サーバのホスト名または IP アドレスを設定します。

構文

mlsrv16 -c "*connection-string***" -ca** *host_or_ip* ...

備考

同じサーバファーム内のすべての Mobile Link サーバに -ca オプションの同じ設定が含まれている必要があります。

ローカル Mobile Link サーバの接続文字列を指定するには、-ca オプションとともに -lsc オプションも使用します。

-ca および -lsc コマンドラインオプションは、コマンドラインに -notifier が含まれていない場合、Mobile Link サーバによって無視されます。

> **注意**
> ポート 4953 は Mobile Link 監視サーバに割り当てられているため、Mobile Link 監視サーバを実行しているコンピュータ上の他のアプリケーションはこのポート番号を使用できません。

参照

- 「-lsc mlsrv16 オプション」63 ページ
- 「-notifier mlsrv16 オプション」65 ページ

-cinit mlsrv16 オプション

サーバのメモリキャッシュの初期サイズを設定します。

構文

mlsrv16 -c "*connection-string***" -cinit** *size***[k | m | g | p]** ...

備考

テーブルデータ、ネットワークバッファ、キャッシュされたダウンロードデータ、同期に使用されるその他の構造体を保持するためにサーバが使用するメモリの初期容量を指定します。このメモリプールに保持できる量を超えるデータがサーバにある場合、データはディスクにスワップされます。データがディスクにスワップされると、パフォーマンスが大幅に低下します。

size には、予約するメモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。数字の後に文字を指定しない場合、サイズはバイト単位になります。

単位 **p** は、物理システムメモリとプロセスアドレス可能領域のうち、いずれか小さい方のパーセンテージを表します。プロセスアドレス可能領域の最大サイズはオペレーティングシステムによって異なります。次に例を示します。

- 2.5 GB - Windows 32 ビット Advanced Server、Enterprise Server、Datacenter Server
- 3.5 GB - Windows x64 Edition 上の 32 ビットデータベースサーバ
- 1.5 GB - その他すべての 32 ビットシステム
- 64 ビットのデータベースサーバの場合、キャッシュサイズは無制限と見なされる

デフォルトは **50m** です。

参照

- 「-cmax mlsrv16 オプション」54 ページ
- 「-cmin mlsrv16 オプション」54 ページ
- 「Mobile Link サーバでのメモリの使用」40 ページ

-cm mlsrv16 オプション

サーバのメモリキャッシュの最大サイズを設定します。このオプションは -cmax オプションのエイリアスであることに注意してください。 「-cmax mlsrv16 オプション」54 ページを参照してください。

-cmax mlsrv16 オプション

サーバのメモリキャッシュの最大サイズを設定します。

構文

mlsrv16 -c "*connection-string***" -cmax** *size***[k | m | g | p]** ...

備考

テーブルデータ、ネットワークバッファ、キャッシュされたダウンロードデータ、同期に使用されるその他の構造体を保持するためにサーバが使用するメモリの最大容量を指定します。このメモリプールに保持できる量を超えるデータがサーバにある場合、データはディスクにスワップされます。データがディスクにスワップされると、パフォーマンスが大幅に低下します。

size には、予約するメモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。数字の後に文字を指定しない場合、サイズはバイト単位になります。

単位 **p** は、物理システムメモリとプロセスアドレス可能領域のうち、いずれか小さい方のパーセンテージを表します。プロセスアドレス可能領域の最大サイズはオペレーティングシステムによって異なります。次に例を示します。

- 2.5 GB - Windows 32 ビット Advanced Server、Enterprise Server、Datacenter Server
- 3.5 GB - Windows x64 Edition 上の 32 ビットデータベースサーバ
- 1.5 GB - その他すべての 32 ビットシステム
- 64 ビットのデータベースサーバの場合、キャッシュサイズは無制限と見なされる

デフォルトは **70p** です。

参照

- 「-cinit mlsrv16 オプション」53 ページ
- 「-cmin mlsrv16 オプション」54 ページ
- 「Mobile Link サーバでのメモリの使用」40 ページ

-cmin mlsrv16 オプション

サーバのメモリキャッシュの最小サイズを設定します。

構文

mlsrv16 -c "*connection-string***" -cmin** *size*[**k** | **m** | **g** | **p**] ...

備考

テーブルデータ、ネットワークバッファ、キャッシュされたダウンロードデータ、同期に使用されるその他の構造体を保持するためにサーバが使用するメモリの最小容量を指定します。このメモリプールに保持できる量を超えるデータがサーバにある場合、データはディスクにスワップされます。データがディスクにスワップされると、パフォーマンスが大幅に低下します。

size には、予約するメモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。数字の後に文字を指定しない場合、サイズはバイト単位になります。

単位 **p** は、物理システムメモリとプロセスアドレス可能領域のうち、いずれか小さい方のパーセンテージを表します。プロセスアドレス可能領域の最大サイズはオペレーティングシステムによって異なります。次に例を示します。

- 2.5 GB - Windows 32 ビット Advanced Server、Enterprise Server、Datacenter Server
- 3.5 GB - Windows x64 Edition 上の 32 ビットデータベースサーバ
- 1.5 GB - その他すべての 32 ビットシステム
- 64 ビットのデータベースサーバの場合、キャッシュサイズは無制限と見なされる

デフォルトは **50m** です。

参照

- 「-cmax mlsrv16 オプション」54 ページ
- 「-cinit mlsrv16 オプション」53 ページ
- 「Mobile Link サーバでのメモリの使用」40 ページ

-cn mlsrv16 オプション

データベースワーカスレッドの、統合データベースとの同時接続最大数を設定します。

構文

mlsrv16 -c "*connection-string***" -cn** *value* ...

備考

データベースワーカスレッドについて Mobile Link サーバから統合データベースへの最大同時接続数を指定します。最小値 (デフォルト) は、データベースワーカスレッド数です。指定した値がデータベースワーカスレッドの数よりも小さいと、警告が表示され、値は自動的に修正されます。

このタイプの Mobile Link データベース接続は、1 つのスクリプトバージョンを使用する同期にのみ使用されます。Mobile Link サーバが -cn オプションで許可されているすべてのデータベース接続を使用している場合、たとえ同期が保留中でも、そのスクリプトバージョンに対するデー

タベース接続が現在存在しないときには、Mobile Link サーバは接続を切断してから、保留中の同期のスクリプトバージョンに対して新しいデータベース接続を作成します。

データベースワーカスレッド数よりも大きい値を指定すると、統合データベースへの接続が低速の場合や、複数のスクリプトのバージョンが使用されている場合に、特にパフォーマンスが向上します。データベース接続の最適な最大数は、スクリプトのバージョン数とデータベースワーカスレッド数を乗算した値です。この最適値を上回る数の接続を使用しても、同期速度が上がるとはかぎりません。また、Mobile Link サーバと統合データベースサーバの両方で、必要以上にリソースが消費されます。

参照

- 「-w mlsrv16 オプション」89 ページ

-cr mlsrv16 オプション

データベース接続リトライの最大回数を設定します。

構文

mlsrv16 -c "*connection-string***" -cr** *value* ...

備考

接続不良のときに、Mobile Link サーバが終了前にデータベースへの接続を試行する最大回数を設定します。デフォルト値では、接続が 3 回リトライされます。

-cs mlsrv16 オプション

Mobile Link システムデータベース (MLSD) 用の接続パラメータを指定します。

構文

mlsrv16 -c "*connection-string***" -cs "***connection-string***"** ...

備考

システムテーブル、プロシージャ、トリガ、ビューなどの Mobile Link サーバシステムオブジェクトを、統合データベース以外のデータベースに格納できます。Mobile Link システム オブジェクトを格納するデータベースは、MLSD と呼ばれます。

コマンドラインでこのコマンドオプションが指定されると、Mobile Link サーバは、ユーザ定義スクリプトをフェッチしたり、ML ユーザ名、リモート ID、進行状況オフセット、最終アップロードタイムスタンプおよび最終ダウンロードタイムスタンプなどの同期ステータスを維持したりするために、MLSD への接続を作成します。Mobile Link サーバは、元の -c コマンドラインオプション接続を使用して統合データベースに接続し、クライアントデータベースとのデータのアップロードやダウンロードを行います。統合データベースには、Mobile Link サーバシステムオブジェクトは必要ありません。エラー報告スクリプトやエラー処理スクリプトを含むすべてのユーザ定義スクリプトは、MLSD からフェッチされ、統合データベースで実行されます。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

このオプションを使用する場合、Mobile Link サーバは、Microsoft 分散トランザクションコーディネーター (MSDTC) を必要とします。

統合データベースおよび MLSD には、サポートされている任意の Mobile Link 統合データベースを使用できます。ただし、対応する ODBC ドライバが Microsoft 分散トランザクションをサポートしている必要があります。

MSDTC を使用するには、統合データベースおよび MLSD にトランザクションログが必要です。

このオプションは、Windows オペレーティングシステムでしか使用できません。

-ct mlsrv16 オプション

接続が使用されなくなってから、タイムアウトになって Mobile Link サーバによって切断されるまでの時間を分単位で設定します。

構文

mlsrv16 -c "*connection-string***" -ct** *connection-timeout* ...

備考

指定された時間にわたって使用されなかった Mobile Link データベース接続は、サーバによって解放されます。-ct オプションを使用してタイムアウトを設定できます。デフォルトのタイムアウト期間は 60 分です。

-dl mlsrv16 オプション

Mobile Link サーバのメッセージウィンドウに画面上の Mobile Link サーバのすべてのメッセージを表示します。

構文

mlsrv16 -c "*connection-string***" -v -dl** ...

備考

Mobile Link サーバのメッセージウィンドウに Mobile Link サーバのすべてのメッセージを表示します。デフォルトでは、Mobile Link サーバのメッセージログファイルへの出力時には (-o を使用)、メッセージの一部だけがウィンドウに表示されます。多数のメッセージがある場合は、このオプションを指定するとパフォーマンスが低下することがあります。

参照

- 「-o mlsrv16 オプション」65 ページ
- 「ファイルへのデータベースサーバメッセージのロギング方法」『SQL Anywhere サーバ データベース管理』

-dr mlsrv16 オプション

Adaptive Server Enterprise の場合のみ。同期に関係するテーブルで、DataRow ロックスキームを使用しないようにします。

構文

mlsrv16 -c "*connection-string***" -dr** ...

備考

このオプションは、DataRow ロックスキームを使用して作成した同期中の統合テーブルがない場合にのみ使用してください。

このオプションを使用すると、Mobile Link サーバによって送信される重複データを減らすことができます。

参照

● 「Mobile Link 独立性レベル」 140 ページ

-ds mlsrv16 オプション

再起動可能なダウンロードで使用します。すべての再起動可能なダウンロードを保存するために Mobile Link サーバが使用できるディスク上のデータの最大量を指定します。

構文

mlsrv16 -c "*connection-string***" -ds** *size***[k | m | g]** ...

備考

Mobile Link サーバは、再起動可能なダウンロードで使用するために、クライアントに受信されなかったダウンロードデータを保持します。このオプションは、組み合わされたすべての同期に対してサーバが保持するデータの量を制限します。

size が小さ過ぎると、サーバがダウンロードデータを廃棄して、ダウンロードの再起動ができなくなる場合があります。サーバがダウンロードデータを破棄するのは、次のいずれかの場合です。

●ユーザがダウンロードを正常に完了した場合。

●再開を有効にしていない状況で、ユーザが新しい同期要求で復帰した場合。

●受信要求のキャッシュが必要な場合。正常にダウンロードできなかった最も古いデータが、最初にクリアされます。

単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、または **g** を使用してください。デフォルトは **10m** です。

Mobile Link サーバは、再起動可能なダウンロードのデータを保持している間、同期がアクティブであるとみなします (Mobile Link プロファイラの send_download フェーズ)。このため、ネットワークタイムアウトを超過しても同期は終了しません。

参照

- 「ダウンロードの失敗の再開」134 ページ
- 「-dc dbmlsync オプション」『Mobile Link クライアント管理』
- 「Resume Partial Download 同期パラメータ」『Ultra Light データベース管理とリファレンス』

-dsd mlsrv16 オプション

スナップショットアイソレーションを無効にします。

構文

mlsrv16 -c "*connection-string***" -dsd** ...

備考

統合データベースが SQL Anywhere (バージョン 10 以降) または Microsoft SQL Server (2005 以降) の場合、ダウンロードのデフォルトの独立性レベルはスナップショットアイソレーションです。統合データベースがこれらのデータベースのそれ以前のバージョンの場合には、デフォルトのダウンロード独立性レベルはコミットされた読み出しです。

スクリプトでデフォルトの独立性レベルを変更することもできます。しかし、SQL Anywhere バージョン 10 以降と Microsoft SQL Server 2005 以降のデータベースでは、独立性レベルはアップロードおよびダウンロードトランザクションの開始時に設定されます。begin_connection スクリプトで独立性レベルを設定しても、begin_upload スクリプトと begin_download スクリプトで上書きされる場合があります。

このオプションは、SQL Anywhere バージョン 10 と Microsoft SQL Server 2005 の統合データベースのみに適用されます。

参照

- 「Mobile Link 独立性レベル」140 ページ
- 「-dt mlsrv16 オプション」59 ページ
- 「-esu mlsrv16 オプション」61 ページ

-dt mlsrv16 オプション

Microsoft SQL Server データベースおよび Adaptive Server Enterprise データベースの場合のみ。Mobile Link が現在のデータベース内のみでトランザクションを検出するようにします。

構文

mlsrv16 -c "*connection-string***" -dt** ...

備考

このオプションにより、Mobile Link は、現在のデータベース内のトランザクションを除く、すべてのトランザクションを無視します。これにより、スループットが向上し、ダウンロードされるローの重複が減少します。

このオプションは、タイムスタンプベースのダウンロードにのみ影響を及ぼします。

次の場合に、このオプションを使用します。

- 統合データベースが、他のデータベースも実行している Microsoft SQL Server または Adaptive Server Enterprise の場合。
- Microsoft SQL Server との組み合わせでのアップロードまたはダウンロードにスナップショットアイソレーションを使用している場合。
- テーブルを Adaptive Server Enterprise と同期するために DataRow ロックスキームを使用している場合。
- アップロードスクリプトまたはダウンロードスクリプトが、サーバ上の他のデータベースにアクセスしない場合。

このオプションは、スナップショットアイソレーションを使用している Microsoft SQL Server データベースと、同期に関係するテーブルに DataRow ロックスキームを使用している Adaptive Server Enterprise データベースのみに適用されます。

参照

- 「Mobile Link 独立性レベル」140 ページ
- 「-dsd mlsrv16 オプション」59 ページ
- 「-esu mlsrv16 オプション」61 ページ

-e mlsrv16 オプション

SQL Anywhere Mobile Link クライアントから送信されたエラーログを格納します。

構文

mlsrv16 -c "*connection-string***" -e** *filename* ...

備考

-e オプションを指定しないと、SQL Anywhere Mobile Link クライアントからのエラーログは、ファイル *mlsrv16.mle* に格納されます。-e オプションは、Mobile Link サーバに対してエラーログを指定のファイルに格納するように指示します。デフォルトでは、リモートサイトでエラーが発生すると、dbmlsync は Mobile Link サーバにリモートログメッセージを 32 キロバイトまで送信します。

このオプションは、同期の問題を診断するために、リモートエラーログへの一元化されたアクセスを可能にします。

リモートサイトから配信される情報量は、dbmlsync の拡張オプション ErrorLogSendLimit で制御できます。

参照

- 「-et mlsrv16 オプション」61 ページ
- 「ErrorLogSendLimit (el) 拡張オプション」『Mobile Link クライアント管理』

-esu mlsrv16 オプション

アップロードにスナップショットアイソレーションを使用します。

構文

mlsrv16 -c "*connection-string***" -esu** ...

備考

デフォルトでは、Mobile Link はアップロードに対して READ COMMITTED 独立性レベルを使用します。これは、通常、最適な独立性レベルです。

アップロードにスナップショットアイソレーションを使用すると、アップロードの更新中にスナップショットトランザクションで競合が発生する場合があります。競合が発生した場合、Mobile Link サーバはアップロード全体をロールバックして、アップロードのリトライを行います。この場合、Mobile Link サーバオプション -r または -rd の設定を調整して、リトライ間隔の遅延時間と最大リトライ回数を指定できます。

スクリプトでデフォルトの独立性レベルを変更することもできます。アップロードの独立性レベルを変更するには、通常、begin_upload スクリプトを使用します。

このオプションは、SQL Anywhere バージョン 10 以降と Microsoft SQL Server 2005 以降の統合データベースのみに適用されます。

参照

- 「Mobile Link 独立性レベル」 140 ページ
- 「-dsd mlsrv16 オプション」 59 ページ
- 「-dt mlsrv16 オプション」 59 ページ
- 「-r mlsrv16 オプション」 73 ページ
- 「-rd mlsrv16 オプション」 74 ページ

-et mlsrv16 オプション

既存のファイルをトランケートした後で、SQL Anywhere Mobile Link クライアントから送信されたエラーログを指定のファイルに格納します。

構文

mlsrv16 -c "*connection-string***" -et** *filename* ...

備考

-et オプションは -e オプションと同じですが、エラーログファイルがトランケートされてから、そのファイルに新しいエラーが追加されます。

リモートサイトから配信される情報量は、dbmlsync の拡張オプション ErrorLogSendLimit で制御できます。

参照

- 「ErrorLogSendLimit (el) 拡張オプション」『Mobile Link クライアント管理』
- 「-e mlsrv16 オプション」 60 ページ

-fips mlsrv16 オプション

すべての Mobile Link セキュアストリームを強制的に FIPS 認定モジュールとします。

構文

mlsrv16 -c *connection-string*" **-fips** ...

備考

このオプションを指定すると、Mobile Link サーバのすべての暗号で FIPS 認定のアルゴリズムが使用されます。-fips オプションが指定されているときには、暗号化されていない接続は使用できますが、単純暗号化は使用できません。

このオプションを指定すると、FIPS 認定のアルゴリズムが指定されているかどうかに関係なく、FIPS 認定のアルゴリズムが接続に使用されます。たとえば、オプション -fips とオプション -x tls(...;fips=no;...) を使用して Mobile Link サーバを起動した場合、fips=no 設定は無視され、サーバは fips=yes として起動されます。

> **注意**
> 別途ライセンスが必要な必須コンポーネント
>
> FIPS 認定の暗号化には別のライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。
>
> 「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

Mobile Link トランスポートレイヤセキュリティの場合、-fips オプションを使用しない RSA が指定されている場合でも、-fips オプションにより、サーバは FIPS 認定の RSA 暗号化アルゴリズムを使用するようになります。

参照

- 「Mobile Link サーバ／クライアント通信の暗号化」『SQL Anywhere サーバ データベース管理』
- 「FIPS 認定の暗号化テクノロジ」『SQL Anywhere サーバ データベース管理』

-ftr mlsrv16 オプション

mlfiletransfer ユーティリティまたは Mobile Link エージェントによってダウンロードされるファイル用のロケーションを指定します。

構文

mlsrv16 -c "*connection-string***" -ftr** *path* ...

備考

このオプションはファイル転送ルートディレクトリを設定します。ユーザに転送するファイルは、ルートディレクトリに配置することも、ユーザ名を使用したサブフォルダに配置することもできます。Mobile Link はまず、要求されたファイルを、接続したクライアントのユーザ名を使用した、ファイル転送ルートディレクトリのサブフォルダで検索します。ファイルがサブフォルダになかった場合、Mobile Link はファイル転送ルートディレクトリで検索します。

このオプションは、mlfiletransfer ユーティリティを使用してファイルをダウンロードするために必要です。

参照

- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」『Mobile Link クライアント管理』
- 「-ftru mlsrv16 オプション」63 ページ
- 「authenticate_file_transfer 接続イベント」282 ページ

-ftru mlsrv16 オプション

mlfiletransfer ユーティリティまたは Mobile Link エージェントによってアップロードされるファイル用のロケーションを指定します。

構文

mlsrv16 -c "*connection-string***" -ftru** *path* …

備考

このオプションは mlfiletransfer ユーティリティによってアップロードされるファイルのファイル転送ルートディレクトリを設定します。ファイルは、このルートディレクトリか、このルートディレクトリ直下のサブフォルダにのみアップロードできます。

authenticate_file_upload スクリプトが存在しないか、このスクリプトが存在し、かつ 1000 ～ 1999 の範囲の authentication_code を返す場合にのみ、ファイルをアップロードできます。この要件は、mlfiletransfer にのみ該当し、Mobile Link エージェントには適用されません。

このオプションは、mlfiletransfer ユーティリティを使用してファイルをアップロードするために必要です。

参照

- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」『Mobile Link クライアント管理』
- 「-ftr mlsrv16 オプション」62 ページ
- 「authenticate_file_transfer 接続イベント」282 ページ

-lsc mlsrv16 オプション

ローカルサーバの接続情報を指定します。この情報は、サーバファームのその他のサーバに渡されます。

構文

mlsrv16 -c "*connection-string***" -lsc** *protocol*[*protocol-options*] ...

protocol : **tcpip** | **tls** | **http** | **https**

protocol-options : **(** *option*=*value*; ... **)**

備考

このオプションは、次の場合にのみ使用します。

●Mobile Link サーバファームで Notifier を実行する場合。

●-rrp サーバオプションを指定して mlreplay ユーティリティを使用する場合。

たとえば、server_rack10 というホストでサーバが実行されている場合、コマンドラインは次のように始めることができます。

```
mlsrv16  -x tcpip(port=200) -zs server5 -lsc tcpip(host=server_rack10;port=200) -c...
```

この例では、他のサーバは、統合データベースで共有ステータスを使用して接続文字列 tcpip(host=server_rack10;port=200) を取得し、その文字列を使用して、起動されたサーバに接続できます。

参照

- 「-rrp mlsrv16 オプション」74 ページ
- 「サーバファーム内の Mobile Link サーバ」39 ページ
- 「-zs mlsrv16 オプション」104 ページ
- 「-rrp mlsrv16 オプション」74 ページ
- 「Mobile Link サーバファームでの Notifier」『Mobile Link サーバ起動同期』

-nc mlsrv16 オプション

同時ネットワーク接続の最大数を設定します。

構文

mlsrv16 -c "*connection-string***" -nc** *connections* ...

備考

制限値に達すると、Mobile Link サーバは新しい同期接続を拒否します。クライアントでは、接続が拒否されたことを示すエラーコードで通信エラーが発行されます。

デフォルトは 1024 です。

非永続 HTTP/HTTPS の同時同期の数を制限するには、-sm よりも大幅に大きい値を -nc に設定してください。-sm の制限に達すると、Mobile Link サーバは HTTP エラー 503 (Service Unavailable) をリモートクライアントに返します。ただし -nc の制限に達すると、ソケットエラーが発行されます。-nc と -sm の差が大きくなると、拒否された接続によって生成されるエラーが、わかりにくいソケットエラーではなく HTTP 503 エラーになる可能性が高くなります。たとえば -sm を 100 に設定し、-nc を 1000 に設定してください。

-nc の最大値はオペレーティングシステムとその設定によって異なります。場合によっては、ソケットの処理能力を高めるために、設定を調整することが必要となります。

参照

- 「-sm mlsrv16 オプション」79 ページ

-notifier mlsrv16 オプション

サーバによって開始される同期用に Notifier を起動します。

構文

mlsrv16 -c "*connection-string***" -notifier** [*notifier-properties-file*] ...

備考

Notifier の設定ファイル名を指定した場合、またはファイル名を指定していないが、デフォルトの Notifier プロパティファイル (*config.notifier*) がある場合は、そのファイルを使用して Notifier が設定されます。このファイルは、統合データベースの ml_properties テーブルに格納されている設定情報よりも優先されます。

それ以外の場合は、統合データベースの ml_properties テーブルに格納されている設定情報が使用されます。

-notifier オプションを使用する場合、有効にしたすべての Notifier が起動されます。

Notifier の有効化の詳細については、「Notifier プロパティ」『Mobile Link サーバ起動同期』を参照してください。

参照

- 「サーバ起動同期の Mobile Link サーバ設定」『Mobile Link サーバ起動同期』
- 「Notifier 設定ファイルを使用したサーバ側の設定」『Mobile Link サーバ起動同期』
- 「Notifier」『Mobile Link サーバ起動同期』
- 「Mobile Link サーバファームでの Notifier」『Mobile Link サーバ起動同期』

-o mlsrv16 オプション

Mobile Link サーバのメッセージログファイルに出力メッセージのログを取り、Mobile Link サーバのメッセージウィンドウに記録されるデータを制限します。

構文

mlsrv16 -c "*connection-string***" -o** *logfile* ...

備考

指定したファイルにすべてのログメッセージを書き込みます。Mobile Link サーバのメッセージウィンドウには (表示されている場合)、通常はログが取られたすべてのメッセージのうち、一部のみが表示されます。

同期中にエラーが発生した場合、Mobile Link サーバは、この出力ファイルに完全なエラーコンテキストを示します。エラーコンテキストには次のような情報が含まれます。

- **リモート ID**　同期しているリモートデータベースのリモート ID です。
- **ユーザ名**　同期中に Mobile Link クライアントに提供された実際のユーザ名。
- **変更後のユーザ名**　modify_user スクリプトで変更されたユーザ名。
- **トランザクション**　エラーが発生したトランザクションのリスト。トランザクションには、authenticate_user、begin_synchronization、upload、prepare_for_download、download、または end_synchronization があります。
- **テーブル名**　テーブル名がある場合はテーブル名。テーブル名がない場合は NULL。
- **ローの操作**　操作には、INSERT、UPDATE、DELETE、または FETCH がある。
- **ローデータ**　エラーが発生したローのすべてのカラム値。
- **スクリプトバージョン**　現在同期に使用されているスクリプトバージョン。
- **スクリプト**　エラーの原因となったスクリプト。

選択した冗長性のレベルに関係なく、ログにエラーコンテキスト情報が表示されます。

参照

- 「-os mlsrv16 オプション」67 ページ
- 「-dl mlsrv16 オプション」57 ページ
- 「-ot mlsrv16 オプション」68 ページ
- 「-on mlsrv16 オプション」66 ページ
- 「-v mlsrv16 オプション」85 ページ

-on mlsrv16 オプション

Mobile Link サーバのメッセージログファイルの最大サイズを指定します。ログファイルがこのサイズに達すると、現在のファイルが拡張子 .old の付いた名前に変更され、新しいファイルが作成されます。

構文

mlsrv16 -c "*connection-string***" -on** *size* **[k | m]**...

備考

size には、メッセージログの最大ファイルサイズを、バイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小サイズは 10 KB です。

ログファイルが指定されたサイズに達すると、Mobile Link サーバは出力ファイルを拡張子 .old の付いた名前に変更し、元の名前で新しいファイルを開始します。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

> **注意**
> .old ファイルがすでに存在する場合は、そのファイルが上書きされます。最大 2 つのファイルが使用されます。古いログファイルを削除しないようにするには、-os オプションを使用します。

このオプションは、-os オプションと同時に使用できません。

参照

- 「-o mlsrv16 オプション」65 ページ
- 「-ot mlsrv16 オプション」68 ページ
- 「-on mlsrv16 オプション」66 ページ
- 「-os mlsrv16 オプション」67 ページ
- 「-v mlsrv16 オプション」85 ページ

-oq mlsrv16 オプション

Windows で、起動エラーの発生時にエラーウィンドウが表示されないようにします。

構文

mlsrv16 -c "*connection-string***" -oq** ...

備考

デフォルトでは、起動エラーが発生すると、Mobile Link サーバにウィンドウが表示されます。-oq オプションを指定すると、このウィンドウは表示されません。

-os mlsrv16 オプション

Mobile Link サーバの現在のメッセージログファイルと古いメッセージログファイルの最大サイズを設定します。

構文

mlsrv16 -c "*connection-string***" -os** *size* [**k** | **m**] ...

備考

size には、出力メッセージのログを取るファイルの最大サイズを指定します。デフォルトの単位はバイトです。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小サイズは 10 KB です。

Mobile Link サーバは現在のファイルサイズを確認してから、ファイルに出力メッセージのログを取ります。ログメッセージによって、ファイルサイズが指定したサイズより大きくなると、Mobile Link サーバはメッセージログファイルの名前を *yymmddxx.mls* に変更します。*xx* は 00 ～ 99 の数字で、*yymmdd* は現在の年月日を表します。

常に最新の出力が -o または -ot で指定したファイルに追加されます。

このオプションは、-on オプションと一緒に使用することはできません。

注意
このオプションを使用すると、ログファイルが無制限に作成されます。この状況を回避するには、-o または -on を使用します。

参照

- 「-o mlsrv16 オプション」65 ページ
- 「-on mlsrv16 オプション」66 ページ
- 「-ot mlsrv16 オプション」68 ページ
- 「-v mlsrv16 オプション」85 ページ

-ot mlsrv16 オプション

最初に Mobile Link サーバのメッセージログファイルの内容を削除してから、そのファイルに出力メッセージのログを取ります。

構文

mlsrv16 -c "*connection-string***" -ot** *logfilename* ...

備考

デフォルトでは、Mobile Link サーバのメッセージウィンドウまたは画面に出力を送信します。

参照

- 「-on mlsrv16 オプション」66 ページ
- 「-os mlsrv16 オプション」67 ページ
- 「-v mlsrv16 オプション」85 ページ
- 「-o mlsrv16 オプション」65 ページ

-ppv mlsrv16 オプション

指定された期間に従って、Mobile Link が新しい定期モニタリングの値を出力するようにします。期間は秒数で指定します。

構文

mlsrv16 -c "*connection-string***" -ppv** *period* ...

備考

これらの値は、サーバの状態を把握するために使用でき、Mobile Link サーバの正常性やパフォーマンスを判断するときに便利です。たとえば、DB_CONNECTIONS と LONGEST_DB_WAIT の値を確認して、-w オプションや同期スクリプトに存在する潜在的な問題を調べることができます。これらの値を使用して、1 秒間にアップロードまたはダウンロードされたロー数や、1 秒間に成功した同期数など、システム全体のスループット測定を簡単に追跡することもできます。

期間の推奨値は 60 秒です。

設定した期間の値が小さすぎると、ログが急速に増大します。

出力の各ローは、値の検索やフィルタが容易になるように、**PERIODIC:** で始まります。

出力される値には、次の情報が含まれます。

出力される値	説明
CMD_PROCESSOR_STAGE_LEN	同期タスクのキューの長さ。
CPU_USAGE	Mobile Link サーバによって使用される CPU 時間の量 (マイクロ秒)。
DB_CONNECTIONS	使用中のデータベース接続数。
FREE_DISK_SPACE	テンポラリディスクのディスク空き容量 (バイト数)。
HEARTBEAT_STAGE_LEN	同期以外の定期的なタスクのキューの長さ。
LONGEST_DB_WAIT	アクティブな同期でデータベースを待機していた最長時間。
LONGEST_SYNC	最も古いアクティブな同期の経過時間 (マイクロ秒)。
MEMORY_USED	使用中の RAM のバイト数 (Windows のみ)。
ML_NUM_CONNECTED_CLIENTS	接続された同期クライアント数。
NOTIFIER_STAGE_LEN	Notifier ワークキューの長さ。
NUM_COMMITS	コミットの合計数。
NUM_CONNECTED_FILE_XFERS	現在接続されている mlfiletransfer 数。
NUM_CONNECTED_LISTENERS	現在接続されている Listener 数。
NUM_CONNECTED_MONITORS	現在接続されているモニタ数。
NUM_CONNECTED_PINGS	現在接続されている、ping を実行するクライアント数。
NUM_CONNECTED_SYNCS	現在接続されているデータ同期数。
NUM_ERRORS	エラーの合計数。
NUM_FAILED_SYNCS	失敗した同期の合計数。
NUM_IN_APPLY_UPLOAD	現在、アップロードの適用フェーズにある同期の数。

出力される値	説明
NUM_IN_AUTH_USER	現在、ユーザ認証フェーズにある同期の数。
NUM_IN_BEGIN_SYNC	現在、同期開始フェーズにある同期の数。
NUM_IN_CONNECT	現在、接続フェーズにある同期の数。
NUM_IN_CONNECT_FOR_ACK	現在、ダウンロード確認の接続フェーズにある同期の数。
NUM_IN_END_SYNC	現在、同期終了フェーズにある同期の数。
NUM_IN_FETCH_DNLD	現在、ダウンロードのフェッチフェーズにある同期の数。
NUM_IN_GET_DB_WORKER_FOR_ACK	非ブロッキングダウンロード確認を処理するためにデータベース接続を現在待機している同期の数。
NUM_IN_NON_BLOCKING_ACK	現在、非ブロッキングダウンロード確認フェーズにある同期の数。
NUM_IN_PREP_FOR_DNLD	現在、ダウンロードの準備フェーズにある同期の数。
NUM_IN_RECVING_UPLOAD	現在、アップロードの受信フェーズにある同期の数。
NUM_IN_SEND_DNLD	現在、ダウンロードの送信フェーズにある同期の数。
NUM_IN_SYNC_REQUEST	現在、同期要求フェーズにある同期の数。
NUM_IN_WAIT_FOR_DNLD_ACK	現在、ダウンロード確認の待機フェーズにある同期の数。
NUM_ROLLBACKS	ロールバックの合計数。
NUM_ROWS_DOWNLOADED	リモートに送信されたローの合計数。
NUM_ROWS_UPLOADED	リモートから受信したローの合計数。
NUM_SUCCESS_SYNCS	成功した同期の合計数。
NUM_UPLOAD_CONNS_IN_USE	現在使用中のアップロード接続の数。
NUM_WAITING_CONS	統合データベースを現在待機している同期の数。

出力される値	説明
NUM_WARNINGS	警告の合計数。
OE_STAGE_LEN	統合化 Outbound Enabler ワークキューの長さ。
PAGES_IN_STREAMSTACK	ネットワークストリームで保持されているページの数。
PAGES_LOCKED	メモリにロードされているキャッシュページ数。
PAGES_LOCKED_MAX	メモリキャッシュ内のページの数。
PAGES_SWAPPED_IN	ディスクから読み込まれたページの合計数。
PAGES_SWAPPED_OUT	ディスクへスワップされたページの合計数。
PAGES_USED	使用されたキャッシュページ数。ディスクにスワップされたページが含まれるため、キャッシュサイズよりも大きくなることがあります。
PRIMARY_IS_KNOWN	プライマリサーバが認識されているかどうかを示します。プライマリサーバがどれであってもかまわない場合は、0 を示します。プライマリサーバがどれであるか認識されている場合は、1 を示します。プライマリサーバがどれであるか認識されていない場合は、2 を示します。
RAW_TCP_STAGE_LEN	ネットワークワークキューの長さ。
SERVER_IS_PRIMARY	サーバがプライマリかセカンダリかを示します。サーバがプライマリの場合は 1、それ以外の場合は 0 を示します。
SIRT_NUM_LWP_HITS	通知を示す、リモートタスクエージェントのライトウェイトポーリング数。
SIRT_NUM_LWPS	リモートタスクエージェントのライトウェイトポーリング数。
SIRT_NUM_REQUESTS	現在未処理のリモートタスク通知の数。
STREAM_STAGE_LEN	高レベルのネットワーク処理キューの長さ。
TCP_BYTES_READ	これまでに読み込まれたバイトの合計数。

出力される値	説明
TCP_BYTES_WRITTEN	これまでに書き込まれたバイトの合計数。
TCP_CONNECTIONS	現在開いている TCP 接続の数。
TCP_CONNECTIONS_CLOSED	これまでに閉じられた接続の合計数。
TCP_CONNECTIONS_OPENED	これまでに開かれた接続の合計数。
TCP_CONNECTIONS_REJECTED	これまでに拒否された接続の合計数。
TIMED_WORK_STAGE_LEN	動的キャッシュワークキューの長さ。
TRACKED_MEMORY	サーバによって割り付けられたメモリの量。このメトリックは、MEMORY_USED メトリックを使用できない Windows 以外のシステムに対して使用します。Microsoft Windows システムでは、精度を上げるために MEMORY_USED メトリックを使用してください。
VM_MEM_USE	付加された VM で使用されるメモリの量。

例

次は、定期モニタリングの値を示す出力の例です。

```
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_USED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_LOCKED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_LOCKED_MAX: 13243
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_OPENED: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_CLOSED: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_REJECTED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_BYTES_READ: 5137
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_BYTES_WRITTEN: 4549
I. 2009-10-28 11:46:29. <Main> PERIODIC: ML_NUM_CONNECTED_CLIENTS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_SWAPPED_OUT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_SWAPPED_IN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_IN_STREAMSTACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: CPU_USAGE: 3359375
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_COMMITS: 7
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROLLBACKS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_SUCCESS_SYNCS: 1
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_FAILED_SYNCS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ERRORS: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_WARNINGS: 3
I. 2009-10-28 11:46:29. <Main> PERIODIC: DB_CONNECTIONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: RAW_TCP_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: STREAM_STAGE_LEN: 5
I. 2009-10-28 11:46:29. <Main> PERIODIC: HEARTBEAT_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: CMD_PROCESSOR_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROWS_DOWNLOADED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROWS_UPLOADED: 7
I. 2009-10-28 11:46:29. <Main> PERIODIC: FREE_DISK_SPACE: 124154904576
```

```
I. 2009-10-28 11:46:29. <Main> PERIODIC: LONGEST_DB_WAIT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: LONGEST_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: MEMORY_USED: 140275712
I. 2009-10-28 11:46:29. <Main> PERIODIC: SERVER_IS_PRIMARY: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_SYNCS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_PINGS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_FILE_XFERS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_MONITORS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_LISTENERS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_WAITING_CONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_SYNC_REQUEST: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_RECVING_UPLOAD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_CONNECT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_AUTH_USER: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_BEGIN_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_APPLY_UPLOAD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_PREP_FOR_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_FETCH_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_END_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_SEND_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_WAIT_FOR_DNLD_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_GET_DB_WORKER_FOR_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_CONNECT_FOR_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_NON_BLOCKING_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_UPLOAD_CONNS_IN_USE: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: TRACKED_MEMORY: 56269577
I. 2009-10-28 11:46:29. <Main> PERIODIC: VM_MEM_USE: 517013504
I. 2009-10-28 11:46:29. <Main> PERIODIC: TIMED_WORK_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NOTIFIER_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_REQUESTS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_LWPS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_LWP_HITS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: OE_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PRIMARY_IS_KNOWN: 0
```

-q mlsrv16 オプション

Mobile Link を起動時に最小化メッセージウィンドウで実行するように指示します。

構文

mlsrv16 -c "*connection-string***" -q** …

備考

Mobile Link サーバのメッセージウィンドウを最小化します。

-r mlsrv16 オプション

デッドロックの最大リトライ回数を設定します。

構文

mlsrv16 -c "*connection-string***" -r** *retries* …

備考

デフォルトでは、Mobile Link サーバは統合データベースでデッドロックされたアップロードを最大 10 回リトライします。デッドロックは解消できる保証がないため、デッドロックが解除されない場合は同期が失敗します。このオプションで、リトライの制限回数を任意に設定できます。サーバによるデッドロックトランザクションのリトライを停止するには、**-r 0** を指定します。この設定の上限は、2 の 32 乗から 1 を引いた値です。

> **注意**
> 正常な同期システムでは、デッドロックは発生しません。発生した場合は、デッドロックが発生しないように同期スクリプトを修正してください。

-rd mlsrv16 オプション

デッドロックリトライ間の最大遅延時間を設定します。

構文

mlsrv16 -c "*connection-string***" -rd** *delay* ...

備考

統合データベースでアップロードトランザクションがデッドロックされると、Mobile Link サーバは不特定の時間待機してから、そのトランザクションをリトライします。遅延時間がランダムなため、その後の試行が成功する可能性が高くなります。このオプションを使用すると、最大遅延時間を秒単位で指定できます。値を 0 にするとリトライが即座に行われますが、より大きな値を指定した方がリトライの成功率が高くなります。デフォルトの最大遅延値は **30** です。

> **注意**
> 正常な同期システムでは、デッドロックは発生しません。発生した場合は、デッドロックが発生しないように同期スクリプトを修正してください。

-rrp mlsrv16 オプション

Mobile Link サーバが起動時に、指定されたディレクトリで mlreplay ユーティリティを実行し、記録されたすべてのセッション (拡張子が *mlr* のファイル) をリプレイするようにします。

このオプションを使用すると、リモートスキーマを Mobile Link サーバにプリロードできます。これにより、リモートスキーマを送信するフィールドにおけるリモートの最初の同期の時間と労力を節約できます。

構文

mlsrv16 -c "*connection-string***" -rrp** *directory* ...

備考

-rrp オプションを使用するには、mlreplay ユーティリティがサーバに接続できるように -lsc オプションを使用してローカルサーバの接続文字列を指定する必要があります。

-rrp オプションと -rp オプションを使用するには

●-rp オプションを使用して同期を記録します。

●スキーマのプリロードに使用する記録済み同期を特定します。パブリケーションのスキーマまたはセットごとに 1 つ必要です。

●記録済み同期を新しいディレクトリにコピーします。

●運用環境で、-rp オプションを指定せず、-rrp オプションを指定して実行します。

参照

- 「-rp mlsrv16 オプション」75 ページ
- 「-lsc mlsrv16 オプション」63 ページ
- 「Mobile Link Replay ユーティリティ (mlreplay)」696 ページ

-rp mlsrv16 オプション

mlreplay ユーティリティでプレイバックする同期の記録先ディレクトリを指定します。

構文

mlsrv16 -c "*connection-string***" -rp** *directory* ...

備考

最高のパフォーマンスを得るには、このオプションを使用して、-rrp オプションで使用される同期を記録します。-rrp オプションを使用すると、ユニークな各スキーマの最初の同期を含むすべての同期でスキーマキャッシュを利用できます。

●-rp オプションを使用して同期を記録します。

●スキーマのプリロードに使用する記録済み同期を特定します。パブリケーションのスキーマまたはセットごとに 1 つ必要です。

●記録済み同期を新しいディレクトリにコピーします。

●運用環境で、-rp オプションを指定せず、-rrp オプションを指定して実行します。

参照

- 「-rrp mlsrv16 オプション」74 ページ
- 「Mobile Link Replay ユーティリティ (mlreplay)」696 ページ

-s mlsrv16 オプション

一度にアップロードできるローの最大数を設定します。

構文

mlsrv16 -c "*connection-string***" -s** *count* ...

備考

一度に挿入、更新、または削除できるローの最大数を *count* に設定します。

Mobile Link サーバは、ODBC ドライバを使用してアップロードローを統合データベースに送信します。このオプションは、各バッチでデータベースサーバに送信されるローの数を制御します。この値を増やすと、アップロードストリームの処理速度を向上させ、ネットワーク時間を短縮できます。しかし、設定を高くすると、Mobile Link サーバでは、アップロードストリームに適用するリソースをより多く必要とする可能性があります。

一度にアップロードされるローの数は、ログファイルで **rowset size** として参照できます。

デフォルトは 10 です。

-sl dnet mlsrv16 オプション

.NET Common Language Runtime (CLR) オプションを設定し、起動時に CLR を強制的にロードします。.NET スクリプトロジックを使用する場合にこのオプションを使用することをおすすめします。

構文

mlsrv16 -c "*connection-string***" -sl dnet** *options* ...

備考

.NET CLR に直接渡すオプションを設定します。

使用可能なオプションは以下のとおりです。

オプション	説明
-D*name=value*	環境変数を設定します。次に例を示します。 -Dsynchtype=far -Dextra_rows=yes 詳細については、.NET フレームワークのクラス System.Environment を参照してください。
-MLAutoLoadPath= *path*	基本アセンブリのロケーションを設定します。プライベートアセンブリでのみ有効です。アセンブリのロケーションを Mobile Link に指示するには、このオプションまたは -MLDomConfigFile を使用します。両方同時には使用できません。-MLAutoLoadPath を使用する場合、イベントスクリプトでドメインを指定できません。デフォルトは現在のディレクトリです。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

オプション	説明
-MLDomConfigFile= *file*	基本アセンブリのロケーションを設定します。共有アセンブリがある場合、すべてのアセンブリをディレクトリにロードしない場合、またはその他の理由で MLAutoLoadPath を使用できない場合に使用します。アセンブリのロケーションを Mobile Link に指示するには、-MLDomConfigFile または -MLAutoLoadPath を使用します。両方同時には使用できません。 -MLDomConfigFile オプションで参照されるファイルパスが、名前にスペースが含まれるフォルダ内にあるファイル (*"C:¥Program Files¥MyCompany¥SyncServer ¥MlDomConfig.xml"*) を参照する場合は、*"-MlDomConfigFile=C:¥Program Files¥MyCompany ¥SyncServer¥MlDomConfig.xml"* のように、オプション全体を二重引用符で囲みます。
-MLStartClasses= *classnames*	サーバの起動時に、ユーザ定義の起動クラスをリスト内の順序でロードしインスタンス化します。
-clrConGC	CLR での同時ガーベジコレクションを有効にします。
-clrFlavor=(wks | svr)	ロードする .NET CLR の種類。サーバは **svr** で、ワークステーションは **wks** です。デフォルトでは、**svr** がロードされます。
-clrVersion= *version*	ロードする .NET CLR のバージョン。プレフィクス **v** が必要です。たとえば、**v1.0.3705** と指定すると、ディレクトリ *¥Microsoft.NET¥Framework¥v1.0.3705* がロードされます。

v4.0 のアセンブリを使用するには、Mobile Link サーバに v4.0 のランタイムがロードされるように、-clrVersion オプションを明示的にインクルードする必要があります。たとえば、-clrVersion=v4.0.30319 のように記述します。

このオプションリストを表示するには、次のコマンドを実行します。

```
mlsrv16 -sl dnet (?)
```

参照

- 「.NET の同期スクリプト」533 ページ

-sl java mlsrv16 オプション

Java VM のオプションを設定し、起動時に Java VM を強制的にロードします。Java スクリプトロジックを使用する場合にこのオプションを使用することをおすすめします。UNIX では、-cp オプションをコロンで区切る必要があります。

構文

mlsrv16 -c "*connection-string***" -sl java (** *options* **) ...**

備考

Java VM に直接渡す -jrepath とその他のオプションを設定します。オプションは次のとおりです。

オプション	説明
-classic	classic Java VM を使用します。
-client	client Java VM を使用します。
-hotspot	hotspot Java VM を使用します。
-server	server Java VM を使用します。これはデフォルトです。
-cp *location***;...**	クラスの検索先となる一連のディレクトリまたは JAR ファイルを指定します。また、-classpath も使用できます。
-D*name=value*	システムプロパティを設定します。次に例を示します。 -Dsynchtype=far -Dextra_rows=yes
-DMLStartClasses=*classname*, ...	サーバの起動時に、ユーザ定義の起動クラスをリスト内の順序でロードしインスタンス化します。
-jrepath *path*	デフォルトの JRE パスを上書きします。デフォルトの JRE パスは、32 ビットプラットフォームの場合は *%SQLANY16%¥Bin32¥jre170*、64 ビットプラットフォームの場合は *%SQLANY16%¥Bin64¥jre170* です。
-verbose [**:class** |**:gc** | **:jni**]	冗長出力を有効にします。
-X *vm-option*	ファイル *%SQLANY16%¥Bin32¥jre170¥bin¥client¥Xusage.txt* (32 ビット) またはファイル *%SQLANY16%¥Bin64¥jre170¥bin¥server¥Xusage.txt* (64 ビット) での記述に従って、VM 固有のオプションを設定します。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

使用できる Java オプションのリストを表示するには、次のように入力します。

```
java
```

UNIX に関する注意事項

オプションはカッコで囲んでください。これは、丸カッコでも、上記の構文に示すように中カッコ { } でもかまいません。

-jrepath オプションは、Windows でのみ使用できます。UNIX で特定の JRE をロードする場合は、LD_LIBRARY_PATH (IBM AIX の場合は LIBPATH、HP-UX の場合は SHLIB_PATH) を設定して、JRE を含むディレクトリを指定します。ディレクトリは、すべての SQL Anywhere インストールディレクトリの前に指定します。

参照

- 「Java による同期スクリプトの作成」459 ページ

例

たとえば、Windows では、次の mlsrv16 コマンドラインの一部では、システムアサートを有効にする Java VM オプションを設定します。

```
mlsrv16 -sl java (-cp ;¥myclasses; -esa) ...
```

Windows では、次の mlsrv16 コマンドラインの一部は、LDAP_SERVER システムプロパティを定義します。

```
mlsrv16 -sl java ( -cp ;¥myclasses; -DLDAP_SERVER=mycorp-ldap ) ...
```

UNIX では、次の mlsrv16 コマンドラインの一部を使用できます。

```
mlsrv16 -sl java { -cp .:$CLASSPATH:/opt/myclasses:/opt/my.jar: }
```

-sm mlsrv16 オプション

ネットワーク接続の最大数を制限することによって、アクティブに動作できる同期の最大数を設定します。

構文

mlsrv16 -c "*connection-string***" -sm** *number* ...

備考

Mobile Link サーバは次の同期タスクを同時に実行します。

1. ネットワークからアップロードデータを読み込んでアンパックする。
2. アップロードを統合データベースに適用する。
3. 統合データベースからダウンロードするローをフェッチする。
4. ダウンロードデータをパックして、リモートデータベースに送信する。

各タスクの同期の数は次のように制限されます。

●タスク 2 と 3 を実行する同期の数は、mlsrv16 -w オプションの設定以下

●タスク 2 を実行する同期の数は、mlsrv16 -wu オプションの設定以下

●4 つのタスクすべてを実行する同期の数は、-sm オプションの設定以下

-sm の値を高くすると、特に -w より高く設定した場合に、Mobile Link サーバはネットワークタスク (1 と 4) をデータベースタスク (2 と 3) より多く処理できます。このようにすると、そのままではネットワークパフォーマンスがボトルネックになるような状況で、データベースワーカがタスクを待つ必要がなくなります。これによりスループットが向上します。しかし、-sm の設定が過度に高く、十分な同時接続がある場合、Mobile Link サーバは、直接使用可能な量を超えるメモリを割り付けできるようになります。その結果、オペレーティングシステムの仮想メモリページングがアクティブになり、メモリはデスクにスワップされて、スループットが極端に低下します。

参照

- 「-w mlsrv16 オプション」 89 ページ
- 「-wu mlsrv16 オプション」 91 ページ
- 「-nc mlsrv16 オプション」 64 ページ

-wn mlsrv16 オプション

ネットワークストリームの同時処理のために Mobile Link サーバが使用するネットワークワーカスレッドの数を設定します。

構文

mlsrv16 -c "*connection-string***" -wn** *count* ...

備考

デフォルト値は 1 です。

複数のネットワークワーカスレッドが存在することで、特に暗号化や圧縮など、大規模な同期または小規模の多数の同期を伴う CPU 集約型ネットワークストリームオプションを使用しているときは、パフォーマンスが向上することがあります。システム内の各要求は、最大で 1 つのネットワークストリームスレッド上でアクティブにできます。

-tc mlsrv16 オプション

実行時間が長い SQL スクリプトのタイムアウトスレッショルドを設定します。

構文

mlsrv16 -c "*connection string***" -tc** *minutes* ...

備考

デフォルトでは、Mobile Link サーバは、各 SQL スクリプトの実行時間を監視し、スクリプトの実行時間が 10 分間に達すると警告メッセージを発行します。実行時間が長いスクリプトは、統合データベースで競合とブロッキングを引き起こす可能性があります。その場合、全体的なスループットが大幅に低下することがあります。

-tf オプションを使用すると、スレッショルドを超える文を取り消すことができます。

デフォルト値は 0 にリセットすることも、正の整数にリセットすることもでき、単位は分です。値が 0 に設定されると、-tc オプションが無効になり、Mobile Link サーバはスクリプトの実行を監視しません。

タイムアウトスレッショルドが 0 以外の値の場合、Mobile Link サーバは指数関数的な方法で警告メッセージを表示します。警告は、実行時間が指定された時間を初めて過ぎたときに表示されます。また、指定された時間の 2 倍、次に 4 倍、というような時間を過ぎたときにも表示されます。

警告メッセージには、現在の同期に使用されている接続 ID と、コンテキストが含まれます。コンテキストには、リモート ID、ML ユーザ名、変更後のユーザ名、トランザクション、テーブル名、ロー値、スクリプトバージョンがあれば表示されます。タイムアウト警告コンテキストは、Mobile Link サーバの冗長設定に関係なく表示されます。

統合データベースが Oracle データベースサーバで実行されていて、タイムアウト警告メッセージが表示されたときは、DBA 権限を持つデータベースユーザが統合データベースをチェックして問題の原因を特定することが必要な場合があります。警告メッセージから、同期によって使用された接続の ServiceName および SERIAL# を知ることができます。同期接続が停止されると、Mobile Link サーバは現在の同期を終了します。

参照

● 「-tf mlsrv16 オプション」 81 ページ

-tf mlsrv16 オプション

このオプションは、実行時間が -tc によって指定されたタイムアウトを過ぎた場合に、Mobile Link サーバに SQL スクリプトを失敗にさせる場合に使用します。このオプションは、統合データベースが Oracle サーバで実行されている場合には使用できません。

構文

mlsrv16 -c "*connection string***" -tf** …

備考

SQL スクリプトが失敗した場合、Mobile Link サーバは、ローをスキップし (スクリプトがアップロードスクリプトの場合や、handle_error スクリプトが 1000 を返した場合)、同期を継続するか、同期をアボートします。

このオプションが指定され、Mobile Link サーバが Oracle サーバに対して実行されている場合、Mobile Link サーバは、警告メッセージを表示します。

-tc 0 が指定されている場合、このオプションは無視されます。

-ts mlsrv16 オプション

Mobile Link サーバのトレースセッションを設定します。

構文

mlsrv16 -c "*connection-string***" -ts** *session-name*(*session-option*=[*option-value*;...])

セッション名は **logging** にする必要があります。

セッションオプション	オプション値
events	システムトレースイベントのカンマで区切られたリストサポートされるイベントは、Info、Warning、および Error です。
targets	*target-type*(*target-option*=*value*[;...]) ここで *target-type* は **file** にできます。

ターゲットオプションは、名前と値のペアとして指定されます。ターゲットファイルには、次のオプションが用意されていることがあります。

ターゲットオプション名	予期される値	説明
filename_prefix	文字列	パス付きまたはパスなしの ETD ファイル名プレフィクスすべての ETD ファイルには、*.etd* という拡張子が付きます。このパラメータは必須です。
max_size	整数	ファイルの最大サイズ (バイト単位)。デフォルトは 0 で、これはファイルサイズに上限がないことを意味し、ディスク領域が許す限り大きくなっていきます。指定したサイズに達すると、新しいファイルが開始されます。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

ターゲットオプション名	予期される値	説明
num_files	整数	イベントトレース情報が書き込まれるファイル数で、max_size が設定されている場合にのみ使用されます。すべてのファイルが指定した最大サイズに達すると、Mobile Link サーバは古いファイルの上書きを開始します。
flush_on_write	yes、true、no、false	記録されるイベントが発生するたびにディスクバッファをフラッシュするかどうかを制御する値。値には、yes、true、no、false を設定できます。デフォルトは false です。このパラメータをオンにすると、多くのトレースイベントが記録される場合、Mobile Link サーバのパフォーマンスが低下することがあります。
compressed	yes、true、no、false	ディスク領域を節約するための ETD ファイルの圧縮を制御する値。デフォルトは false です。

備考

オプションの -ts logging 部分の後で指定するすべての情報は、スペースなしで指定する必要があります。

参照

- 「イベントトレースデータ (ETD) ファイル管理ユーティリティ (dbmanageetd)」『SQL Anywhere サーバ データベース管理』

例

次に、-ts オプションの例を示します。

```
-ts
logging{events=Info,warning,Error;targets=file{filename_prefix=mls_etd;max_size=10000000;num_files=
10;flush_on_write=true}}
```

-tx mlsrv16 オプション

アップロードのトランザクションを使用している場合、このオプションは、トランザクションのグループをバッチにし、まとめてコミットします。

構文

mlsrv16 -c "*connection-string***" -tx** *count* ...

備考

このオプションを使用すると、アップロードのトランザクションを行うときのパフォーマンスが向上します。

count は、負以外の任意の値にすることができます。デフォルトは 1 であり、これは各トランザクションを別々にコミットすることを意味します。値 0 を使用すると、すべてのトランザクションがアップロードされてからコミットが 1 回実行されます。

count の最適な値は、パフォーマンステストによってのみ決定できます。

参照

- 「-tu dbmlsync オプション」『Mobile Link クライアント管理』

-ud mlsrv16 オプション

Mobile Link をデーモンとして実行するように指示します。

構文

mlsrv16 -c "*connection-string***" -ud** ...

備考

このオプションは UNIX プラットフォームにのみ適用されます。

参照

- 「現在のセッション外での Mobile Link サーバの使用」33 ページ

-ui mlsrv16 オプション

X Window サーバがサポートされている Linux で、使用可能な表示がない場合にシェルモードで Mobile Link サーバを起動します。

構文

mlsrv16 -c "*connection-string***" -ui** ...

備考

-ui を指定すると、サーバは使用可能な表示を探そうとします。X-Window Server が実行されていなかったなどの理由で、使用可能な表示が見つからなかった場合は、Mobile Link サーバはシェルモードで起動されます。

-ux mlsrv16 オプション

Linux の場合に、メッセージを表示する、Mobile Link サーバのメッセージウィンドウを開きます。

構文

mlsrv16 -c "*connection-string* **-ux** ...

備考

-ux が指定されている場合、Mobile Link サーバは使用可能な表示を見つけます。たとえば、DISPLAY 環境変数が設定されていなかったり、X-Window Server が実行されていなかったりしたために、使用可能な表示が見つからなかった場合、Mobile Link サーバは起動できません。

クワイエットモードで Mobile Link サーバのメッセージウィンドウを実行するには、-q を使用します。

Windows では、Mobile Link サーバのメッセージウィンドウが自動的に表示されます。

参照

● 「-q mlsrv16 オプション」73 ページ

-v mlsrv16 オプション

メッセージログファイルにログを取る情報を指定できます。

構文

mlsrv16 -c "*connection-string***" -v**[*levels*] ...

備考

このオプションは、メッセージログファイルに書き込まれるメッセージのタイプを制御します。

-v を単独で指定すると、Mobile Link サーバは、各同期について最小量の情報を書き込みます。多くのレベルを指定するほど、メッセージログファイルへの出力が詳細になります。

冗長レベルを上げ過ぎるとパフォーマンスに影響する可能性があるので、冗長レベルを上げるのは開発中だけにしてください。

対象とする Mobile Link ユーザまたはリモート ID に対して、Mobile Link サーバが異なるログの冗長性を使用するように設定できます。Mobile Link サーバは、5 分ごとに ml_property テーブルをチェックし、Mobile Link ユーザまたはリモート ID の冗長性設定を調べます。 「対象 Mobile Link ユーザとリモート ID に対するログの冗長性」662 ページを参照してください。

バイト長が 32767 バイトを超える CHAR、VARCHAR、NCHAR、または NVARCHAR カラムが同期されるとき、Mobile Link サーバでは冗長性を持つカラムの値の内容は一部のみが表示されます。この場合は、データの最初のチャンク (最長 100 バイト) が表示されます。このことは i レベル、q レベル、r レベルに適用されます。

使用可能なレベルは次のとおりです。たとえば -vnrsu など、一度に 1 つまたは複数のオプションを使用できます。

- **+** すべての小文字の冗長性レベルをオンにします。
- **c** 呼び出された各同期スクリプトの内容を表示します。このレベルには、s の機能が含まれます。
- **e** システムイベントスクリプトを表示します。システムイベントスクリプトを使用して、Mobile Link システムテーブルに問い合わせ、管理します。
- **f** 最初の読み込みエラーを表示します。このオプションは、負荷分散装置がサーバの活性を確認するとき、データを送信しない接続を確立して同期が失敗したために発生したエラーのログを取ります。このオプションを使用して、負荷分散装置が正常に活性チェックを実行していることを確認します。

  TCP/IP オプション **ignore** も参照してください。 詳細については、「-x mlsrv16 オプション」92 ページを参照してください。
- **h** 同期中のリモートスキーマを示します。
- **i** アップロードされた各ローのカラム値を表示します。アップロードとダウンロードされた各ローのカラム値を表示する -vr オプションの代わりにこのオプションを使用すると、ログに記録されるデータ量が減少します。-vi と -vq を同時に指定することは、-vr を指定することと同じです。
- **k** キャッシュが増大または縮小すると、ログに行を出力します。この出力を使用して、ロードのテスト中に最適なキャッシュサイズを見つけることができます。このキャッシュサイズは、システムを配備するときに静的キャッシュサイズとして使用できます。
- **m** 同期が完了するたびに、各同期と各同期フェーズの継続時間をログに出力します。同期フェーズは以下に示します。これらは、Mobile Link プロファイラに表示されるフェーズと同じです。すべての時間はミリ秒 (ms) で表されます。
  - **[同期要求]** Mobile Link クライアントと Mobile Link サーバの間のネットワーク接続を確立してから、アップロードストリームの最初のバイトを受信するまでの時間。
  - **[アップロードの受信]** Mobile Link サーバでアップロードストリームの最初のバイトが受信されてから、Mobile Link クライアントからのアップロードストリームが完全に受信されるまでの時間。ダウンロード専用同期でもこの時間が長くなる場合があります。時間は、アップロードストリームのサイズと、転送用のネットワーク帯域幅によって異なります。
  - **[DB ワーカの取得]** 空いているデータベースワーカスレッドを取得するために必要な時間。
  - **[接続]** 新しいデータベース接続が必要な場合に、データベースワーカスレッドでデータベース接続を確立するために必要な時間。たとえば、前の接続でのエラーの後や、スクリプトのバージョンが変更された場合に新しい接続が必要になります。

- **[ユーザを認証]** ユーザの認証に必要な時間。
- **[同期の開始]** begin_synchronization イベントが定義されている場合はこのイベントに必要な時間と、各サブスクリプションの last_upload_time をフェッチするのに必要な時間の合計。
- **[アップロードの適用]** アップロードしたデータを統合データベースに適用するために必要な時間。
- **[ダウンロードの準備]** prepare_for_download イベントに必要な時間。
- **[ダウンロードのフェッチ]** 統合データベースからダウンロードするローをフェッチしてダウンロードストリームを作成するために必要な時間。フェッチダウンロードフェーズには、ダウンロードストリームを作成する時間は含まれません。それはダウンロードの送信フェーズで行われます。大量のダウンロードには非常に長い時間がかかる場合があります。そのような場合、ダウンロードはメモリに収まりません。
- **[同期の終了]** end_synchronization イベントに必要な時間。この後、データベースワーカスレッドが解放されます。このフェーズは、ダウンロードストリームがリモートデータベースに送信される前に発生します。
- **[ダウンロードの送信]** ダウンロードストリームをリモートデータベースに送信するために必要な時間。時間は、ダウンロードストリームのサイズと、転送用のネットワーク帯域幅によって異なります。アップロード専用同期の場合、ダウンロードストリームは単なるアップロード確認です。

  ダウンロードの送信フェーズには、ダウンロードストリームを作成する時間が含まれます。そのため、大量のダウンロードには非常に長い時間がかかる場合があります。そのような場合、ダウンロードはメモリには収まりません。
- **[ダウンロード確認の待機]** ダウンロードがリモートデータベースに適用され、リモートデータベースからダウンロード確認が送信されるのを待つ時間。このフェーズは、Mobile Link クライアントでダウンロード確認が有効になっている場合にのみ表示されます。
- **[ダウンロード確認の DB ワーカの取得]** ダウンロード確認を受信してから、データベースワーカスレッドが空くのを待つ時間。このフェーズは、Mobile Link クライアントでダウンロード確認が有効になっている場合にのみ表示されます。
- **[ダウンロード確認の接続]** 新しいデータベース接続が必要な場合に、データベースワーカスレッドでデータベース接続を確立するために必要な時間。このフェーズは、Mobile Link クライアントでダウンロード確認が有効になっている場合にのみ表示されます。
- **[非ブロッキングダウンロード確認]** publication_nonblocking_download_ack 接続と nonblocking_download_ack 接続のイベントに必要な時間。このフェーズは、Mobile Link クライアントでダウンロード確認が有効になっている場合にのみ表示されます。

各値は、値の検索が容易になるように、"PHASE:" で始まります。

次に、さまざまな同期フェーズの期間を表している出力例を示します。

```
I. 2008-06-05 14:48:36. <1> PHASE: start_time: 2008-06-05 14:48:36.048
I. 2008-06-05 14:48:36. <1> PHASE: duration: 175
```

```
I. 2008-06-05 14:48:36. <1> PHASE: sync_request: 0
I. 2008-06-05 14:48:36. <1> PHASE: receive_upload: 19
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect: 18
I. 2008-06-05 14:48:36. <1> PHASE: authenticate_user: 51
I. 2008-06-05 14:48:36. <1> PHASE: begin_sync: 69
I. 2008-06-05 14:48:36. <1> PHASE: apply_upload: 0
I. 2008-06-05 14:48:36. <1> PHASE: prepare_for_download: 1
I. 2008-06-05 14:48:36. <1> PHASE: fetch_download: 4
I. 2008-06-05 14:48:36. <1> PHASE: wait_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: end_sync: 0
I. 2008-06-05 14:48:36. <1> PHASE: send_download: 10
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: nonblocking_download_ack: 0
```

- **n**　同期ごとのローカウントの合計を表示します。
- **o**　SQL パススルーアクティビティを表示します。
- **p**　同期ごとのリモート進行オフセットと統合進行オフセットの両方を表示します。
- **q**　ダウンロードされた各ローのカラム値を表示します。アップロードとダウンロードされた各ローのカラム値を表示する -vr オプションの代わりにこのオプションを使用すると、ログに記録されるデータ量が減少します。-vi と -vq を同時に指定することは、-vr を指定することと同じです。
- **r**　アップロードまたはダウンロードされた各ローのカラム値を表示します。アップロードされた各ローのカラム値だけをログに記録する場合は、-vi を使用します。ダウンロードされた各ローのカラム値だけをログに記録する場合は、-vq を使用します。
- **R**　同期の場合のみ、各ログメッセージ内のリモート ID を表示します。Mobile Link サーバはプレフィクス yyyy-mm-dd hh:mm:ss. <sync_id> (remote_id,) をログエントリに追加します。

  このオプションを -vU オプションと一緒に使用すると、ログメッセージ内のユーザ名も表示されます。

  これらの 2 つのコマンドラインオプションは、-v+ オプションの影響を受けません。つまり、-v+ オプションを使用しても、Mobile Link サーバでは、ログメッセージにリモート ID または Mobile Link ユーザ名は追加されません。
- **s**　呼び出された各同期スクリプトの名前を表示します。
- **t**　ODBC 標準フォーマットのスクリプトから生成された、変換された SQL を表示します。このレベルには、c の機能が含まれます。次の例は、SQL Anywhere での文の自動変換を示します。

  ```
  I. 2009-02-11 11:02:14. [102]: begin_upload synch2
  { call SynchLogLine( ?, ?, 'begin_upload' ) }
  I. 2009-02-11 11:02:14. [102]: Translated SQL:
  call SynchLogLine( ?, ?, 'begin_upload' )
  ```

  次の例は、同じ文の Microsoft SQLServer での変換を示します。

```
I. 2009-02-11 11:03:21. [102]: begin_upload synch2
{ call SynchLogLine( ?, ?, 'begin_upload' ) }
I. 2009-02-11 11:03:21. [102]: Translated SQL:
EXEC SynchLogLine ?, ?, 'begin_upload'
```

- **u**　未定義のテーブルスクリプトを表示します。このことは、経験の浅いユーザが同期処理とイベントのフローを理解するのに役立ちます。

- **U**　同期の場合のみ、各ログメッセージ内のユーザ名を表示します。Mobile Link サーバはプレフィクス yyyy-mm-dd hh:mm:ss. <sync_id> (,user_name) をログエントリに追加します。

  このオプションを -vR オプションと一緒に使用すると、ログメッセージ内のリモート ID も表示されます。

  これらの 2 つのコマンドラインオプションは、-v+ オプションの影響を受けません。つまり、-v+ オプションを使用しても、Mobile Link サーバでは、ログメッセージにリモート ID または Mobile Link ユーザ名は追加されません。

参照

- 「Mobile Link の同期統計のプロパティ」208 ページ

-w mlsrv16 オプション

同時に使用するデータベースワーカスレッド数の初期値を設定します。スレッド数の最大値は、-wm オプションで指定されます。

構文

mlsrv16 -c "*connection-string***" -w** *count* ...

備考

各データベースワーカスレッドでは、同期要求を一度に 1 つ受け入れます。ただし、その他のすべてのデータベースワーカスレッドと同時に受け入れます。

データベースワーカスレッドはそれぞれ、統合データベースへの接続を 1 つ使用します。Mobile Link サーバは、接続をさらに 1 つ管理用に開きます。したがって、Mobile Link サーバから統合データベースへの接続の最小数は、*count* + 1 となります。

データベースワーカスレッド数は Mobile Link 同期スループットに大きく影響するので、特定の同期設定に最適なデータベースワーカスレッド数を判別するためのテストを実行する必要があります。データベースワーカスレッド数によって同時に統合データベースでアクティブにできる同期の数が決まります。残りの同期はキューイングされてデータベースワーカスレッドが使用可能になるのを待機します。データベースワーカスレッドを追加するとスループットが向上する可能性がありますが、アクティブな同期の間で競合が発生する確率も高くなります。データベースワーカスレッドを追加していくと、ある時点で、複数の同期を同時に行うメリットよりも競合が発生する確率の方が大きくなり、スループットが低下します。

このオプションの設定値は -wu オプションのデフォルト設定にもなります。-wu オプションは、統合データベースに同時にアップロードできるスレッド数を制限するためのオプションです。これは、ダウンロードを行うデータベースワーカスレッドの最適数が、アップロードを行うデー

タベースワーカスレッドの最適数より多い場合に役立ちます。データベースワーカスレッド数を大きく設定し (-w を使用)、アップロードを同時に適用できるデータベースワーカスレッド数を小さく設定する (-wu を使用) と、最高のスループットを達成できる場合があります。通常、-wu の最適数は統合データベースによって異なり、リモートデータベースの処理速度またはネットワーク速度とはあまり関係ありません。したがって、-w を使用してスレッド数を増やす場合、-wu を使用して同時にアップロードできるスレッド数を制限できます。

データベースワーカスレッド数のデフォルトは **5** です。

参照

- 「-wm mlsrv16 オプション」 90 ページ
- 「-wu mlsrv16 オプション」 91 ページ
- 「-sm mlsrv16 オプション」 79 ページ
- 「-cn mlsrv16 オプション」 55 ページ

-wm mlsrv16 オプション

データベースワーカスレッドの最大数を設定します。

構文

mlsrv16 -c "*connection-string***" -wm** *count* ...

備考

Mobile Link サーバは、必要に応じて、システムのパフォーマンスをモニタしながら、データベースワーカスレッドの数を自動的に調整します。Mobile Link サーバは、-w オプションで設定される初期値と、-wm オプションで設定される最大値の間の任意の値を使用します。

この機能によって、使用環境では少ないロードテストで、スループットを向上させることができます。-w と -wm で十分な範囲が指定されていれば、Mobile Link サーバは、スループットが最大になるデータベースワーカスレッド数を自動的に選択できます。ただし、データベースワーカスレッド数の調整に使用されるヒューリスティックは、常にうまく動作するとは限りません。また、-w と -wm によって設定された範囲内では、どうしても最大のスループットを達成できない場合もあります。展開された環境の特性を考慮したロードテストによってのみ、スループットが最大になるデータベースワーカスレッド数を正確に導き出すことができます。

この値を設定しない場合、デフォルトの最大データベースワーカスレッド数は、-w オプションで設定される値に等しくなります。 -wm オプションを設定しない場合は、データベースワーカスレッド数は -w の値に固定されるため、Mobile Link による自動調整は行われません。

参照

- 「-w mlsrv16 オプション」 89 ページ
- 「-wu mlsrv16 オプション」 91 ページ
- 「-sm mlsrv16 オプション」 79 ページ
- 「-cn mlsrv16 オプション」 55 ページ
- 「データベースワーカスレッドの自動調整」 144 ページ

-wu mlsrv16 オプション

アップロードを同時に統合データベースに適用できるデータベースワーカスレッドの最大数を設定します。

構文

mlsrv16 -c "*connection-string***" -wu** *count* ...

備考

-wu オプションを使用して、アップロードを同時に統合データベースに適用できるデータベースワーカスレッド数を制限します。制限値に達すると、統合データベースへのアップロードの適用準備が完了しているデータベースワーカスレッドは、別のデータベースワーカスレッドがアップロードを終了するまで待機します。

統合データベースで発生する競合の最も一般的な原因は、アップロードを同時に適用するデータベースワーカスレッドが多すぎることです。通常、ダウンロードで競合が発生することははるかに少ないので、ダウンロードは mlsrv16 -w オプションのみによって制限されます。そのため、-w の設定は -wu の設定以上にしてください。

デフォルトでは、すべてのデータベースワーカスレッドで同時にアップロードを適用できます。使用されるデータベースワーカスレッドの数は -w オプションで設定します。デフォルトは **5** です。

-wu を指定しない場合、同時に実行される任意またはすべてのデータベースワーカスレッドに、アップロードが適用される可能性があります。-wu を指定する場合は、アップロードは特定の数のデータベースワーカスレッドにのみ適用されます。その場合、Mobile Link サーバがスループットを向上させようとしてデータベースワーカスレッドの数を増加させるため、一時的に競合の発生が多くなる場合があります。こうした競合が検出されると、スレッド数を減少させます。

アップロードがダウンロード専用同期と常に混在するような負荷の高い環境では、-wu オプションの設定をおすすめします。

例

LAN と PC 上のリモートデータベースを使用するパイロット設定では、アップロード専用同期とダウンロード専用同期の両方に対して、データベースワーカスレッドの最適数は約 10 であり、それが統合データベースの CPU 使用率 100% に相当します。データベースワーカスレッドの数が少ないと、スループットが低下し、統合データベースの CPU 使用率が低下します。データベースワーカスレッドの数が多くても、統合データベースの処理速度はデータベースワーカスレッド数が 10 の場合と同じなので、スループットは向上しません。

参照

- 「-w mlsrv16 オプション」 89 ページ
- 「-wm mlsrv16 オプション」 90 ページ
- 「-sm mlsrv16 オプション」 79 ページ

-x mlsrv16 オプション

Mobile Link サーバが同期要求を受信するために使用するネットワークプロトコルオプションを設定します。

> **注意**
> 別途ライセンスが必要な必須コンポーネント
>
> FIPS 認定の暗号化には別のライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。
>
> 「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

構文

mlsrv16 -c "*connection-string***" -x** *protocol***[***protocol-options***] [-x** *protocol***[***protocol-options***]** ... **]** ...

protocol : **tcpip** | **tls** | **http** | **https** | **oe**

protocol-options : **(** *option***=***value*; ... **)**

備考

使用するすべてのプロトコルに対して -c オプションを指定する必要があります。たとえば、Mobile Link で TCP/IP と HTTP の両方を受信するには、次のような指定を行います。

```
mlsrv16 -x tcpip(port=1234) -x http(port=2222)
```

デフォルトは tcpip でポート 2439 を使用します。

パラメータ

許可されている protocol の値は次のとおりです。

- **tcpip**　TCP/IP を使用した接続を受け入れます。
- **tls**　TCP/IP およびトランスポートレイヤセキュリティ (TLS) を使用した接続を受け入れます。
- **http**　標準の HTTP Web プロトコルを使用した接続を受け入れます。
- **https**　安全なトランザクションを処理する HTTP の変形プロトコルを使用した接続を受け入れます。HTTPS プロトコルは、RSA 暗号化を使用して HTTP over SSL/TLS を実装します。
- **oe**　Relay Server を使用する場合は、統合化 Outbound Enabler を使用します。サーバのコマンドラインに複数の **-x oe** プロトコルを含めることはできません。

option=value の形式で次のネットワークプロトコルオプションを指定することもできます。個々の複数のオプションは、セミコロンで区切ってください。

- **TCP/IP オプション**　tcpip プロトコルを指定する場合は、次のプロトコルオプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

TCP/IP プロトコルオプション	説明
collect_network_data={ **yes** | **no** }	同期のたびに同期スクリプトでネットワーク情報を読み取れるようにします。NetworkData インタフェース [Mobile Link サーバ Java]497 ページと NetworkData インタフェース [Mobile Link サーバ .NET]593 ページを参照してください。
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
ignore=*hostname*	接続を確立する場合に、Mobile Link サーバが無視するホスト名または IP アドレス。このオプションにより、考えられる最低レベルでの負荷分散装置からの要求を無視することができるため、Mobile Link サーバログや Mobile Link Profiler 出力ファイルでの過剰な出力を回避できます。無視するホストは複数指定できます。たとえば、-x tcpip(ignore=lb1;ignore=123.45.67.89) の形式で指定します。コマンドラインで -x の複数のインスタンスを指定した場合、すべてのインスタンスでホストが無視されます。たとえば、-x tcpip(ignore=1.1.1.1) -x http と指定した場合、1.1.1.1 の接続は TCP/IP と HTTP の両方のストリームで無視されます。
port=*portnumber*	Mobile Link サーバが受信に使用するソケットポート番号。デフォルトのポートは 2439 です。これは、Mobile Link サーバの IANA 登録ポート番号です。

- **TLS (TCP/IP とトランスポートレイヤセキュリティ) オプション**　tls プロトコル (トランスポートレイヤセキュリティを使用する TCP/IP) を指定する場合は、次のプロトコルオプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

TLS プロトコルオプション	説明
collect_network_data={ **yes** | **no** }	同期のたびに同期スクリプトでネットワーク情報を読み取れるようにします。NetworkData インタフェース [Mobile Link サーバ Java]497 ページと NetworkData インタフェース [Mobile Link サーバ .NET]593 ページを参照してください。

TLS プロトコルオプション	説明
e2ee_private_key=*file*	RSA プライベートキーを含む、PEM または DER でコード化されたファイル。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。 PEM および DER でコード化されたファイルは、createkey ユーティリティを使用して作成します。 「キーペアジェネレータユーティリティ (createkey)」『SQL Anywhere サーバ データベース管理』を参照してください。
e2ee_private_key_password=*password*	プライベートキーファイルのパスワード。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。 このオプションが指定された場合、e2ee_private_key パラメータも指定する必要があります。 このパスワードを Mobile Link サーバのコマンドラインで表示されないようにするには、dbfhide ユーティリティを使用します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
fips={**yes**|**no**}	RSA で TLS プロトコルを使用している場合、fips=yes を指定して、TCP/IP プロトコルと、FIPS 認定の暗号化アルゴリズムを使用して接続を受け入れることができます。FIPS 認定接続では、別の FIPS 140-2 認定ソフトウェアを使用します。FIPS 認定暗号化のない RSA 暗号化を使用しているサーバは、fips オプションが有効化された RSA を使用するクライアントと互換性があります。fips オプションが有効化された RSA を使用するサーバは、fips オプションが有効化されていない RSA を使用しているクライアントと互換性があります。
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
identity=*identity-file*	サーバ認証で使用される ID ファイルのパスとファイル名。

TLS プロトコルオプション	説明
identity_password=*password*	ID ファイルのパスワードを指定する省略可能なパラメータ。 このオプションが指定された場合、identity オプションも指定する必要があります。 「トランスポートレイヤセキュリティ」『SQL Anywhere サーバ データベース管理』を参照してください。 このパスワードを Mobile Link サーバのコマンドラインで表示されないようにするには、dbfhide ユーティリティを使用します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
ignore=*hostname*	接続を確立する場合に、Mobile Link サーバが無視するホスト名または IP アドレス。このオプションにより、考えられる最低レベルでの負荷分散装置からの要求を無視することができるため、Mobile Link サーバログや Mobile Link Profiler 出力ファイルでの過剰な出力を回避できます。無視するホストは複数指定できます。たとえば、-x tcpip(ignore=lb1;ignore=123.45.67.89) の形式で指定します。
port=*portnumber*	Mobile Link サーバが受信に使用するソケットポート番号。デフォルトのポートは 2439 です。これは、Mobile Link サーバの IANA 登録ポート番号です。
trusted_certificates=*certificate_file*	このオプションを使用してクライアント証明書が有効であることを確認し、さらに NetworkData.ClientCertificates API を使用して、authenticate_user スクリプトで証明書を認証します。 NetworkData インタフェース [Mobile Link サーバ Java]497 ページと NetworkData インタフェース [Mobile Link サーバ .NET]593 ページを参照してください。

- **HTTP オプション**　http プロトコルを指定する場合は、次のプロトコルオプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

HTTP オプション	説明
buffer_size=*number*	Mobile Link サーバから送信される HTTP メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTP メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65536 バイトです。
collect_network_data={ **yes** | **no** }	同期のたびに同期スクリプトでネットワーク情報を読み取れるようにします。NetworkData インタフェース [Mobile Link サーバ Java]497 ページと NetworkData インタフェース [Mobile Link サーバ .NET]593 ページを参照してください。
header_limit=*number*	HTTP 要求で送信可能なヘッダデータの最大容量。要求が指定した値を上回ると、サーバは HTTP エラーコードを返し、要求をアボートします。たとえば、-x http(header_limit=200000) により上限が 200000 バイトになります。デフォルト値は 64000 バイトです。
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
log_bad_request={ **yes** | **no** }	yes に設定すると、Mobile Link サーバは不完全または予期しない HTTP 要求を受信した場合にエラーを出力します。これらのエラーは、-vf オプションによって出力されるエラーに似ています。デフォルトは no です。 「-v mlsrv16 オプション」85 ページを参照してください。
port=*portnumber*	Mobile Link サーバが受信に使用するソケットポート番号。デフォルトのポートは 80 です。
version=*http-version*	Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルトバージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。

- **HTTPS オプション**　HTTPS プロトコルでは、トランスポートレイヤセキュリティで RSA デジタル証明書を使用します。FIPS 暗号化を指定すると、プロトコルは、HTTPS と互換性のある、別の FIPS 140-2 承認ソフトウェアを使用します。

  詳細については、「トランスポートレイヤセキュリティを使用する Mobile Link サーバの起動」『SQL Anywhere サーバ データベース管理』を参照してください。

  https プロトコルを指定する場合は、オプションで次のプロトコルオプションを指定できます (これらのオプションでは、大文字と小文字は区別されます)。

HTTPS オプション	説明
buffer_size=*number*	Mobile Link サーバから送信される HTTPS メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTPS メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65536 バイトです。
collect_network_data={ **yes** | **no** }	同期のたびに同期スクリプトでネットワーク情報を読み取れるようにします。NetworkData インタフェース [Mobile Link サーバ Java]497 ページと NetworkData インタフェース [Mobile Link サーバ .NET]593 ページを参照してください。
e2ee_private_key=*file*	RSA プライベートキーを含む、PEM または DER でコード化されたファイル。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。 PEM および DER でコード化されたファイルは、createkey ユーティリティを使用して作成します。「キーペアジェネレータユーティリティ (createkey)」『SQL Anywhere サーバ データベース管理』を参照してください。

HTTPS オプション	説明
e2ee_private_key_password=*password*	プライベートキーファイルのパスワード。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。 このオプションが指定された場合、e2ee_private_key オプションも指定する必要があります。 このパスワードを Mobile Link サーバのコマンドラインで表示されないようにするには、dbfhide ユーティリティを使用します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
fips={**yes**|**no**}	RSA で TLS プロトコルを使用している場合、fips=yes を指定して、TCP/IP プロトコルと、FIPS 認定の暗号化アルゴリズムを使用して接続を受け入れることができます。FIPS 認定接続では、別の FIPS 140-2 認定ソフトウェアを使用します。FIPS 認定暗号化のない RSA 暗号化を使用しているサーバは、fips オプションが有効化された RSA を使用するクライアントと互換性があります。fips オプションが有効化された RSA を使用するサーバは、fips オプションが有効化されていない RSA を使用しているクライアントと互換性があります。
header_limit=*number*	HTTPS 要求で送信可能なヘッダデータの最大容量。要求が指定した値を上回ると、サーバはエラーコードを返し、要求をアボートします。たとえば、-x https(header_limit=200000) により上限が 200000 バイトになります。デフォルト値は 64000 バイトです。
host= *hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
identity=*server-identity*	サーバ認証で使用される ID ファイルのパスとファイル名。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

HTTPS オプション	説明
identity_password=*password*	ID ファイルのパスワードを指定する省略可能なパラメータ。 このオプションが指定された場合、identity オプションも指定する必要があります。 「トランスポートレイヤセキュリティ」『SQL Anywhere サーバ データベース管理』を参照してください。 このパスワードを Mobile Link サーバのコマンドラインで表示されないようにするには、dbfhide ユーティリティを使用します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
log_bad_request={ **yes** | **no** }	yes に設定すると、Mobile Link サーバは不完全または予期しない HTTP 要求を受信した場合にエラーを出力します。これらのエラーは、-vf オプションによって出力されるエラーに似ています。デフォルトは no です。 「-v mlsrv16 オプション」 85 ページを参照してください。
port=*portnumber*	Mobile Link サーバが受信に使用するソケットポート番号。ポート番号は、Mobile Link サーバがモニタするように設定されているポートと一致させます。デフォルトのポートは 443 です。
trusted_certificates=*certificate_file*	このオプションを使用してクライアント証明書が有効であることを確認し、さらに NetworkData.ClientCertificates API を使用して、authenticate_user スクリプトで証明書を認証します。 NetworkData インタフェース [Mobile Link サーバ Java]497 ページと NetworkData インタフェース [Mobile Link サーバ .NET]593 ページを参照してください。

HTTPS オプション	説明
version=*http-version*	Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルトバージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。

- **OE オプション**　oe プロトコルを指定する場合は、次のプロトコルオプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

OE プロトコルオプション	説明
buffer_size=*number*	Mobile Link サーバから送信される HTTP メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTP メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65536 バイトです。
config=*file*	Outbound Enabler 設定ファイル。ファイル内で -cs オプションを指定していない場合、スタンドアロン Outbound Enabler の設定ファイルと同じフォーマットを持ちます。 設定ファイルは 1 つのみ指定してください。
header_limit=*number*	OE 要求で送信可能なヘッダデータの最大容量。要求が指定した値を上回ると、サーバはエラーコードを返し、要求をアボートします。たとえば、-x oe(header_limit=200000) により上限が 200000 バイトになります。デフォルト値は 64000 バイトです。
log_bad_request={**yes**|**no**}	yes に設定すると、Mobile Link サーバは不完全または予期しない HTTP 要求を受信した場合にエラーを出力します。これらのエラーは、-vf オプションによって出力されるエラーに似ています。デフォルトは no です。 「-v mlsrv16 オプション」85 ページを参照してください。

OE プロトコルオプション	説明
version=*http-version*	Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルトバージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。

例

次のコマンドラインは TCP/IP ポートを 12345 に設定します。

```
mlsrv16 -c "DSN=SQL Anywhere 16 CustDB;UID=DBA;PWD=sql" -x tcpip(port=12345)
```

次の例では、セキュリティのタイプ (RSA)、サーバ ID ファイル、サーバのプライベートキーを保護する ID パスワードを指定します。

```
mlsrv16 -c "DSN=my_cons"
 -x tls(identity=c:¥test¥serv_rsa1.crt;identity_password=pwd)
```

次の例は前の例と似ていますが、ID ファイル名にスペースが含まれる点だけが異なります。

```
mlsrv16 -c "DSN=my_cons"
 -x "tls(identity=c:¥Program Files¥test¥serv_rsa1.crt;identity_password=pwd)"
```

次の例は、HTTPS を使用したエンドツーエンド暗号化の使い方を示します。

```
mlsrv16 -c "DSN=my_cons" -x https(identity=my_identity.crt;
identity_password=my_id_pwd;e2ee_private_key=my_pk.pem;
e2ee_private_key_password=my_pk_pwd)
```

次の例は、統合化 Outbound Enabler の使用方法を示します。

```
mlsrv16 -c "DSN=my_cons" -x oe(config=oeconfig.txt)
```

上記の例では、*oeconfig.txt* ファイルには次の内容が含まれています。

```
-f farm.name
-id servername
-t a18b2e37dbc296322ec5a6eded6ba896
-cr "host=relayserver.sybase.com;https=0;port=80"
```

Java 用の trusted_certificates の例

次に、NetworkData インタフェースを使用して安全な同期から証明書情報を取得する方法の例を示します。

```
public class OrderProcessor {
   DBConnectionContext _cc;

   public OrderProcessor( DBConnectionContext cc ) {
      _cc = cc;
   }
```

```
    // The method used for the authenticate_user event.
    public void AuthUser() {
        NetworkData nd = _cc.getNetworkData();
        if( nd != null ) {
            if( nd.isTLS() ) {
                CertPath certs = nd.getCertificateChain();
                if( certs != null ) {
                    System.out.println( "  client-side cert:" );
                    int n = 1;
                    for( Certificate c : certs.getCertificates() ) {
                        System.out.println( "    cert " + n++ );
                        X509Certificate c509 = (X509Certificate) c;
                        System.out.println( "      Subject: " + c509.getSubjectX500Principal().getName() );
                        System.out.println( "      Issuer: " + c509.getIssuerX500Principal().getName() );
                    }
                } else {
                    System.out.println( "  no client cert" );
                }
            }
        }
    }
}
```

次の SQL 文を実行して、Java メソッドを登録します。

```
ml_add_java_connection_script( <version>, 'authenticate_user', 'OrderProcessor.AuthUser' )
```

次の 2 つの例で、Mobile Link コマンドラインに追加するオプションを示します。最初は HTTPS のための例で、2 番目は TLS のための例です。

```
mlsrv16 -c <connection_string> -x https(collect_network_data=1;trusted_certificates=<certificate_file>) -sl java

mlsrv16 -c <connection_string> -x tls(collect_network_data=1;trusted_certificates=<certificate_file>) -sl java
```

.NET 用の trusted_certificates の例

次に、NetworkData インタフェースを使用して安全な同期から証明書情報を取得する方法の例を示します。

```
public class OrderProcessor {
    DBConnectionContext _cc;

    public OrderProcessor( DBConnectionContext cc ) {
        _cc = cc;
    }

    public void AuthUser() {
        NetworkData nd = _cc.NetworkData;
        if( nd != null ) {
            if( nd.IsTLS ) {
                X509Certificate2Collection certs = nd.ClientCertificates;
                if( certs != null ) {
                    PrintLn( "  client-side cert:" );
                    int n = 1;
                    foreach( X509Certificate2 x509 in certs ) {
                        PrintLn( "    cert " + n++ );
                        PrintLn( "      Subject: " + x509.SubjectName.Name );
                        PrintLn( "      Issuer: " + x509.IssuerName.Name );
                    }
```

```
            }
          }
        }
      }
    }
```

次の SQL 文を実行して、.NET メソッドを登録します。

```
ml_add_dnet_connection_script( <version>, 'authenticate_user', 'OrderProcessor.AuthUser' )
```

次の 2 つの例で、Mobile Link コマンドラインに追加するオプションを示します。最初は HTTPS のための例で、2 番目は TLS のための例です。

```
mlsrv16 -c <connection_string> -x https(collect_network_data=1;trusted_certificates=<certificate_file>) -sl dnet
```

```
mlsrv16 -c <connection_string> -x tls(collect_network_data=1;trusted_certificates=<certificate_file>) -sl dnet
```

-zf mlsrv16 オプション

各同期の始めに Mobile Link サーバがスクリプトの変更をチェックします。

> **警告**
> -zf オプションを使用し Mobile Link サーバを実行すると、Mobile Link サーバのパフォーマンスが低下する場合があるので、可能な限り回避することをおすすめします。

構文

mlsrv16 -c "*connection-string*" -zf

備考

-zf オプションを使用しない場合、Mobile Link サーバはスクリプト変更が行われていないものと見なし、起動後にスクリプト変更をチェックしません。

-zp mlsrv16 オプション

競合を検出するために、タイムスタンプの比較の精度を調整します。

構文

mlsrv16 -c "*connection-string*" -zp

備考

このオプションを指定すると、競合を検出するためにタイムスタンプを比較するときに、Mobile Link サーバがリモートデータベースと統合データベースで表せる最も精度の高いタイムスタンプを使用するようにします。リモートデータベース上の更新されたタイムスタンプによって、次の同期で見せかけの競合が発生する場合があるため、このオプションは、統合データベースのタイムスタンプがリモートデータベースのタイムスタンプよりも精度が高い場合に有用です。このオプションを指定すると、Mobile Link はそのような競合を無視します。精度が不一致で -zp が使用されていない場合は、同期ごとにスキーマが異なるテーブル別の警告がログに書き込まれる

ので、-zp オプションの使用を推奨します。可能であればリモートデータベースのタイムスタンプの精度を調整するようユーザに通知する、同期ごとの警告もさらに追加されます。

-zs mlsrv16 オプション

mlstop 用の Mobile Link サーバの名前を指定します。

構文

mlsrv16 -c "*connection-string***" -zs** *name*

備考

デフォルト名は <default> です。

指定する名前には、ASCII の英数字を使用できますが、その他の文字は使用できません。

-zs オプションを使用して起動した Mobile Link サーバの停止に mlstop を使用するときは、サーバ名を mlstop のコマンドラインで指定する必要があります。たとえば mlstop myMLserver のようになります。Mobile Link サーバがインストールされているコンピュータからしか、シャットダウンは開始できません。

参照

● 「Mobile Link 停止ユーティリティ (mlstop)」692 ページ

-zt mlsrv16 オプション

Mobile Link サーバを実行するのに使用されるプロセッサの最大数を指定します。

構文

mlsrv16 -c "*connection-string***" -zt** *number*

備考

一部の ODBC ドライバでは、このオプションが必須です。また、プロセッサリソースを厳密に制御できます。

このオプションは、Windows および Linux オペレーティングシステムでしか使用できません。デフォルトは、コンピュータに搭載されているプロセッサの数です。

-zu mlsrv16 オプション

authenticate_user スクリプトと authenticate_user_hashed スクリプトが未定義の場合に、ユーザの自動的な追加を制御します。

構文

mlsrv16 -c "*connection-string***" -zu**{ + | - } ...

備考

このオプションを -zu+ として指定すると、認識されなかった Mobile Link ユーザ名が最初の同期時に自動的に ml_user テーブルに追加されます。-zu- を引数に指定するか、まったく指定しない場合は、認識されないユーザ名を同期できません。

このオプションは、開発中にユーザを登録するのに使用すると便利です。配備されたアプリケーションでの使用はおすすめしません。

参照

- 「新しいユーザからの同期」『Mobile Link クライアント管理』
- 「Mobile Link ユーザ」『Mobile Link クライアント管理』
- 「Mobile Link ユーザ認証ユーティリティ (mluser)」694 ページ
- 「authenticate_user 接続イベント」291 ページ

-zus mlsrv16 オプション

アップロードするローがテーブルにないときでも、Mobile Link サーバがテーブルのアップロードスクリプトを呼び出すようにします。

構文

mlsrv16 -c "*connection-string***" -zus** …

備考

デフォルトでは、テーブルのローがアップロードされない場合、Mobile Link サーバは、定義されていてもそのテーブルのアップロードスクリプトを呼び出しません。このオプションはデフォルトの動作を無効にして、ローがアップロードされなくても、Mobile Link サーバがテーブルのアップロードスクリプトを呼び出すようにします。

-zw mlsrv16 オプション

表示する警告メッセージのレベルを制御します。

構文

mlsrv16 -c "*connection-string***" -zw** *levels*

備考

Mobile Link には、5 つのレベルの警告メッセージがあります。

レベル	説明
0	すべての警告メッセージを表示しない
1	サーバレベルと高い ODBC レベル : Mobile Link サーバが起動するときに警告メッセージを表示

レベル	説明
2	同期レベルとユーザレベル：同期が開始するときに警告メッセージを表示
3	スキーマレベル：Mobile Link サーバがクライアントスキーマを処理するときに警告メッセージを表示
4	スクリプトレベルと低い ODBC レベル：Mobile Link サーバがスクリプトをフェッチ、準備、または実行するときに警告メッセージを表示
5	テーブルレベルまたはローレベル：Mobile Link サーバがアップロードまたはダウンロードでテーブル操作を実行するときに警告メッセージを表示

レポートする警告メッセージのレベルを指定する場合は、複数のレベルをカンマで区切るか、2 つのドットで範囲を指定できます。たとえば、**-zw 1..3,5** は **-zw 1,2,3,5** と同じです。

メッセージのレポートはパフォーマンスにほとんど影響しません。レベル数が高いほど、多くのメッセージが生成される傾向があります。

同じコマンドラインで -zw を 2 回以上使用すると、Mobile Link は最後のインスタンスのみを認識します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

デフォルトは **1,2,3,4,5** です。この場合、すべてのレベルの警告メッセージが表示されます。

-zwd mlsrv16 オプション

特定の警告コードを無効にします。

構文

mlsrv16 -c "*connection-string***" -zwd** *code*, ...

備考

特定の警告コードを無効にすると、同じレベルの他のコードがレポートされる場合でも、その警告コードはレポートされません。

警告メッセージコードの完全なリストについては、「Mobile Link サーバの警告メッセージ」『エラーメッセージ』を参照してください。

同じコマンドラインで -zwd を 2 回以上使用すると、Mobile Link は設定を累積します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

-zwe mlsrv16 オプション

特定の警告コードを有効にします。

構文

mlsrv16 -c "*connection-string***" -zwe** *code*, ...

備考

特定の警告コードを有効にすると、-zw で同じレベルの他のコードを無効にしてある場合でも、その警告コードがレポートされます。

警告メッセージコードの完全なリストについては、「Mobile Link サーバの警告メッセージ」『エラーメッセージ』を参照してください。

同じコマンドラインで -zwe を 2 回以上使用すると、Mobile Link は設定を累積します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

同期の方法

Mobile Link 開発のヒント

同期機能をアプリケーションに追加すると、複雑なアプリケーションを作成できます。複雑な機能はほとんどの場合管理可能ですが、常に注意する必要があります。リモートから統合データベースに至る (他の統合データベースアプリケーションを含む) 同期システム全体には、多数のコンポーネントがあり、それぞれに注意が必要です。以下のヒントを参考にしてください。

同期機能をプロトタイプアプリケーションに追加する場合、トラブルの原因となっているコンポーネントを推測するのは困難であるため、同期機能のないプロトタイプから始めてください。プロトタイプが正常に動作するようになったら、同期を有効にします。

簡単な同期の方法から始めてください。簡単なアップロードまたはダウンロードを行う場合、スクリプトは 1 つか 2 つしか必要ありません。スクリプトが正しく動作していれば、タイムスタンプ、プライマリキープール、競合解決、任意のビジネスロジックなどのより高度な方法を導入できます。

Mobile Link とプライマリキー

同期システムでは、プライマリキーは、異なるデータベース (リモートと統合) 内の同じローを識別する唯一の方法であり、競合を検出する唯一の方法です。したがって、Mobile Link アプリケーションは以下の規則に従う必要があります。

- 同期される各テーブルには、プライマリキーが存在する必要がある。
- 同期テーブルのプライマリキーの値は更新しない。
- 同期テーブルのプライマリキーは、同期されるすべてのデータベース間でユニークでなければならない。

参照

- 「ユニークなプライマリキー」117 ページ

タイムスタンプベースのダウンロードの実装

タイムスタンプによる方法は、効率よくダウンロードするために最も便利な一般的な手法です。この方法では、各ユーザが最後に同期を行った時間が追跡され、それ以降に変更されたローだけがダウンロードされます。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

Mobile Link は、各 Mobile Link ユーザが最後にデータをダウンロードした日時を示す TIMESTAMP 値を管理します。この値は、「最終ダウンロード時刻」と呼ばれます。

♦ タスク

1. テーブル用にタイムスタンプベースの同期を実装するには、統合データベースで、ローの最終修正時刻を保持する last_modified カラムを追加します。通常、このカラムは次のように宣言されます。

DBMS	最終変更カラム
Adaptive Server Enterprise	datetime
IBM DB2 LUW	timestamp NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
Microsoft SQL Server	datetime
MySQL	timestamp default CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
Oracle	timestamp
SQL Anywhere	timestamp DEFAULT timestamp

2. download_cursor イベントと download_delete_cursor イベントのスクリプト内で、最初のパラメータを TIMESTAMP カラムの値と比較します。

結果

タイムスタンプベースの同期が実装されます。

例

次の例は、Mobile Link Contact のサンプルから抜粋したものであり、タイムスタンプベースのダウンロードの実装方法を示します。

●テーブル定義

```
CREATE TABLE "DBA"."Customer"(
  "cust_id"  integer NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
  "name"  char(40) NOT NULL,
  "rep_id"  integer NOT NULL,
  "last_modified" timestamp NULL DEFAULT timestamp,
  "active"  bit NOT NULL,
  PRIMARY KEY ("cust_id") )
```

●download_cursor スクリプト

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer KEY JOIN SalesRep
WHERE Customer.last_modified >= {ml s.last_table_download}
   AND SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
```

参照

- 「Contact サンプルの顧客窓口の同期」『Mobile Link クイックスタート』
- 「スクリプトでの最終ダウンロード時刻」 109 ページ
- 「同期論理のソースコード」『Mobile Link クイックスタート』

スクリプトでの最終ダウンロード時刻

最終ダウンロードタイムスタンプは、多くの Mobile Link イベントにパラメータとして指定されます。ダウンロードフェーズの直前に、最後に成功した同期中に統合データベースから取得された値です。現在の Mobile Link のユーザが同期を行ったことがない場合や同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

複数のパブリケーションがあり、それらを異なる時間に同期させている場合には、複数の異なる最終ダウンロードタイムスタンプを持つことができます。このため、最終ダウンロードタイムスタンプには次の 2 つのスクリプトパラメータ名があります。

- **last_table_download**　同期される現在のテーブルの最終ダウンロード時刻です。
- **last_download**　すべてのテーブルが同期されていた最後の時間です。同期されるどのテーブルでも、last_table_download の最も古い値になります。

Mobile Link スクリプトで名前付きパラメータの代わりに疑問符を使用すると、イベントに基づいて正しい値が常に使用されます。SQL スクリプトでの疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。

警告
最終変更情報を保持しているカラムは同期しないでください。リモートデータベースがこのようなカラムを要求する場合は、別のカラム名を使用してください。そうしないと、TIMESTAMP 値が、アップロードされた値で上書きされ、統合データベースでローが最後に変更された時刻が保持されません。

参照

- 「ダウンロードタイムスタンプの生成および使用方法」110 ページ
- 「スクリプトのパラメータ」234 ページ

例

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer KEY JOIN SalesRep
WHERE Customer.last_modified >= {ml s.last_table_download}
   AND SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
```

ダウンロードタイムスタンプの生成および使用方法

Mobile Link は、次のようにタイムスタンプベースのダウンロードのタイムスタンプを生成して使用します。

- アップロードのコミット後、prepare_for_download イベントを呼び出す直前に、Mobile Link サーバは統合データベースから現在の時刻をフェッチして、値を保存します。この TIMESTAMP 値は現在のダウンロードの開始時刻を表します。次回の同期では、この時刻の後に変更されたデータのみをダウンロードします。

> **注意**
> 統合データベースでスナップショットアイソレーションがサポートされている場合、ダウンロードのタイムスタンプは次のうち早い方の時刻になります。
>
> ○現在の時刻
>
> ○最初に開いたトランザクションの開始時刻

- Mobile Link サーバは、ダウンロードの一部としてこの TIMESTAMP 値を送信し、クライアントはそれを保存します。

- クライアントは次回同期するとき、アップロードと一緒に送信する **last_download_timestamp** にこの TIMESTAMP 値を使用します。

- Mobile Link サーバは、クライアントがアップロードしたばかりの last_download_timestamp をダウンロードスクリプトに渡します。すると、スクリプトは、最後の last_download_timestamp 以降のタイムスタンプを持つ変更を選択できるので、新しい変更だけがダウンロードされるようになります。

最終ダウンロード時刻が格納される場所

最終ダウンロード時刻は、リモートデータベースに格納されます。この場所は、ダウンロードが正常に適用されたかどうかがわかっているのはリモートデータベースのみであるという理由で適切な場所です。

SQL Anywhere リモートでは、最終ダウンロード時刻はサブスクリプションごとに格納されます。「SYSSYNC システムビュー」『SQL Anywhere サーバ SQL リファレンス』を参照してください。

Ultra Light リモートでは、最終ダウンロード時刻はパブリケーションごとに格納されます。「syspublication システムテーブル」『Ultra Light データベース管理とリファレンス』を参照してください。

最終ダウンロード時刻の変更

まれな状況として、last_download_timestamp の変更が必要な場合があります。たとえば、リモートデータベースのすべてのデータを誤って削除した場合には、最終ダウンロードタイムスタンプの値をリセットする modify_last_download_timestamp 接続スクリプトを定義して、リモートデータベースを再度ダウンロードできます。generate_next_last_download_timestamp や modify_next_last_download_timestamp という別のイベントもあります。これらを使用すると、現在の同期ではなく、次回の同期のタイムスタンプを設定できます。たとえば、UTC の TIMESTAMP 値を使用して、テーブルの last_modified カラムの UTC 値と比較する必要がある場合などです。次の項を参照してください。

- 「modify_last_download_timestamp 接続イベント」
- 「generate_next_last_download_timestamp イベント」
- 「modify_next_last_download_timestamp 接続イベント」

Ultra Light にも、リモートから最終ダウンロード時刻を変更する機能があります。次の項を参照してください。

- ULResetLastDownloadTime メソッド [Ultra Light Embedded SQL]『Ultra Light C/C++ プログラミング』
- ULConnection.ResetLastDownloadTime メソッド [Ultra Light C++]『Ultra Light C/C++ プログラミング』
- ULConnection.ResetLastDownloadTime メソッド [Ultra Light.NET]『Ultra Light .NET プログラミング』

参照

- 「スクリプトでの最終ダウンロード時刻」109 ページ

夏時間対応

分散データベースシステムでは、データの同期が夏時間への切り替え時に重なると問題が発生します。たとえば、データを消失する可能性があります。これは夏時間から元に戻った結果、あいまいになりかねない 1 時間がある秋にだけ問題になります。

夏時間に対応するには、次の 4 つの方法があります。

- 統合データベースサーバで UTC (協定世界時) を使用する。
- 統合データベースサーバで夏時間を無効にする。
- 時刻の切り替え時の 1 時間はシャットダウンする。
- ダウンロード TIMESTAMP カラムで UTC のタイムスタンプを使用し、generate_next_last_download_timestamp または modify_next_last_download_timestamp スクリプトを使用して次の最終ダウンロードタイムスタンプ用に UTC のタイムスタンプを提供する。

スナップショットを使った同期

スナップショットを使ってテーブルを同期する場合、テーブルのローのうちで関係するローすべてを完全にダウンロードします。すでにダウンロード済みのローもダウンロードされます。この方法が最も簡単ですが、不必要に大量のデータセットが交換されるため、パフォーマンスが低下し、通信費も高くなる可能性があります。

スナップショットを使った同期によって、テーブルのすべてのローをダウンロードできます。また、ローの分割方法と組み合わせて実行することもできます。「リモートデータベース間でローを分割する」113 ページを参照してください。

スナップショットを使った同期をいつ行うか

通常、次の特徴を両方満たすテーブルに対してスナップショットを使用すると最も有効です。

- **ロー数が比較的少ない**　ローの数が少ない場合は、ローを全部ダウンロードしても大きなオーバヘッドにはなりません。
- **頻繁にローが変更される**　テーブルのほとんどのローが頻繁に変更される場合は、前回の同期の後で変更されていないローを明示的に除外してもあまり効果はありません。

テーブルの内容が為替レートのリストになっている場合は、通貨の種類はそれほど多くないので、この方法が適しています。また、ほとんどのレートは頻繁に更新されます。ビジネスの性質によって、価格、利率のリスト、または最新ニュース項目といった内容を含むテーブルが考えられます。

スナップショットベースの同期の実装

スナップショットベースの同期を実装するときは、次の点に注意してください。

- リモートユーザが値を更新しない場合は、アップロードスクリプトを未定義のままにしておきます。
- テーブルのローを削除する場合は、リモートテーブルのローをすべて削除する download_delete_cursor スクリプトを作成するか、少なくともすべてのローがもう必要ないことを定義します。後者の方法を使用する場合は、統合データベースからローを削除しないで、削除のマークを付けてください。ローの値を知らないと、リモートデータベースからローを削除できません。

  「download_delete_cursor スクリプト」265 ページを参照してください。
- download_cursor スクリプトを作成し、リモートテーブルに登録するローをすべて選択します。

スナップショット同期使用時のローの削除

統合データベースからローを削除しないで、削除のマークを付けてください。ローの値を知らないと、リモートデータベースからローを削除できません。download_cursor スクリプトの場合はマークなしのローだけを、download_delete_cursor スクリプトの場合はマーク付きのローだけを選択します。

download_delete_cursor スクリプトは、download_cursor スクリプトより先に実行されます。ダウンロードにローが含まれる場合は、同じプライマリキーを持つローを削除リスト内に含める必要

はありません。ダウンロードしたローをリモートロケーションで取得するときに、同じプライマリキーを持つ既存のローは置き換えられます。

「ローをダウンロードするスクリプト」262 ページを参照してください。

別の削除方法

リモートデータベースからローを削除する場合、download_cursor スクリプトを使わなくても、リモートアプリケーションを使ってローを削除できます。たとえば、同期のすぐ後に、アプリケーションで SQL 文を実行して不要なローを削除できます。

アプリケーションによって削除されたローは、通常は次回の同期で Mobile Link サーバにアップロードされますが、STOP SYNCHRONIZATION DELETE 文を使って、アップロードされないようにできます。次に例を示します。

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM table-name
 WHERE expiry_date < CURRENT TIMESTAMP;
COMMIT;
START SYNCHRONIZATION DELETE;
```

「download_delete_cursor スクリプト」265 ページを参照してください。

スナップショットの例

サンプルアプリケーションの ULProduct テーブルは、スナップショットを使った同期によって管理されます。テーブルに入っているローの数は比較的少ないため、スナップショットを使った同期でのオーバヘッドがわずかです。

1. アップロードスクリプトがありません。これは、リモートデータベースでは製品情報を追加できないという業務意思を反映しています。

2. download_delete_cursor スクリプトがないため、リストから製品を削除しないものと見なします。

3. download_cursor スクリプトによって、現在のすべての製品に関して、prod_id、price、prod_name が選択されます。既存の製品の場合は、リモートテーブル内のその製品の価格が更新されます。新しい製品の場合は、リモートテーブルにローが挿入されます。

   ```
   SELECT prod_id, price, prod_name
   FROM ULProduct
   ```

ローの数が極めて少ないテーブルでのスナップショットを使った同期の別の例については、「Contact サンプルの営業担当者の同期」『Mobile Link クイックスタート』を参照してください。

リモートデータベース間でローを分割する

各 Mobile Link リモートデータベースは、異なるデータのサブセットを統合データベース内に持つことができます。リモートデータベース間でデータが「分割」されるように、自分専用の同期スクリプトを作成できます。

共通部分がないように切断分割にすることも、重複を持たせて分割することもできます。たとえば、従業員ごとに独自の顧客セットを持っていて、かつ顧客を共有していない場合は、「切断」

分割になります。複数のリモートデータベースに存在するように顧客を共有している場合、分割は「重複」を含みます。

分割は、テーブル用のスクリプトである download_cursor と download_delete_cursor で実行されます。これらのスクリプトによって、リモートデータベースにローをダウンロードするように定義します。各スクリプトは、パラメータとして Mobile Link ユーザ名を使用します。スクリプトでこのパラメータを WHERE 句に指定して、ユーザごとに適切なローを取得します。

切断分割

download_cursor スクリプトと download_delete_cursor スクリプトによって、同期で使用するテーブルごとに分割を制御します。このスクリプトでは、最終ダウンロードタイムスタンプと、同期を呼び出すときに指定する Mobile Link ユーザ名の 2 つのパラメータを使用します。

リモートデータベース間でテーブルを分割するには、次のガイドラインに従います。

- テーブル定義でカラムを指定し、そこに統合データベースの同期ユーザ名を持たせます。このカラムをリモートデータベースにダウンロードする必要はありません。

- このカラムがスクリプトのパラメータと一致するように、download_cursor スクリプトと downloade_delete_cursor スクリプトの WHERE 句に条件文を指定します。

  スクリプトパラメータは、スクリプト内で名前付きパラメータによって表されます。次の例では、download_cursor スクリプトによって、テーブル Contact を emp_id で分割します。

```
SELECT id, contact_name
FROM Contact
WHERE last_modified >= {ml s.last_table_download}
AND emp_id = {ml s.username}
```

参照

- 「download_cursor テーブルイベント」329 ページ
- 「download_delete_cursor テーブルイベント」332 ページ
- 「Contact サンプルの顧客の同期」『Mobile Link クイックスタート』
- 「Contact サンプルの顧客窓口の同期」『Mobile Link クイックスタート』

例

CustDB サンプルアプリケーション内のプライマリキープールテーブルを使って、リモートデータベースごとに独自のプライマリキー値のセットを指定します。この方法は、プライマリキーの重複を避けるために使用します。 「プライマリキープール」122 ページを参照してください。

この方法では、プライマリキープールテーブルがリモートデータベース間で切断分割されるようにしてください。

キープールテーブル ULCustomerIDPool にあるプライマリキー値は、各ユーザが顧客を追加するときに使用します。ULCustomerIDPool テーブルには次の 3 つのカラムがあります。

- **pool_cust_id**　ULCustomer テーブルで使用するプライマリキー値。このカラムだけがリモートデータベースにダウンロードされます。

- **pool_emp_id**　このプライマリキーの所有者である従業員。
- **last_modified**　このテーブルは、last_modified カラムに基づいたタイムスタンプを使って管理されます。

  タイムスタンプの同期については、「タイムスタンプベースのダウンロードの実装」108 ページを参照してください。

このテーブルの download_cursor スクリプトを次に示します。

```
SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE last_modified >= {ml s.last_table_download}
 AND pool_emp_id = {ml s.username}
```

重複のある分割

統合データベースの一部のテーブルは、複数のリモートデータベースに属するローを持ちます。各リモートデータベースは統合データベース内にローのサブセットを持ち、さらにそのサブセットは他のリモートデータベースと重複しています。顧客テーブルの場合は、こうしたことがよく起こります。この場合、顧客テーブルと複数のリモートデータベース間で多対多の関係があり、通常、この関係を表すテーブルが存在します。download_cursor イベントと download_delete_cursor イベントのスクリプトでは、関係を表すテーブルにダウンロードされるテーブルをジョインする必要があります。

例

CustDB サンプルアプリケーションでは、この方法を ULOrder テーブルに使用します。ULEmpCust テーブル上での ULCustomer と ULEmployee の関係は、多対多の関係です。

各リモートデータベースは、ULOrder テーブルから、emp_id カラムの値が Mobile Link ユーザ名と一致するローのみを受信します。

CustDB アプリケーションの ULOrder テーブルで、SQL Anywhere バージョンの download_cursor スクリプトを使用した例を次に示します。

```
SELECT o.order_id, o.cust_id, o.prod_id,
   o.emp_id, o.disc, o.quant, o.notes, o.status
FROM ULOrder o , ULEmpCust ec
WHERE o.cust_id = ec.cust_id
 AND ec.emp_id = {ml s.username}
 AND ( o.last_modified >= {ml s.last_table_download}
  OR ec.last_modified >= {ml s.last_table_download})
 AND  ( o.status IS NULL
  OR o.status != 'Approved' )
 AND ( ec.action IS NULL )
```

このスクリプトは非常に複雑です。スクリプトを見ると、リモートデータベースのテーブルを定義するクエリには、統合データベースのテーブルを複数指定できることがわかります。このスクリプトは、次のすべての条件に一致する ULOrder のローをすべてダウンロードします。

- ULOrder の cust_id カラムと ULEmpCust の cust_id カラムが一致する。

●ULEmpCust の emp_id カラムが同期ユーザ名と一致する。

●注文、または従業員と顧客の関係に対する最終更新日がどちらも、同期ユーザ用の最新の同期時間よりも新しい。

●ステータスは "**Approved**" 以外である。

ULEmpCust のアクションカラムを使用して、削除するカラムにマークを付けます。NULL の場合、ローは完全にアクティブと見なされます (削除されません)。

download_delete_cursor スクリプトを次に示します。

```
SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant, o.notes, o.status
  FROM ULOrder o, dba.ULEmpCust ec
  WHERE o.cust_id = ec.cust_id
    AND ( ( o.status = "Approved" AND o.last_modified >= {ml s.last_table_download} )
      OR ( ec.action = "D"  )  )
    AND ec.emp_id = {ml s.username}
```

リモートデータベースから "Approved" のローがすべて削除されます。

分割された外部キーテーブル

前の項の例では、他のテーブルの基準に従ってテーブルを分割する方法を説明しています。「重複のある分割」115 ページを参照してください。

リモートデータベースの一部のテーブルには、切断のサブセットか重複したサブセットがありますが、サブセットを決定するカラムはありません。これらは、通常、別のテーブルを参照する外部キー (または一連の外部キー) がある外部キーテーブルです。参照先のテーブルにはカラムがあり、これによって適切なサブセットが決定されます。

この場合、download_cursor スクリプトと download_delete_cursor スクリプトでは、参照先のテーブルをジョインしたり、WHERE 句を使用してローを適切なサブセットに制限したりする必要があります。

例については、Customer テーブルのダウンロードスクリプト「Contact サンプルの顧客窓口の同期」『Mobile Link クイックスタート』を参照してください。

アップロード専用の同期とダウンロード専用の同期

デフォルトでは、同期は双方向です。つまり、データはアップロードおよびダウンロードされます。しかし、アップロード専用の同期またはダウンロード専用の同期を選択できます。

注意
このトピックでは、Mobile Link 同期システムをデータベースに作成するときにアップロード専用およびダウンロード専用の同期を設定する方法について説明します。アップロード専用またはダウンロード専用は、Sybase Central で同期モデルを作成する場合にも指定できます。

SQL Anywhere リモートデータベース

- **アップロード**　アップロード専用の同期を実行するには、dbmlsync オプション -uo または拡張オプション UploadOnly を使用します。 次の項を参照してください。

  - 「-uo dbmlsync オプション」『Mobile Link クライアント管理』
  - 「UploadOnly (uo) 拡張オプション」『Mobile Link クライアント管理』

- **ダウンロード**　ダウンロード専用の同期を実行するには、dbmlsync オプション -ds または拡張オプション DownloadOnly を使用します。 次の項を参照してください。

  - 「-ds dbmlsync オプション」『Mobile Link クライアント管理』
  - 「DownloadOnly (ds) 拡張オプション」『Mobile Link クライアント管理』

  SQL Anywhere リモートデータベースはダウンロード専用のパブリケーションを使用することもできます。このダウンロード方法はダウンロード専用同期とは異なります。 「ダウンロード専用のパブリケーション」『Mobile Link クライアント管理』を参照してください。

Ultra Light リモートデータベース

- **アップロード**　アップロード専用の同期を実行するには、Upload Only 同期パラメータを使用します。

  「Upload Only 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

- **ダウンロード**　ダウンロード専用の同期を実行するには、Download Only 同期パラメータを使用します。

  「Download Only 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

ユニークなプライマリキー

同期される各テーブルには、プライマリキーが必要です。また、同期された各テーブルのプライマリキーは、同期対象のすべてのデータベース間でユニークでなければなりません。プライマリキーの値は更新しないようにしてください。

多くの場合、テーブルのプライマリキーとして単一のカラムを使用すると便利です。たとえば、顧客にはそれぞれユニークな ID 値を割り当ててください。営業担当者全員がデータベース接続を直接管理できる環境で作業する場合は、ID 番号の割り当ては簡単に実施できます。顧客テーブルに新しい顧客が挿入されると、テーブルに最後に追加された値よりも大きな値を持つ新規のプライマリキーが自動的に追加されます。

接続が切断された環境では、新しいローの挿入時に、プライマリキーにユニークな値を割り当てるのは簡単ではありません。営業担当者が新しい顧客を追加する場合は、顧客テーブルのリモートコピーにも同様な操作を行います。自分以外の営業担当者が、顧客テーブルの自分以外のコピーに対して操作を行っている場合、同じ顧客 ID 値を使わせないようにします。

この項では、同期対象のすべてのデータベース間でユニークなプライマリキーの生成に関する問題を解決する次の方法について説明します。

- 「複合キー」
- 「UUID」
- 「GLOBAL AUTOINCREMENT」
- 「プライマリキープール」

複合キー

Mobile Link リモート ID は、同期システム内のリモートデータベースをユニークに定義します。したがって、ユニークなプライマリキーを簡単に作成するには、値の一部として Mobile Link リモート ID を含む複合プライマリキーを作成します。ユニークな Mobile Link ユーザ名を保持している場合には、リモート ID の代わりにユーザ名を使用できます。

参照

- 「リモート ID」『Mobile Link クライアント管理』

UUID

newid( ) 関数を使用してプライマリキーに対して完全にユニークな値を作成することによって、プライマリキーをユニークにすることができます。作成された UUID は、uuidtostr( ) 関数を使用して文字列に変換できます。また、strtouuid( ) 関数を使用してバイナリに戻すことができます。

UUID は GUID とも呼ばれ、すべてのコンピュータを通じてユニークです。ただし、この値は完全にランダムのため、値が追加された日時や値の順序を判別することはできません。また、UUID の値は他の方法 (グローバルオートインクリメントを含む) で必要な値よりかなり大きく、プライマリキーテーブルと外部キーテーブルの両方でより多くのテーブル領域を必要とします。UUID を使用するテーブルのインデックスも効率性に劣ります。

参照

SQL Anywhere データベース：

- 「NEWID デフォルト」『SQL Anywhere サーバ SQL の使用法』
- 「NEWID 関数 [その他]」『SQL Anywhere サーバ SQL リファレンス』
- 「UNIQUEIDENTIFIER データ型」『SQL Anywhere サーバ SQL リファレンス』

Ultra Light データベース：

- 「Ultra Light でのプライマリキーの一意性」『Ultra Light データベース管理とリファレンス』
- 「NEWID 関数 [その他]」『Ultra Light データベース管理とリファレンス』

例

次の SQL Anywhere の CREATE TABLE 文は、完全にユニークなプライマリキーを作成します。

```
CREATE TABLE customer (
  cust_key UNIQUEIDENTIFIER NOT NULL
        DEFAULT NEWID( ),
```

```
rep_key VARCHAR(5),
PRIMARY KEY(cust_key))
```

GLOBAL AUTOINCREMENT

SQL Anywhere と Ultra Light のデータベースでは、デフォルトのカラム値を GLOBAL AUTOINCREMENT に設定できます。このデフォルト設定は、ユニークな値を管理するカラムのすべてに適用できますが、特にプライマリキーの場合に役立ちます。

GLOBAL AUTOINCREMENT の値は、連続する値範囲でリモートデータベース間に分割されます。値のセットは有限であるため、各範囲のサイズが大きくなるほど、使用可能な範囲は少なくなります。必要に応じて範囲の正しいサイズを設定するには、注意が必要です。範囲は不足する可能性がありますが、不足を検出して新しい範囲を割り当てることができます。「デフォルト値の選択方法」121 ページを参照してください。

GLOBAL AUTOINCREMENT カラムの使用

デフォルトカラム値を、ユニークな値を保持する GLOBAL AUTOINCREMENT に設定できます。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

必要に応じて範囲の正しいサイズを設定するには、注意が必要です。範囲は不足する可能性がありますが、不足を検出して新しい範囲を割り当てることができます。「デフォルト値の選択方法」121 ページを参照してください。

♦ タスク

1. カラムを GLOBAL AUTOINCREMENT カラムとして宣言します。

   DEFAULT GLOBAL AUTOINCREMENT を指定すると、そのカラムの値のドメインが分割されます。各分割には同じ数の値が含まれます。たとえば、データベース内の整数カラムの分割サイズを 1000 に設定した場合、1 つの分割が 1001 から 2000 まで拡大します。また、2 つ目の分割は 2001 から 3000 まで拡大し、以降、同じように拡大していきます。

   「DEFAULT GLOBAL AUTOINCREMENT」120 ページを参照してください。

2. global_database_id 値を設定します。

   SQL Anywhere と Ultra Light のデータベースでは、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。たとえば、データベースに ID 番号 10 を割り当て、分割サイズが 1000 の場合、このデータベースのデフォルト値は 10001 ～ 11000 の範囲から選択されます。このデータベースの別のコピーで、ID 番号 11 が割り当てられたデータベースからは、11001 ～ 12000 の範囲にある同一カラムのデフォルト値が指定されます。

「グローバルデータベース ID」120 ページを参照してください。

結果

カラムは GLOBAL AUTOINCREMENT カラムとして設定されます。

DEFAULT GLOBAL AUTOINCREMENT

Sybase Central でカラムのプロパティを選択するか、CREATE TABLE 文または ALTER TABLE 文に DEFAULT GLOBAL AUTOINCREMENT フレーズを組みこむことで、作業データベースにデフォルト値を設定できます。

オプションで、AUTOINCREMENT キーワードの直後にカッコで分割サイズを指定できます。この分割サイズには任意の正の整数を設定できますが、通常、分割サイズは、サイズの値がすべての分割で不足しないように選択されます。

INT または UNSIGNED INT 型のカラムの場合、デフォルトの分割サイズは $2^{16} = 65536$ です。他の型のカラムの場合、デフォルトの分割サイズは $2^{32} = 4294967296$ です。特にカラムが INT または BIGINT 型以外のときは、これらのデフォルトが適切ではない場合があるため、分割サイズを明示的に指定してください。

たとえば、次の SQL 文では 2 つのカラム (顧客 ID 番号を格納する整数カラムと顧客名を格納する文字列カラム) を持つ簡単なテーブルが作成されます。分割サイズは 5000 に設定されます。このサイズは、各リモートデータベースに挿入される新しいローが少ないアプリケーションデータベースに適しています。

```
CREATE TABLE customer (
 id   INT          DEFAULT GLOBAL AUTOINCREMENT (5000),
 name VARCHAR(128) NOT NULL,
 PRIMARY KEY (id)
)
```

参照

- SQL Anywhere： 「CREATE TABLE 文」『SQL Anywhere サーバ SQL リファレンス』
- Ultra Light： 「CREATE TABLE 文 [Ultra Light]」『Ultra Light データベース管理とリファレンス』

グローバルデータベース ID

アプリケーションを配備するときには、各データベースに対して必ず異なる ID 番号を割り当てます。ID 番号はさまざまな方法で作成して配布できます。テーブルに値を設定し、リモート ID など、ユニークなプロパティに基づいて、各データベースに適切なローをダウンロードするのも 1 つの方法です。

グローバルデータベース ID 番号の設定　SQL Anywhere では、パブリックオプション global_database_id の値を設定して、データベースのグローバル ID を設定します。ID 番号は正の整数にします。

Ultra Light では、global_id オプションを設定して、データベースのグローバル ID を設定します。

参照

- 「global_database_id オプション」『SQL Anywhere サーバ データベース管理』
- 「Ultra Light global_database_id オプション」『Ultra Light データベース管理とリファレンス』

デフォルト値の選択方法

グローバルデータベース ID は、SQL Anywhere ではパブリックオプション global_database_id、Ultra Light では global_id オプションを使用して設定します。

各データベース内のグローバルデータベース ID オプションは、ユニークな正の整数に設定してください。特定のデータベースのデフォルト値の範囲は、$pn + 1$ ～ $p(n + 1)$ です。ここで、p は分割サイズ、n はグローバルデータベース ID の値を表します。たとえば、分割サイズを 1000、グローバルデータベース ID を 3 に設定すると、範囲は 3001 ～ 4000 になります。

SQL Anywhere と Ultra Light では、次の規則を適用してデフォルト値を選択します。

- カラムに現在の分割の値が含まれていない場合、最初のデフォルト値は $pn + 1$ である。ここで、p は分割サイズ、n はグローバルデータベース ID の値を表します。
- カラムに現在の分割の値が含まれていても、そのすべてが $p(n + 1)$ 未満であれば、この範囲内でこれまで使用した最大値より 1 大きい値が次のデフォルト値になる。
- デフォルトのカラム値は、現在の分割以外のカラムの値の影響を受けない。つまり、$pn + 1$ より小さいか $p(n + 1)$ より大きい数には影響されない。Mobile Link 同期を介して別のデータベースからレプリケートされた場合に、このような値が存在する可能性があります。

グローバルデータベース ID がデフォルト値の 2147483647 に設定されると、カラムには NULL 値が挿入されます。NULL 値が許可されていない場合に、ローの挿入を試みるとエラーが発生します。たとえば、テーブルのプライマリキーにカラムが含まれている場合に、この状況が発生します。

グローバルデータベース ID には負の値は設定できないので、正の値が常に選択されます。ID 番号の最大値を制限するのは、カラムのデータ型と分割サイズだけです。

デフォルトの NULL 値は、分割で値が不足したときにも生成されます。この場合には、データベースに新しいユニークなグローバルデータベース ID 値を割り当てて、別の分割からデフォルト値を選択できるようにしてください。カラムで NULL が許可されていない場合、NULL 値を挿入しようとするとエラーが発生します。未使用の値が残り少ないことを検出し、SQL Anywhere データベースのこのような状態を処理するには、GlobalAutoincrement タイプのイベントを作成します。

特定の分割で値が不足する場合は、新しいグローバルデータベース ID をそのデータベースに割り当てることができます。方法が適切なものであれば、新しいデータベース ID 番号を割り当てることができます。未使用のデータベース ID 値のプールを管理する方法も、その 1 つです。このプールは、プライマリキープールと同じ方法で管理されます。 「プライマリキープール」122 ページを参照してください。

分割で値が不足しそうな場合に、自動的にデータベース管理者へ通知する (またはその他のアクションを実行する) ようにイベントハンドラを設定できます。 SQL Anywhere データベースについては、「イベントのトリガ条件」『SQL Anywhere サーバ データベース管理』を参照してください。

参照

- 「グローバルデータベース ID」120 ページ
- SQL Anywhere： 「global_database_id オプション」『SQL Anywhere サーバ データベース管理』
- Ultra Light： 「Ultra Light global_database_id オプション」『Ultra Light データベース管理とリファレンス』

例

SQL Anywhere データベースでは、次の文はデータベース ID 番号を 20 に設定します。

```
SET OPTION PUBLIC.global_database_id = 20
```

特定カラムの分割サイズが 5000 の場合、このデータベースのデフォルト値は 100001 ～ 105000 の範囲から選択されます。

プライマリキープール

ユニークなプライマリキーの問題を解決する効果的な方法の 1 つは、データベースの各ユーザに、必要に応じて使用できるプライマリキー値のプールを割り当てることです。たとえば、営業担当者ごとに 100 個の新しい ID 値を割り当てます。各営業担当者は、自分のプール内の値を、新しい顧客に自由に割り当てることができます。

プライマリキープールの実装方法の概要を次に示します。

1. 統合データベースと各リモートデータベースに新しいテーブルを追加して、新しいプライマリキープールを格納します。統合データベースのユニークな値を格納するカラムとは別に、これらのテーブルにはユーザ名を格納するカラムが必要です。このユーザ名のカラムによって、値を割り当てる権限を持つ者を識別できます。

2. 統合データベースで、ストアドプロシージャを作成し、十分な数の新しい ID 値が各ユーザに確実に割り当てられるようにします。新しいエントリを多数挿入する、または同期をあまり行わないリモートユーザには、特に多く、新しい値を割り当てます。

3. download_cursor スクリプトを作成して、各ユーザに割り当てられた新しい値を選択し、それをリモートデータベースにダウンロードします。「タイムスタンプベースのダウンロードの実装」108 ページを参照してください。

4. リモートデータベースを使用するアプリケーションを変更し、ユーザが新しいローを挿入するときに、プールに入っている値をアプリケーションが使用するようにします。アプリケーションは、その値をプールから削除して、値の再使用を防ぎます。

5. upload_delete スクリプトを作成して、削除したキーをアップロードします。ユーザがリモートデータベースの自分専用の値プールから削除した値と対応するローが、Mobile Link サーバによって、統合データベースの値のプールから削除されます。

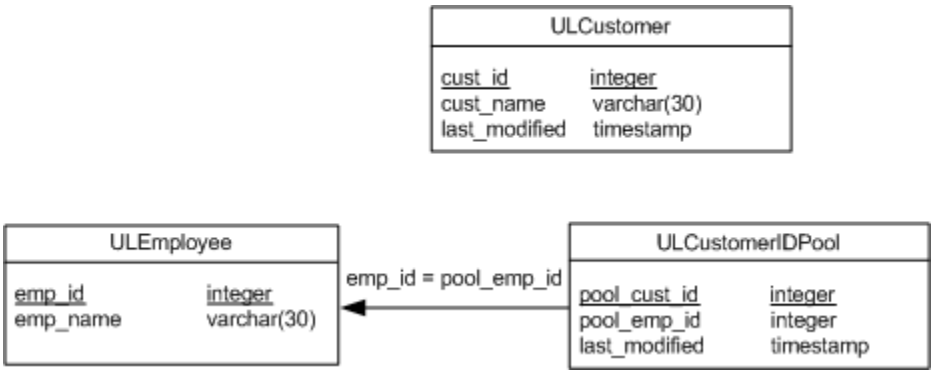
6. end_upload スクリプトを作成し、値のプールを管理するストアドプロシージャを呼び出します。これで、ユーザのプールに対してさらに多くの値が追加され、削除済みの値がアップロード中に置き換わります。

例

リモートユーザは、CustDB サンプルアプリケーションを使って顧客を追加できます。新しいローにはそれぞれユニークなプライマリキー値があることが必要です。ただし、データエントリ中は、まだ各リモートデータベースは切断された状態です。

ULCustomerIDPool にはプライマリキー値のリストがあり、この値を各リモートデータベースで使用できます。また、値を使い切ってしまうと、ULCustomerIDPool_maintain ストアドプロシージャによってプール内の値が完全に補充されます。管理プロシージャは、テーブルレベルの end_upload スクリプトによって呼び出されます。各リモートデータベースのプールは、download_cursor スクリプトと upload_delete スクリプトによって管理されます。

1. 統合データベースの ULCustomerIDPool テーブルには、新しい顧客 ID 番号のプールが格納されます。ULCustomer テーブルとは直接のリンク関係はありません。



2. ULCustomerIDPool_maintain プロシージャによって、統合データベースの ULCustomerIDPool テーブルが更新されます。SQL Anywhere 統合データベース用のサンプルコードを次に示します。

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
   DECLARE pool_count INTEGER;

   -- Determine how may ids to add to the pool
   SELECT COUNT(*) INTO pool_count
   FROM ULCustomerIDPool
   WHERE pool_emp_id = syncuser_id;

   -- Top up the pool with new ids
   WHILE pool_count < 20 LOOP
     INSERT INTO ULCustomerIDPool ( pool_emp_id )
     VALUES ( syncuser_id );
     SET pool_count = pool_count + 1;
   END LOOP;
END
```

このプロシージャは、現在のユーザに現在割り当てられている番号をカウントします。また、新しいローを挿入して、このユーザが十分な数の顧客 ID 番号を使用できるようにします。

このプロシージャは、ULCustomerIDPool テーブル用の end_upload テーブルスクリプトによって、アップロードの最後に呼び出されます。スクリプトを次に示します。

```
CALL ULCustomerIDPool_maintain( {ml s.username} )
```

3. ULCustomerIDPool テーブル用の download_cursor スクリプトは、リモートデータベースに新しい番号をダウンロードします。

```
SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE pool_emp_id = {ml s.username}
AND last_modified >= {ml s.last_table_download}
```

4. 新しい顧客を挿入するには、リモートデータベースを使用しているアプリケーションで、プール中の未使用の ID 番号を選択して、その番号をプールから削除してから、この ID 番号による新しい顧客情報を挿入します。次に示す Ultra Light アプリケーション用の Embedded SQL 関数は、プールから新しい顧客番号を取り出します。

```
bool CDemoDB::GetNextCustomerID( void )
/************************************/
{
   short ind;

   EXEC SQL SELECT min( pool_cust_id )
   INTO :m_CustID:ind FROM ULCustomerIDPool;
   if( ind < 0 ) {
      return false;
   }
   EXEC SQL DELETE FROM ULCustomerIDPool
   WHERE pool_cust_id = :m_CustID;
   return true;
}
```

競合の解決

警告
同期テーブルのプライマリキーは更新しないでください。プライマリキーは、異なるデータベース (リモートと統合) 内の同じローを識別する唯一の方法であり、競合を検出する唯一の方法なので、プライマリキーを更新すると、プライマリキーの目的が無効になります。

競合は、統合データベースにローをアップロードしているときに発生する可能性があり、エラーとは異なります。競合が起こる可能性がある場合は、適切な値を計算するプロセスを定義するか、最低でも競合のログを取ってください。優れたアプリケーションを設計するには、競合の解決は不可欠です。

デフォルトでは次のように処理されます。

- ローを挿入しようとしたときに、そのローが挿入済みであることが検出されると、エラーが生成されます。
- ローを削除しようとしたときに、そのローが削除済みであることが検出されると、2 回目の削除の試行は無視されます。

異なる動作が必要な場合は、この項で説明するアップロードイベントを 1 つ以上定義して動作を実装できます。

同期のダウンロード処理中は、リモートデータベースでは競合は発生しません。ダウンロードしたローに新しいプライマリキーが含まれている場合は、その値は新しいローに挿入されます。新しいプライマリキーが既存のローのプライマリキーと一致する場合は、そのローの値が更新されます。

例

User1 が最初に 10 個の在庫を売り出し、そのうち 3 個を販売して、Remote1 にある在庫の値を 7 個に更新します。User2 は 4 個販売し、Remote2 にある在庫を 6 に更新します。Remote1 が同期を実行すると、統合データベースは 7 に更新されます。Remote2 が同期を実行すると、在庫の値が 10 ではなくなっているため、競合が検出されます。この競合をプログラムで解決するには、次のような 3 つのロー値が必要となります。

1. 統合データベースにある現在の値。
2. Remote2 がアップロードした新しいローの値。
3. Remote2 が最後の同期中に取得した古いローの値。

この場合、ビジネス論理は新しい在庫数を計算し、競合を解決するために次の方法を使用できます。

```
current consolidated - (old remote - new remote)
-> 7 - (10-6) = 3
```

競合の処理方法に関するその他の例については、次の項を参照してください。

- 「Contact サンプルの製品の同期」『Mobile Link クイックスタート』

競合検出

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、更新された新しい値 (更新後イメージ) だけでなく、最後のダウンロード、またはこのローの最初のアップロード以前に存在していたローの値から取得された古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

Mobile Link サーバは、upload_fetch スクリプトまたは upload_fetch_column_conflict スクリプトが適用された場合にのみ競合を検出します。upload_fetch を使用すると、競合する更新は競合として通知されます。upload_fetch_column_conflict を使用すると、同じカラムに対する競合する更新のみ通知されます。

upload_update 用のストアドプロシージャを使用して、任意での競合の検出と解決を設定できます。競合の検出と解決はこのスクリプトによって完全に制御されるので、Mobile Link での競合のトリガはありません。

upload_fetch または upload_fetch_column_conflict スクリプトによる競合の検出

テーブルに対して upload_fetch または upload_fetch_column_conflict スクリプトを定義すると、Mobile Link サーバは、アップロードされた更新の更新前イメージを、スクリプトから返される、同じプライマリキー値を持つローの値と比較します。更新前イメージの値が統合データベースの現在の値と一致しない場合、Mobile Link サーバは競合を検出します。サーバは upload_old_row_insert スクリプトと upload_new_row_insert スクリプトを呼び出し、その後、競合を検出した場合に resolve_conflict スクリプトを呼び出します。

注意
競合中に upload_old_row_insert スクリプトと upload_new_row_insert スクリプトが定義されていない場合はエラーが発生します。同期テーブルにこれらのスクリプトが必要ない場合は、--{ml_ignore} 文を使用してこれらのスクリプトを無視するものとして定義します。

upload_fetch スクリプトと upload_fetch_column_conflict スクリプトの違いは、Mobile Link サーバが競合の検出に使用する基準にあります。upload_fetch スクリプトを使用した場合、フェッチされたローと更新前イメージのローの相違は競合として処理されます。upload_fetch_column_conflict スクリプトを使用した場合は、フェッチされたローと更新前イメージのローの間で、リモートデータベースによって更新されたカラムのみが比較されます。つまり、upload_fetch はローベースの競合検出を実行し、upload_fetch_column_conflict はカラムベースの競合検出を実行します。

upload_fetch スクリプトは、更新対象となるローに対応する統合データベースのテーブルから、データの単一のローを選択します。このスクリプトを使用するには、次の 2 つの方法があります。1 つは、アップロードされた更新前イメージと同じプライマリキーと同じカラム値を持つローを選択する方法です。ローが返されない場合、Mobile Link サーバは競合を検出します。スクリプトのこの使用方法では、構文は次のようになります (ここで、pk1, pk2, ... はプライマリキーカラムで、col1, col2, ... は非プライマリキーカラムです)。

SELECT *pk1*, *pk2*, ...*col1*, *col2*, ...
FROM *table-name*
WHERE *pk1* **= {ml r.***pk1***} AND** *pk2* **= {ml r.***pk2***}** ...
AND *col1* **= {ml o.***col1***} AND** *col2* **= {ml o.***col2***}** ...

注意
この競合検出方法は、BLOB や CLOB などの大きなバイナリカラムを含む同期テーブルでは使用できません。

もう 1 つは、同じプライマリキーを持つローを選択し、Mobile Link サーバがフェッチされたローをアップロードされた更新前イメージと比較できるようにする方法です。カラムが異なる場合、Mobile Link サーバは競合を検出します。この方法は、すべての同期可能なカラムタイプで使用できます。

SELECT *pk1*, *pk2*, ...*col1*, *col2*, ...
FROM *table-name*
WHERE *pk1* **= {ml r.***pk1***} AND** *pk2* **= {ml r.***pk2***}** ...

「upload_fetch テーブルイベント」435 ページを参照してください。

upload_fetch_column_conflict イベントは upload_fetch と似ていますが、upload_fetch_column_conflict では、最後の同期以降にリモートデータベースと統合データベースで同じカラムが更新された場合にのみ Mobile Link サーバがローの競合を検出するという点が異なります。異なるユーザは、同じカラムを更新しないかぎり、同じローを更新することができ、競合は発生しません。upload_fetch_column_conflict イベントは、BLOB がない同期テーブルにのみ適用できます。

upload_fetch_column_conflict スクリプトを使用し、競合が検出されない場合、upload_update スクリプトに渡されるロー値は、upload_fetch_column_conflict スクリプトによってリモートデータベースのアップロードまたは現在の統合値から取得されます。リモートデータベースで更新されたカラムにはリモートデータベースの値が使用されます。それ以外の場合は現在の統合値が使用されます。つまり、リモートデータベースで更新されたカラムのみが統合データベースで更新されます。

「upload_fetch_column_conflict テーブルイベント」437 ページを参照してください。

リモートデータベースのテーブルごとに、upload_fetch または upload_fetch_column_conflict スクリプトを 1 つのみ指定できます。

統合データベースのローのロック

upload_fetch スクリプトによって競合が検出されてから、競合解決が完了するまでに、統合データベースのローが変更される可能性があります。データが正しくなくなる可能性があるこの問題を回避するには、ローロックを使用して upload_fetch スクリプトまたは upload_fetch_column_conflict スクリプトを実装できます。

SQL Anywhere 統合データベースでは、UPDLOCK キーワードと HOLDLOCK キーワードのどちらも使用できますが、同時実行性については UPDLOCK の方が適しています。次に例を示します。

```
SELECT column-names from table-name WITH (UPDLOCK)
  WHERE where-clause
```

Microsoft SQL Server の場合は、HOLDLOCK を使用します。次に例を示します。

```
SELECT column-names FROM table-name WITH (HOLDLOCK)
  WHERE where-clause
```

Adaptive Server Enterprise の場合は、HOLDLOCK を使用します。次に例を示します。

```
SELECT column-names FROM table-name
  HOLDLOCK
  WHERE where-clause
```

例

upload_fetch スクリプトを定義します。Mobile Link サーバはこのスクリプトを使用して、統合データベースの現在のローを取り出し、更新されたローの更新前イメージとこのローを比較します。2 つのローに同じ値が含まれている場合、競合はありません。2 つのローが異なる場合には、競合が検出され、Mobile Link は upload_old_row_insert と upload_new_row_insert スクリプトを呼び出し、その後に resolve_conflict スクリプトを呼び出します。

「resolve_conflict スクリプトによる競合の解決」128 ページを参照してください。

upload_update スクリプトによる競合の検出

upload_fetch、upload_fetch_column_conflict、upload_old_row_insert、upload_new_row_insert、resolve_conflict のスクリプトを定義しません。代わりに、競合の検出と解決を処理するストアドプロシージャを作成して、upload_update スクリプトで呼び出します。

「upload_update スクリプトによる競合の解決」130 ページを参照してください。

競合解決

競合を解決するために、いくつかのオプションがあります。

- 競合が発生した場合に、テンポラリテーブルまたは永久テーブルと resolve_conflict スクリプトを使用して解決する。

  「resolve_conflict スクリプトによる競合の解決」128 ページを参照してください。

- 競合が発生した場合に、upload_update スクリプトを使用して解決する。

  「upload_update スクリプトによる競合の解決」130 ページを参照してください。

- テーブル用の end_upload スクリプトを使用して、すべての競合を一度に解決する。

  「end_upload テーブルイベント」361 ページを参照してください。

resolve_conflict スクリプトによる競合の解決

Mobile Link サーバが upload_fetch スクリプトを使用して競合を検出すると、次のイベントが発生します。

- Mobile Link サーバは、リモートデータベースからアップロードされた古いロー値を upload_old_row_insert スクリプトの定義に従って挿入します。通常、古い値はテンポラリテーブルに挿入されます。

  「upload_old_row_insert テーブルイベント」443 ページを参照してください。

- Mobile Link サーバは、リモートデータベースからアップロードされた新しいロー値を upload_new_row_insert スクリプトの定義に従って挿入します。通常、新しい値はテンポラリテーブルに挿入されます。

  「upload_new_row_insert テーブルイベント」441 ページを参照してください。

- Mobile Link サーバは、resolve_conflict スクリプトを実行します。このスクリプトでは、ストアドプロシージャを呼び出したり、実行手順の順序を定義したりすることによって、新しいロー値と古いロー値を使用して競合を解決できます。

参照

- 「resolve_conflict テーブルイベント」417 ページ

例

次の例では、6 つのイベントに対してスクリプトを作成し、次にストアドプロシージャを作成します。

●begin_synchronization スクリプトでは、contact_new と contact_old という 2 つのテンポラリテーブルを作成します。(begin_connection スクリプトでこれを行うこともできます)。

●upload_fetch スクリプトは、競合を検出します。

●競合がある場合、upload_old_row_insert スクリプトと upload_new_row_insert スクリプトは、リモートデータベースからアップロードした新しいデータと古いデータを使用して、2 つのテンポラリテーブルを設定します。

●resolve_conflict スクリプトは、MLResolveContactConflict ストアドプロシージャを呼び出して、競合を解決します。

イベント	スクリプト
begin_synchronization	CREATE TABLE #contact_new(id INTEGER, location CHAR(36), contact_date DATE); CREATE TABLE #contact_old(id INTEGER, location CHAR(36), contact_date DATE)
upload_fetch	SELECT id, location, contact_date FROM contact WHERE id = {ml r.id}
upload_old_row_insert	INSERT INTO #contact_new(id, location, contact_date) VALUES ({ml r.id}, {ml r.location}, {ml r.contact_date})
upload_new_row_insert	INSERT INTO #contact_old(id, location, contact_date) VALUES ({ml r.id}, {ml r.location}, {ml r.contact_date})
resolve_conflict	CALL MLResolveContactConflict( )
end_synchronization	DROP TABLE #contact_new; DROP TABLE #contact_old

MLResolveContactConflict ストアドプロシージャは次のとおりです。

```
CREATE PROCEDURE MLResolveContactConflict( )
BEGIN
  --update the consolidated database only if the new contact date
  --is later than the existing contact date
  UPDATE contact c
    SET c.contact_date = cn.contact_date
      FROM #contact_new cn
      WHERE c.id = cn.id
        AND cn.contact_date > c.contact_date;
  --cleanup
  DELETE FROM #contact_new;
  DELETE FROM #contact_old;
END
```

upload_update スクリプトによる競合の解決

競合を解決するのに resolve_conflict スクリプトを使用する代わりに、upload_update スクリプトでストアドプロシージャを呼び出すこともできます。この方法では、プログラムで競合の検出と解決の両方を行う必要があります。

ストアドプロシージャは、新しい (更新後イメージの) 値と古い (更新前イメージの) 値の両方を含む、すべてのカラムを受け入れる必要があります。

以下は、upload_update スクリプトの例です。

```
{CALL UpdateProduct(
  {ml o.id}, {ml o.name}, {ml o.desc}, {ml r.name}, {ml r.desc}
)
}
```

以下は、UpdateProduct ストアドプロシージャの例です。

```
CREATE PROCEDURE UpdateProduct(
 @id INTEGER,
 @preName VARCHAR(20),
 @preDesc VARCHAR(200),
 @postName VARCHAR(20),
 @postDesc VARCHAR(200) )
BEGIN
   UPDATE product
    SET name = @postName, description = @postDesc
    WHERE id = @id
      AND name = @preName
      AND description = @preDesc
   IF @@rowcount=0 THEN
      // A conflict occurred: handle resolution here.
   END IF
END
```

この方法は、resolve_conflict スクリプトによる競合の解決よりも管理が簡単です。それは、管理するスクリプトが 1 つだけで、すべての論理が 1 つのストアドプロシージャに含まれているからです。ただし、テーブルカラムが NULL 入力可の場合、または BLOB や CLOB が含まれている場合には、ストアドプロシージャのコードが複雑になる可能性があります。また、Mobile Link 統合データベースとしてサポートされている RDBMS の一部には、ストアドプロシージャに渡すことができる値のサイズに制限があります。

参照

- 「upload_update スクリプトによる競合の検出」
- 「upload_update テーブルイベント」
- 「resolve_conflict スクリプトによる競合の解決」
- 「COALESCE 関数 [その他]」『SQL Anywhere サーバ SQL リファレンス』
- 「グローバル変数」『SQL Anywhere サーバ SQL リファレンス』の @@rowcount

例

次のストアドプロシージャ sp_update_my_customer には、競合を検出し解決するための論理が定義されています。このストアドプロシージャは古いカラム値と新しいカラム値を受け入れます。この例は SQL Anywhere の機能を使用します。スクリプトは次のように実装できます。

```
{CALL sp_update_my_customer(
  {ml o.cust_1st_pk},
```

```
   {ml o.cust_2nd_pk},
   {ml o.first_name},
   {ml o.last_name},
   {ml o.nullable_col},
   {ml o.last_modified},
   {ml r.first_name},
   {ml r.last_name},
   {ml r.nullable_col},
   {ml r.last_modified}
)}
CREATE PROCEDURE sp_update_my_customer(
     @cust_1st_pk        INTEGER,
     @cust_2nd_pk        INTEGER,
     @old_first_name     VARCHAR(100),
     @old_last_name      VARCHAR(100),
     @old_nullable_col   VARCHAR(20),
     @old_last_modified  DATETIME,
     @new_first_name     VARCHAR(100),
     @new_last_name      VARCHAR(100),
     @new_nullable_col   VARCHAR(20),
     @new_last_modified  DATETIME
)
BEGIN
DECLARE @current_last_modified DATETIME;
// Detect a conflict by checking the number of rows that are
// affected by the following update. The WHERE clause compares
// old values uploaded from the remote database to current values in
// the consolidated database. If the values match, there is
// no conflict. The COALESCE function returns the first non-
// NULL expression from a list, and is used in this case to
// compare values for a nullable column.


UPDATE my_customer
SET first_name                = @new_first_name,
 last_name                    = @new_last_name,
 nullable_col                 = @new_nullable_col,
 last_modified                = @new_last_modified

WHERE cust_1st_pk             = @cust_1st_pk
 AND cust_2nd_pk              = @cust_2nd_pk
 AND first_name               = @old_first_name
 AND last_name                = @old_last_name
 AND COALESCE(nullable_col, '') = COALESCE(@old_nullable_col, '')
 AND last_modified            = @old_last_modified;
...
// Use the @@rowcount global variable to determine
// the number of rows affected by the update. If @@rowcount=0,
// a conflict has occurred. In this example, the database with
// the most recent update wins the conflict. If the consolidated
// database wins the conflict, it retains its current values
// and no action is taken.

IF( @@rowcount = 0 ) THEN
// A conflict has been detected. To resolve it, use business
// logic to determine which values to use, and update the
// consolidated database with the final values.

  SELECT last_modified INTO @current_last_modified
     FROM my_customer WITH( HOLDLOCK )
     WHERE cust_1st_pk=@cust_1st_pk
       AND cust_2nd_pk=@cust_2nd_pk;

     IF( @new_last_modified > @current_last_modified ) THEN
```

```
    // The remote database has won the conflict: use the values it
    // uploaded.

      UPDATE my_customer
      SET first_name    = @new_first_name,
        last_name     = @new_last_name,
        nullable_col  = @new_nullable_col,
        last_modified = @new_last_modified
      WHERE cust_1st_pk  = @cust_1st_pk
        AND cust_2nd_pk   = @cust_2nd_pk;

    END IF;
END IF;
END;
```

削除

ローを統合データベースから削除する場合、download_delete_cursor で明示的にローを選択して、そのローが格納されているすべてのリモートデータベースから削除できるように、ローのレコードが必要です。これを行う 2 つの方法として、論理削除を使用する方法と、シャドウテーブルを使用する方法があります。

- **論理削除**　この方法では、ローは削除されません。不要になったデータについては、ステータスカラムで非アクティブのマークが付けられます。download_cursor と download_delete_cursor の WHERE 句やほとんどのアプリケーションクエリでは、ローのステータスを参照できます。

  この方法は、CustDB サンプルアプリケーションで使用されており、ULEmpCust.action カラムには削除を示す "D" が入っています。スクリプトでは、この値を使用してリモートデータベースからレコードを削除し、さらに、同期処理の最後に統合データベースからもレコードを削除します。CustDB ではこの方法を ULOrder テーブルに使用し、Contact サンプルではこの方法を Customer、Contact、Product テーブルに使用しています。

  Mobile Link 同期モデルでの論理削除のサポートは、論理削除カラムが統合データベースのみにあり、リモートにはないことを前提としています。統合スキーマを新しいリモートスキーマにコピーする場合、モデルの同期設定の論理削除カラムに対応するすべてのカラムを除外します。新しいモデルでは、デフォルトのカラム名は *deleted* です。

  論理削除カラム名をリモートスキーマに追加するには、次の手順に従います。

  1. **同期モデル作成ウィザード**の **[削除のダウンロード]** ページで、**[論理削除を使用する]** をクリックします。

  2. 論理削除カラム名を変更し、統合データベースのどのカラム名とも一致しないようにします。

  3. ウィザードが完了したら、リモートスキーマを更新し、デフォルトのテーブル選択を保持します。論理削除カラム名がスキーマ変更リストに表示され、リモートスキーマに追加されます。

> **注意**
> リモートの論理削除カラムのカラムマッピングを統合データベースの論理削除カラムに設定する必要があります。

- **シャドウテーブル**　この方法では、シャドウテーブルを作成し、削除したローのプライマリキー値をそこに格納します。ローが削除されると、1 つのトリガによってシャドウテーブルが設定されます。download_delete_cursor スクリプトは、シャドウテーブルを使用して、リモートデータベースからローを削除します。シャドウテーブルには、実際のテーブルのプライマリキーカラムだけが必要です。

参照

- 「download_delete_cursor スクリプト」265 ページ

削除同期の一時停止

STOP SYNCHRONIZATION DELETE 文を使用して同期サブスクリプションのあるパブリケーションに属するテーブルまたはカラムへの変更の自動ロギングを一時的に停止します。通常、これらの変更は次の同期時に統合データベースにアップロードされます。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

STOP SYNCHRONIZATION DELETE 文を実行すると、その接続に対してそれ以降に実施された削除操作は同期されません。この効果は、START SYNCHRONIZATION DELETE 文が実行されるまで継続します。この効果はネストしません。つまり、最初の STOP SYNCHRONIZATION DELETE 文の後に同じ文を実行してもさらなる効果はないということです。

この機能は、特別な修正のために使用できますが、自動同期機能の一部が無効化されるので、注意して使用してください。この方法は、download_delete_cursor スクリプトを使用して必要なローを削除する処理に代わる、実用的な代替手段です。

♦ タスク

1. 次の文を発行して削除の自動ロギングを停止します。

   ```
   STOP SYNCHRONIZATION DELETE
   ```

2. 必要に応じて、DELETE 文を使用して同期対象のテーブルからローを削除します。これまでの変更内容をコミットします。

3. 次の文を使用して削除のログを再開します。

   ```
   START SYNCHRONIZATION DELETE
   ```

結果

削除されたローは Mobile Link サーバに送られないため、統合データベースからは削除されません。

参照

- SQL Anywhere クライアント：「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ SQL リファレンス』
- Ultra Light クライアント：「STOP SYNCHRONIZATION DELETE 文 [Ultra Light]」『Ultra Light データベース管理とリファレンス』
- SQL Anywhere クライアント:「START SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ SQL リファレンス』
- Ultra Light クライアント：「START SYNCHRONIZATION DELETE 文 [Ultra Light]」『Ultra Light データベース管理とリファレンス』

ダウンロードの失敗

非ブロッキングダウンロード確認の使用

非ブロッキングダウンロード確認トランザクションで、ダウンロード内容の書き換え情報を保持します。この情報は、リモートデータベースによってダウンロードが正常に適用された後に呼び出される publication_nonblocking_download_ack スクリプトまたは nonblocking_download_ack スクリプトで更新してください。

エラーが発生するか SendDownloadAck が OFF の場合、これらの非ブロッキングダウンロード確認スクリプトは呼び出されず、統合データベースのダウンロードタイムスタンプは更新されません。同期スクリプトをテストするときに人為的にダウンロードを失敗させて、失敗したダウンロードをスクリプトで処理できることを確認してください。

ブロッキングダウンロード確認の使用

ブロッキングダウンロード確認はサポートされなくなりました。すべてのダウンロード確認は非ブロッキングとして処理されます。

「SendDownloadAck (sa) 拡張オプション」『Mobile Link クライアント管理』と「Send Download Acknowledgement 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

ダウンロードの失敗の再開

ダウンロードの失敗は、ダウンロード中の通信エラーまたはリモートユーザによるダウンロードのキャンセルによって発生します。Mobile Link サーバは、再起動可能なダウンロードで使用できるように、クライアントに受信されなかったダウンロードデータを保持します。サーバがダウンロードデータを破棄するのは、次のいずれかの場合です。

●ユーザがダウンロードを正常に完了した場合。

●再開しようとせずに、ユーザが新しい同期要求で復帰した場合。

●新しいダウンロードのキャッシュが必要な場合。正常にダウンロードできなかった最も古いデータが、最初にクリアされます。

Mobile Link は、ダウンロードの失敗からのリカバリを支援する機能を備えています。この機能を使用すると、ダウンロード全体の再送を防ぐこともできます。この機能は、SQL Anywhere と Ultra Light リモートデータベースでそれぞれ別に実装されています。 「-ds mlsrv16 オプション」58 ページを参照してください。

注意
再開可能なダウンロードを成功させるには、リモートデータベースを同じ Mobile Link サーバに接続する必要があります。リモートデータベースを別の Mobile Link サーバに接続した場合、または同じ Mobile Link サーバであっても失敗したダウンロードが除去されている場合は、ダウンロードを再開しようとしても失敗します。

失敗したダウンロードの再開が必要になる状況

ネットワークの品質が低ければ低いほど、および、ダウンロードサイズが大きければ大きいほど、再開可能なダウンロードの必要性が高くなります。小規模な同期のみを実行する場合、または同期を LAN や WLAN で実行する場合は、ダウンロードの再開が必要になる可能性は低くなります。

SQL Anywhere リモートデータベース

ダウンロード中に同期が失敗すると、ダウンロードされたデータはリモートデータベースには適用されません。ただし、正常に送信されたダウンロードの部分は、リモートデバイスのテンポラリファイルに格納されます。dbmlsync は、このファイルを使用して長時間のデータ再送を防ぎ、ダウンロードの失敗からリカバリします。

この機能を実装するには 3 つの方法があります。どの方法でも、dbmlsync はアボートされ、アップロードする新しいデータが存在する場合は再開されたダウンロードは失敗します。

- **-dc** ダウンロードが失敗した後、次回 dbmlsync を起動するときに、-dc を使用してダウンロードを再開します。失敗したダウンロードの一部が送信されている場合、Mobile Link サーバは、ダウンロードの残りだけを送信します。

  詳細については、「-dc dbmlsync オプション」『Mobile Link クライアント管理』を参照してください。

- **ContinueDownload (cd) 拡張オプション** dbmlsync コマンドラインで使用した場合、cd 拡張オプションは -dc オプションと同じように動作します。このオプションは、データベース内に格納したり、sp_hook_dbmlsync_set_extended_options を使用して 1 つの同期内に設定することもできます。

  「ContinueDownload (cd) 拡張オプション」『Mobile Link クライアント管理』と「sp_hook_dbmlsync_set_extended_options」『Mobile Link クライアント管理』を参照してください。

- **sp_hook_dbmlsync_end hook** restart パラメータを使用して、ダウンロードを再開できます。restartable download パラメータが true に設定されている場合は、ダウンロードを再開できます。ダウンロードファイルが存在し、一定のサイズの場合は、再起動可能なダウンロー

ドのサイズを使用することによって、フック内にロジックを作成してダウンロードを再開することもできます。

「sp_hook_dbmlsync_end」『Mobile Link クライアント管理』を参照してください。

Ultra Light リモートデータベース

次に示すように、ダウンロードが失敗した後の Ultra Light アプリケーションの動作を制御できます。

- 同期時に Keep Partial Download 同期パラメータを true に設定している場合、ダウンロードが完了する前に失敗すると、Ultra Light はダウンロードされた変更の部分を適用します。また Ultra Light は、Partial Download Retained パラメータを true に設定します。

  この時点では、Ultra Light データベースは一貫性のない状態になります。アプリケーションの仕様に応じて、同期を正常に完了するか、データの変更を許可する前にロールバックしてください。 「Keep Partial Download 同期パラメータ」『Ultra Light データベース管理とリファレンス』と「Partial Download Retained 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

- ダウンロードを再開するには、Resume Partial Download 同期パラメータを true に設定し、再度同期を実行します。 「Resume Partial Download 同期パラメータ」『Ultra Light データベース管理とリファレンス』を参照してください。

  再開された同期はアップロードを実行せず、失敗したダウンロードによってダウンロードされた変更のみをダウンロードします。つまり、失敗したダウンロードは完了しますが、前回の実行以降に行われた変更は同期しません。これらの変更を取得するには、一度失敗したダウンロードが完了してから、再度同期を実行する必要があります。または、Rollback Partial Download を呼び出し、Resume Partial Download を false に設定した状態で同期します。

  ダウンロードを再開すると、失敗した同期から多数の同期パラメータがもう一度自動的に使用されます。たとえば、パブリケーションパラメータは無視されます。同期は、最初のダウンロードで要求されたパブリケーションをダウンロードします。唯一設定が必要なパラメータは、Resume Partial Download パラメータ (true に設定) と User Name パラメータです。また、以下のパラメータ (設定されている場合) の設定も有効です。

  - Keep Partial Download
  - DisableConcurrency
  - Observer
  - User Data

- 失敗したダウンロードからの変更を、同期を再開せずにロールバックするには、変更をロールバックする適切なメソッドまたは関数を呼び出します。この関数は、embedded SQL の ULRollbackPartialDownload 関数です。Ultra Light コンポーネントの場合は、Connection オブジェクトのメソッドです。

  - **Ultra Light.NET** ULConnection.RollbackPartialDownload メソッド [Ultra Light.NET]『Ultra Light .NET プログラミング』

  - **Embedded SQL** ULRollbackPartialDownload メソッド [Ultra Light Embedded SQL]『Ultra Light C/C++ プログラミング』

サーバやネットワークが使用できないなどの理由で同期が完了できない場合や、エンドユーザに作業を続行させながらデータの一貫性を維持したい場合に、失敗したダウンロードから変更をロールバックできます。

通信エラーの詳細については、エラーメッセージを参照してください。

注意
send_download_ack 同期パラメータが true に設定されている場合、再開されたダウンロードでは設定は無視されます。

ダウンロード確認

ダウンロード確認は同期のオプションのコンポーネントであり、リモートデータベースでダウンロードが正常に適用されると、クライアントは直ちに Mobile Link サーバに通知します。このことは、リモートのダウンロード受信の受け取り後、できるだけ早くビジネスロジックを実行する必要がある展開の場合におすすめします。リモートでデータが受信されたことを保証する目的では、ダウンロード確認は必要ありません。

ダウンロード確認には、サポートされなくなったブロッキングと、非ブロッキングの 2 つのモードがあります。すべてのダウンロード確認は非ブロッキングとして処理されるようになりました。

ダウンロード確認を使用するには、クライアントとサーバの両方に設定があります。

クライアントでは、dbmlsync の拡張オプション SendDownloadACK または Ultra Light の同期パラメータ Send Download Acknowledgement を使用してダウンロード確認を指定します。

サーバでは、非ブロッキングダウンロード確認を使用している場合、統合データベースに最後の正常なダウンロードの時刻を記録するために 2 つの接続イベント (publication_nonblocking_download_ack と nonblocking_download_ack) を使用できます。

注意
ダウンロード確認は、再開可能なダウンロードでは使用できません。「ダウンロードの失敗の再開」134 ページを参照してください。

参照

- 「publication_nonblocking_download_ack 接続イベント」406 ページ
- 「nonblocking_download_ack 接続イベント」400 ページ
- dbmlsync： 「SendDownloadAck (sa) 拡張オプション」『Mobile Link クライアント管理』
- Ultra Light： 「Send Download Acknowledgement 同期パラメータ」『Ultra Light データベース管理とリファレンス』

ストアドプロシージャコールからの結果セット

ストアドプロシージャコールから結果セットをダウンロードできます。たとえば、次のテーブルに対する download_cursor があるとします。

```
CREATE TABLE MyTable (
   pk INTEGER PRIMARY KEY NOT NULL,
   col1 VARCHAR(100) NOT NULL,
   col2 VARCHAR(20) NOT NULL,
   employee VARCHAR(100) NOT NULL,
   last_modified TIMESTAMP NOT NULL DEFAULT TIMESTAMP
)
```

download_cursor テーブルスクリプトは次のようになります。

```
SELECT pk, col1, col2
  FROM MyTable
    WHERE last_modified >= {ml s.last_table_download}
    AND employee = {ml s.username}
```

MyTable へのダウンロードに、より高度なビジネス論理を使用する場合は、次のようにスクリプトを作成できます。DownloadMyTable は、2 つのパラメータ (最終ダウンロードタイムスタンプと Mobile Link ユーザ名) を取り、結果セットを返すストアドプロシージャです (この例では、移植性のため ODBC 呼び出し規則を使用しています)。

```
{call DownloadMyTable( {ml s.last_table_download}, {ml s.username} )}
```

次に、サポートされる統合データベースごとの簡単な例を示します。詳細については、使用している統合データベースのマニュアルを参照してください。

次の例は、SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server に適用されます。

```
CREATE PROCEDURE DownloadMyTable
   @last_dl_ts DATETIME,
   @u_name VARCHAR( 128 )
AS
BEGIN
    SELECT pk, col1, col2
      FROM MyTable
        WHERE last_modified >= @last_dl_ts
        AND employee = @u_name
END
```

Oracle では、ストアドプロシージャで定義された REF CURSOR によって結果セットを返すことができます。ただし、SQL Anywhere 16 - Oracle ODBC ドライバを使用している場合は、REF CURSOR パラメータをストアドプロシージャのパラメータリストの最後に定義する必要があります。REF CURSOR パラメータは、OUT または IN OUT として定義できます。次のストアドプロシージャは、Oracle で動作します。

```
create or replace procedure DownloadMyTable(
   v_last_dl_ts IN TIMESTAMP,
   v_user_name IN VARCHAR,
   v_ref_crsr OUT SYS_REF_CURSOR ) As
Begin
   Open v_ref_crsr For
      select pk, col1, col2
         from MyTable
            where last_modified >= v_last_dl_ts
            and employee = v_user_name;
End DownloadMyTable;
```

次に、ml_add_table_script ストアドプロシージャを使用して、DownloadMyTable の呼び出しを同期テーブル MyTable の download_cursor スクリプトとして定義します。

```
CALL ml_add_table_script(
 'v1',
 'MyTable',
 'download_cursor',
 '{CALL DownloadMyTable(
    {ml s.last_table_download},{ml s.username} )}'
);
```

Oracle では、DownloadMyTable ストアドプロシージャは 3 つではなく 2 つのパラメータのみを受け取り、Mobile Link サーバは REF CURSOR によって結果セットをフェッチします。REF CURSOR は、ストアドプロシージャの定義で最後のパラメータとして定義されています。

次の例は、IBM DB2 LUW に適用されます。

```
CREATE PROCEDURE DownloadMyTable(
  IN last_dl_ts TIMESTAMP,
  IN u_name VARCHAR( 128 ) )
      LANGUAGE SQL
      MODIFIES SQL DATA
      COMMIT ON RETURN NO
      DYNAMIC RESULT SETS 1

      BEGIN
         DECLARE C1, cursor WITH RETURN FOR
            SELECT pk, col1, col2 FROM MyTable
            WHERE last_modified >= last_dl_ts AND employee = u_name;
         OPEN C1;
      END;
```

自己参照テーブル

一部のテーブルは自己参照します。たとえば、employee テーブルに、従業員をリストするカラムと各従業員のマネージャをリストするカラムが含まれていて、マネージャを管理するマネージャの階層がある場合があります。これらのテーブルを同期するのは、困難である可能性があります。それは、Mobile Link のデフォルト動作ではリモートデータベースでのすべての変更を結合するので、効率的ではあっても、トランザクションの順序が失われるためです。

この状況を処理するには、次の 2 つの方法があります。

- SQL Anywhere リモートデータベースを使用している場合は、dbmlsync -tu オプションを使用して、リモートデータベースの各トランザクションが別のトランザクションとして送信されるように指定できます。

- マッピングテーブル (従業員とマネージャのマッピング) を追加して、以前の自己参照テーブルにおけるトランザクションの順序が問題にならないようにします。

参照

- 「-tu dbmlsync オプション」『Mobile Link クライアント管理』

Mobile Link 独立性レベル

Mobile Link は、RDBMS で独立性レベルが有効になっている場合、最適な独立性レベルで統合データベースに接続します。データの一貫性を維持しながら最高のパフォーマンスを提供するデフォルトの独立性レベルが選択されます。

一般的に、Mobile Link はアップロードには独立性レベル SQL_TXN_READ_COMMITTED を使用し、可能な場合、ダウンロードにはスナップショットアイソレーションを使用します。スナップショットアイソレーションが使用できない場合、Mobile Link は SQL_TXN_READ_COMMITTED を使用します。SQL_TXN_READ_COMMITTED アイソレーションを使用したダウンロードは、別のトランザクションが完了するまでブロックされる可能性があります。このようなブロッキングが発生すると、同期のスループットが大幅に低下することがあります。スナップショットアイソレーションは、トランザクションが統合データベースで閉じられるまでダウンロードがブロックされる問題を解消します。ただし、ダウンロードによって更新が実行されないことが前提であり、このような処理が実行されないようにすることをおすすめします。

スナップショットアイソレーションを使用すると、重複データがダウンロードされる可能性があります (たとえば、実行時間の長いトランザクションによって同じスナップショットが長時間使用される場合)。しかし、Mobile Link クライアントはこの問題を自動的に処理するので、低下するのはリモート側での転送時間と処理作業量だけです。それでも、トランザクションが長時間実行されないようにすることをおすすめします。

通常、独立性レベル 0 (READ UNCOMMITTED) は同期には不適切で、データの不整合を引き起こす可能性があります。

独立性レベルは、統合データベースへの接続が行われた直後に設定されます。その時点でさらに他の接続設定が行われてから、トランザクションがコミットされます。独立性レベルと、場合によってはその他の設定を有効にするために、ほとんどの RDBMS で COMMIT が必要です。

SQL Anywhere バージョン 10 以降の統合データベース

SQL Anywhere バージョン 10 以降はスナップショットアイソレーションをサポートしています。デフォルトでは、Mobile Link はアップロードには SQL_TXN_READ_COMMITTED 独立性レベルを、ダウンロードにはスナップショットアイソレーションを使用します。

SQL Anywhere 統合データベースでスナップショットアイソレーションを有効にした場合だけ、Mobile Link はスナップショットアイソレーションを使用できます。スナップショットアイソレーションが有効でない場合、Mobile Link はデフォルトの SQL_TXN_READ_COMMITTED を使用します。

データベースがスナップショットアイソレーションを使用できるようにすると、パフォーマンスに影響を与える可能性があります。これは、スナップショットアイソレーションを使用するトランザクションの数に関係なく、修正されたすべてのローのコピーを保持する必要があるからです。「スナップショットアイソレーションを有効化する方法」『SQL Anywhere サーバ SQL の使用法』を参照してください。

アップロードに対してスナップショットアイソレーションを有効にするには mlsrv16 -esu オプションを使用し、スナップショットアイソレーションを無効にするには mlsrv16 -dsd オプションを使用します。接続スクリプトで Mobile Link のデフォルトの独立性レベルを変更する必要がある場合は、begin_upload スクリプトまたは begin_download スクリプトで変更してください。

begin_connection スクリプトでデフォルトの独立性レベルを変更すると、アップロードとダウンロードトランザクションの開始時に設定が上書きされます。

「-esu mlsrv16 オプション」61 ページと「-dsd mlsrv16 オプション」59 ページを参照してください。

バージョン 10 より前の SQL Anywhere バージョンの統合データベース

バージョン 10 より前の SQL Anywhere を使用している場合、デフォルトの Mobile Link 独立性レベルは SQL_TXN_READ_COMMITTED です。begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

Adaptive Server Enterprise 統合データベース

Adaptive Server Enterprise では、デフォルトの Mobile Link 独立性レベルは SQL_TXN_READ_COMMITTED です。begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

Oracle 統合データベース

Oracle はスナップショットアイソレーションをサポートしていますが、READ COMMITTED と呼ばれています。デフォルトでは、Mobile Link は、アップロードとダウンロードに対してスナップショットアイソレーション／READ COMMITTED 独立性レベルを使用します。

begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

Mobile Link サーバがスナップショットアイソレーションを最大限有効に利用できるようにするには、Mobile Link サーバが使用する Oracle アカウントは、Oracle システムビュー GV_$TRANSACTION に対するパーミッションを持っている必要があります。持っていない場合には、警告が表示され、ローはダウンロードで失われることがあります。SYS だけがこのアクセス権を付与できます。このアクセス権を付与する Oracle の構文は次のとおりです。

```
grant select on SYS.GV_$TRANSACTION to user-name;
```

Microsoft SQL Server 2005 以降の統合データベース

Microsoft SQL Server 2005 はスナップショットアイソレーションをサポートしています。デフォルトでは、Mobile Link はアップロードには SQL_TXN_READ_COMMITTED 独立性レベルを、ダウンロードにはスナップショットアイソレーションを使用します。

SQL Server 統合データベースでスナップショットアイソレーションを有効にした場合だけ、Mobile Link はスナップショットアイソレーションを使用できます。スナップショットアイソレーションが有効でない場合、Mobile Link はデフォルトの SQL_TXN_READ_COMMITTED を使用します。詳細については、SQL Server のマニュアルを参照してください。

アップロードに対してスナップショットアイソレーションを有効にするには mlsrv16 -esu オプションを使用し、スナップショットアイソレーションを無効にするには mlsrv16 -dsd オプションを使用します。接続スクリプトで Mobile Link のデフォルトの独立性レベルを変更する必要があ

る場合は、begin_upload スクリプトまたは begin_download スクリプトで変更してください。begin_connection スクリプトでデフォルトの独立性レベルを変更すると、アップロードとダウンロードトランザクションの開始時に設定が上書きされます。

「-esu mlsrv16 オプション」61 ページと「-dsd mlsrv16 オプション」59 ページを参照してください。

SQL Server でスナップショットアイソレーションを使用するには、Mobile Link サーバをデータベースに接続するのに使用するユーザ ID が、SQL Server システムテーブル SYS.DM_TRAN_ACTIVE_TRANSACTIONS にアクセスするパーミッションを持っている必要があります。このパーミッションが与えられていない場合、Mobile Link はデフォルトレベルの SQL_TXN_READ_COMMITTED を使用します。

統合データベースが、他のデータベースも実行している Microsoft SQL Server で実行している場合、アップロードまたはダウンロードにスナップショットアイソレーションを使用している場合、およびアップロードまたはダウンロードスクリプトがサーバ上の他のデータベースにアクセスしていない場合は、Mobile Link サーバの -dt オプションを指定してください。このオプションにより、Mobile Link は、現在のデータベース内のトランザクションを除く、すべてのトランザクションを無視します。これにより、スループットが向上したり、ダウンロードされるローの重複が減少したりする可能性があります。

「-dt mlsrv16 オプション」59 ページを参照してください。

参照

- 「-dsd mlsrv16 オプション」59 ページ
- 「-esu mlsrv16 オプション」61 ページ
- 「-dt mlsrv16 オプション」59 ページ
- 「同期処理」『Mobile Link クイックスタート』
- 「独立性レベルと一貫性」『SQL Anywhere サーバ SQL の使用法』
- 「スナップショットアイソレーション」『SQL Anywhere サーバ SQL の使用法』

Mobile Link のパフォーマンス

次に、Mobile Link の最高のパフォーマンスを引き出すために推奨される事項のリストを示します。

テスト

さまざまな要素が、同期システムのスループットに総合的な影響を及ぼします。例を次に示します。

- リモートデータベースを実行するデバイスのタイプ
- リモートデータベースのスキーマ
- リモートのデータ量と同期頻度
- ネットワーク特性 (HTTP、プロキシ、Web サーバ、Relay Server などの特性)

●Mobile Link サーバを実行するハードウェア

●同期スクリプト

●同時に同期するデータ量

●使用する統合データベースのタイプ

●統合データベースを実行するハードウェア

●すべての非同期アクティビティを含む、統合データベースにおけるアクティビティ

●統合データベースのスキーマ

テストは非常に重要です。配備する前に、運用環境で使用するのと同じハードウェアとネットワークを使用してテストを実行してください。また、リモートの数、同期頻度、データ量も同じにしてテストを実行します。このようなテストには、Mobile Link Replay Tool が役立ちます。「Mobile Link Replay ユーティリティ (mlreplay)」696 ページを参照してください。

テストでは、以下のパフォーマンスのヒントを参考にしてください。

競合の回避

同期スクリプトでは、競合を回避し、同時実行性を最大化します。

たとえば、begin_download スクリプトが、テーブル内のカラムを増分してダウンロードの合計数をカウントするとします。複数のユーザが同時に同期を行う場合、このスクリプトによって効果的にダウンロードが直列化されます。begin_synchronization、end_synchronization、prepare_for_download の各スクリプトでも同じカウンタが有効です。これらのスクリプトはコミットの直前に呼び出され、データベースのロックは短時間ですむからです。リモート ID ごとにカウントし、後でクエリによって合計数を取得する方法をおすすめします。

「競合」149 ページを参照してください。

同期のトランザクション構造については、「同期処理のトランザクション」『Mobile Link クイックスタート』を参照してください。

マルチスレッドネットワーク処理の使用

Mobile Link サーバにより、ネットワークストリームを同時に処理する複数のネットワークワーカスレッドがサポートされます。複数のネットワークワーカスレッドが存在することで、特に暗号化や圧縮など、大規模な同期または小規模の多数の同期を伴う CPU 集約型ネットワークストリームオプションを使用しているときは、パフォーマンスが向上することがあります。システム内の各要求は、最大で 1 つのネットワークストリームスレッド上でアクティブにできます。

ストリームスレッドを設定するには、-wn mlsrv16 オプションを使用します。「-wn mlsrv16 オプション」80 ページを参照してください。

最適な数のデータベースワーカスレッドの使用

Mobile Link データベースワーカスレッドでは、固定した数を指定するか、または Mobile Link サーバに自動的に調整させるようにするかを選択できます。-wm オプションを使用せず、固定数

を使用する場合には、-w オプションにさまざまな値を実際に設定しながら、最適なスループットを達成できる最小の数を決定する必要があります。自動調整の場合は、-wm オプションでデータベースワーカスレッドの最大数を指定します。また、-w オプションは最小数となる初期値を指定するために使用します。-w オプションを使用しない場合は、デフォルトは 5 です。たとえば、-wm 50 を使用して、-w オプションを使用しない場合には、5 ～ 50 の範囲内で、Mobile Link サーバによってアクティブなデータベースワーカスレッドの数が定期的に調整されます。

データベースワーカスレッドの数が多いほど、統合データベースに同時にアクセスできる同期の数が多くなり、スループットが向上する「可能性があります」が、競合やブロッキングが発生する可能性が高くなります。

データベースワーカスレッドの数を小さい値にしておくと、統合データベースでの競合発生回数、統合データベースへの接続数、最適なキャッシュに必要なメモリを少なくすることができます。

参照

- 「データベースワーカスレッド数」149 ページ
- 「データベースワーカスレッドの自動調整」144 ページ
- 「-wm mlsrv16 オプション」90 ページ
- 「-w mlsrv16 オプション」89 ページ
- 「-wu mlsrv16 オプション」91 ページ

データベースワーカスレッドの自動調整

Mobile Link サーバは、最大のスループットを実現できるように、負荷に基づいてデータベースワーカスレッドの数を自動調整できます。この自動調整を有効にするには、-w mlsrv16 オプションを使用して、同時に使用するデータベースワーカスレッド数の初期値を設定します。また、-wm mlsrv16 オプションを使用して、同時に使用するデータベースワーカスレッド数の最大値を設定します。Mobile Link サーバはシステムのパフォーマンスをモニタしながら、そのときの状況に応じて、-w および -wm　オプションで設定したパラメータの範囲内でデータベースワーカスレッドの数を増減することによって調整します。

統合データベースへのアップロードに同時に適用可能なデータベースワーカスレッドの最大数を指定する、-wu mlsrv16 オプションで設定した値は、自動的には調整されません。-wu オプションが指定されない場合は、任意またはすべてのデータベースワーカスレッドにアップロードが適用される可能性があります。適用されるデータベースワーカスレッドの数は変動します。Mobile Link サーバがスループットを高めようとしてデータベースワーカスレッドの数を増加させると、問題が検出されてスレッド数を減少させるまでの間、一時的に競合の発生が増加してしまう可能性があります。

サイズの小さいアップロードトランザクションの使用

サイズの大きいアップロードを実行すると、統合データベースにおけるトランザクションが大規模になる可能性があります。トランザクションが大規模になると、トランザクションで保持されるロックが多くなるため、ブロッキングと競合が増えます。これにより、同期スループットと統合データベース全体のスループットの両方が大幅に低下することがあります。アップロードのサイズを小さくすると、ブロッキングと競合が減り、スループットは大幅に向上する可能性があります。

SQL Anywhere リモートを使用した Mobile Link 同期システムでは、次のいずれかの方法で、dbmlsync からサイズの小さいアップロードを送信できます。

- dbmlsync の -tu オプションを使用して、トランザクション単位のアップロードを実行します。各トランザクションが個別に送信されます。「-tu dbmlsync オプション」『Mobile Link クライアント管理』を参照してください。

- dbmlsync の Increment (inc) 拡張オプションを使用して、インクリメンタルアップロードを実行します。各インクリメントに、結合されたトランザクションが含まれます。インクリメントが大きいほど、一般的に 1 回のアップロードに結合されるトランザクション数は多くなります。「Increment (inc) 拡張オプション」『Mobile Link クライアント管理』を参照してください。

サーバ側では、-tx オプションを使用してクライアントの多数のトランザクションを 1 つの統合側のトランザクションにまとめることによって、パフォーマンスをチューニングできます。このオプションは、クライアント側のオプションを設定してある場合に、クライアントを変更しないで -tx をチューニングするだけで済むため、便利です。「-tx mlsrv16 オプション」83 ページを参照してください。

スループットを最大化するために、これらのクライアント側とサーバ側のオプションをテストしてチューニングします。

不要な BLOB の同期の回避

BLOB が変更されていない場合、頻繁に同期されるローに BLOB を含めるのは非効率的です。これを避けるには、BLOB と BLOB ID を含むテーブルを作成し、このテーブル内で同期が必要な ID を参照します。

データベース接続の最大数の設定

Mobile Link データベース接続の最大数を、同期スクリプトバージョンの数と Mobile Link データベースワーカスレッド数の積に 1 を加えた数に設定します。これにより、Mobile Link がデータベース接続を閉じたり作成したりする必要が少なくなります。接続の最大数は mlsrv16 -cn オプションで設定します。

「Mobile Link データベース接続」151 ページと「-cn mlsrv16 オプション」55 ページを参照してください。

十分な物理メモリの確保

Mobile Link サーバを実行しているコンピュータに、他のメモリ要件に加えて、キャッシュを確保する十分な物理メモリがあることを確認してください。サーバに 1.5 GB を超えるメモリキャッシュが必要な場合は、64 ビットプラットフォームへの移行を検討してください。

アクティブに処理される同期の数は、データベースワーカスレッドの数によって制限されません。Mobile Link サーバでは、多数の同期のアップロードのアンパックとダウンロードの送信を同時に行うことができます。サーバでディスクへのページングが開始されると、サーバのスループットは大幅に低下します。したがって、同期をディスクにページングしないで処理するために十分なメモリキャッシュが Mobile Link サーバにあることが重要です。キャッシュが小さすぎるかどうかを判断するために、サーバログに警告 10082 があるかどうか、または、Mobile Link 用 SQL Anywhere モニタに「キャッシュがいっぱいです」という警告があるかどうかを検索します。

Mobile Link サーバでは、必要に応じて、メモリを自動的に拡張または縮小します。Mobile Link サーバのメモリキャッシュを制御するには、-cmax、-cmin、-cinit オプションを使用します。

参照

- 「-cinit mlsrv16 オプション」53 ページ
- 「-cmax mlsrv16 オプション」54 ページ
- 「-cmin mlsrv16 オプション」54 ページ

十分な処理能力の使用

十分な処理能力を Mobile Link に確保して、Mobile Link サーバの処理がボトルネックにならないようにしてください。通常、Mobile Link サーバで必要とされる CPU 処理能力は、統合データベースの場合に比べて非常に小さくてすみます。ただし、Java または .NET のローハンドリングを使用する場合は、Mobile Link サーバの処理要件が増えます。実際には、ボトルネックの要因になることが多いのは、ネットワークの制限やデータベースの競合です。

スクリプト実行時の最適化

統合データベースのスクリプトを実行するときのパフォーマンスは重要です。テーブルに対してインデックスを作成し、アップロードカーソルスクリプトとダウンロードカーソルスクリプトによって必要なローだけが効率的に特定されるようにすると便利です。ただし、インデックスが多すぎるとアップロードに時間がかかります。

Sybase Central で**同期モデル作成ウィザード**を使用して Mobile Link アプリケーションを作成する場合、モデルを展開すると、インデックスがダウンロードカーソルごとに自動的に定義されます。

最小の冗長ロギングの使用

ビジネスニーズに合う最小の冗長ロギングを使用してください。デフォルトでは、冗長ロギングは設定されておらず、Mobile Link はディスクにログを書き込みません。ロギングの冗長性を制御するには、-v オプションを使用します。また、-o オプションか -ot オプションを使用すると、ファイルへのロギングを有効にできます。

冗長ログファイルの代わりに、Mobile Link プロファイラで同期をモニタできます。Mobile Link プロファイラは Mobile Link サーバと同じコンピュータ上になくてもかまいません。また、モニタ接続が Mobile Link サーバのパフォーマンスに与える影響はほとんどありません。「Mobile Link プロファイラ」191 ページを参照してください。

オペレーティングシステムの制限の考慮

TCP/IP を介してサーバがサポートできる同時接続数は、オペレーティングシステムによって制限されます。1000 を超えるクライアントが同時に同期しようとしたときにこの制限に達する可能性があります。この制限に達すると、オペレーティングシステムで予期しない動作が発生する可能性があります。たとえば、接続が予期せずに終了したり、接続しようとするクライアントが拒否されたりします。この動作を防ぐには、TCP/IP 接続の制限を引き上げるようにオペレーティングシステムを設定して -nc オプションを設定するか、-sm オプションを使用して、オペレーティングシステムの制限より少ない数をリモート接続の最大数として指定します。

クライアントは、-sm オプションで指定された同時同期の最大数をすでに受け入れている Mobile Link サーバと同期しようとすると、エラーコード -1305

(SQLE_MOBILINK_COMMUNICATIONS_ERROR) を受信します。クライアントアプリケーションはこのエラーを処理して、数分後に接続を再び試みます。

次の項を参照してください。

- 「-sm mlsrv16 オプション」79 ページ
- 「-nc mlsrv16 オプション」64 ページ
- 「通信エラーが発生しました。」『エラーメッセージ』

Java または .NET の同期論理と SQL 同期論理

Java または .NET の同期論理と、SQL 同期論理では、スループットに著しい差は見られません。ただし、Java と .NET の同期論理では、同期ごとに余分なオーバヘッドが発生し、より多くのメモリが必要です。

さらに、SQL 同期論理は統合データベースが稼働しているコンピュータで実行されますが、Java または .NET の同期論理は Mobile Link サーバが稼働しているコンピュータで実行されます。このため、統合データベースの負荷が高い場合は、Java または .NET の同期論理が適していることがあります。

ダイレクトローハンドリングを使用して同期すると、Mobile Link サーバの負荷が高くなるので、ダイレクトローハンドリングの実装方法によっては、必要な RAM、ディスク容量、CPU 処理能力が増える可能性があります。

優先同期

一部のテーブルを他のテーブルより高い頻度で同期する必要がある場合は、それらのテーブル用に別のパブリケーションとサブスクリプションを作成します。Sybase Central で同期モデルを使用する場合、これを行うには、複数のモデルを作成します。この優先パブリケーションを他のパブリケーションより頻繁に同期し、他のパブリケーションをピーク時以外の時間に同期できます。

必要なローだけのダウンロード

スナップショットではなくタイムスタンプ同期を使用するなどして、必要なローだけをダウンロードするようにしてください。不必要なローのダウンロードは無駄であり、同期のパフォーマンスに悪影響を及ぼします。

必要な場合にのみ同期

同期の頻度が高すぎると、Mobile Link 同期システムに無用なな負荷がかかる場合があります。必要な同期の頻度は慎重に決定してください。徹底的にテストを実施して、期待するパフォーマンスが運用環境で達成されることを確認してください。

大規模なアップロードのロー数の推定

SQL Anywhere クライアントの場合、多数のローをアップロードするときは、アップロードするロー数の推定値を dbmlsync に指定すると、アップロード速度を大幅に改善できます。この操作には、dbmlsync -urc オプションを使用します。

「-urc dbmlsync オプション」『Mobile Link クライアント管理』を参照してください。

バックグラウンド同期の使用

リモートユーザの観点では、バックグラウンドでの同期がより高い頻度で実行されると、同期をできるだけ速く実行する緊急性は低くなります。同期中もリモートユーザが作業を継続できるように、リモートアプリケーションでバックグラウンド同期が使用されるように設計することを検討してください。

Mobile Link のパフォーマンスに影響を与える主要な要因

Mobile Link 同期のためのスループットなど、システム全体のパフォーマンスは、通常、そのシステムのある時点のボトルネックによって制限されます。Mobile Link 同期では、同期スループットを制限するボトルネックとして次のものが考えられます。

- **統合データベースのパフォーマンス**　Mobile Link で特に重要なのは、統合データベースによる Mobile Link スクリプトの実行速度です。複数のデータベースワーカスレッドがスクリプトを同時に実行するので、最高のスループットを得るには、同期スクリプトでのデータベース競合を避ける必要があります。

- **Mobile Link のデータベースワーカスレッド数**　スレッド数が少なければ少ないほど、データベース接続数は少なくなるので、統合データベースでの競合発生率は低くなり、オペレーティングシステムのオーバヘッドは少なくなります。ただし、データベースワーカスレッドの数があまりに少ないと、ワーカスレッドが解放されるまでクライアントが待機したままになったり、統合データベースへの接続数が少ないために、効率的な重複ができなくなったりします。

- **クライアントから Mobile Link への通信の帯域幅**　ダイヤルアップ接続、広域ネットワーク (WAN) や無線ネットワークでの接続など、通信速度の遅い接続では、クライアントと Mobile Link サーバがデータ転送を待たなければならないことがあります。

- **クライアントの処理速度**　ダウンロードには、ローやインデックスの書き込みなど、より多くのクライアント処理が含まれるため、クライアントの処理速度の遅さは、アップロードよりもダウンロードでボトルネックになる可能性が高くなります。

- **Mobile Link サーバを実行しているコンピュータの処理速度**　Mobile Link を実行しているコンピュータの処理能力が低い場合や、Mobile Link データベースワーカスレッドとバッファに十分なメモリがない場合には、Mobile Link の実行速度が同期のボトルネックになることがあります。バッファとデータベースワーカスレッドが物理メモリに収まる場合は、Mobile Link サーバのパフォーマンスがディスク速度に左右されることはほとんどありません。

- **Mobile Link から統合データベースへの通信の帯域幅**　Mobile Link と統合データベースが同じコンピュータで稼働している場合や、別々のコンピュータで稼働していても高速ネットワークで接続されている場合には、帯域幅がボトルネックになる可能性は低くなります。

Mobile Link のパフォーマンスのチューニング

Mobile Link 同期のスループットを最大限にするには、複数の同期が同時に発生し、効率よく実行することが重要になります。複数の同時同期を有効にするため、Mobile Link ではさまざまなタスク用にデータベースワーカスレッドのプールを用意しています。そのうちの 1 つは、ネット

ワークからのアップロードデータの読み込みとそのアンパック専用です。「データベースワーカスレッド」と呼ばれる別のスレッドプールは、統合データベースへのアップロードの適用と統合データベースからダウンロードするデータのフェッチに使用されます。また別のデータベースワーカスレッドのプールは、ダウンロードデータのパックとリモートデータベースへの送信専用です。各データベースワーカスレッドは、同期スクリプトを使い、統合データベースへの単一の接続を使用して、変更の適用とフェッチを行います。

競合

最も重要な要因は、同期スクリプトでのデータベース競合を避けることです。複数のクライアントがデータベースを使用する他の場合と同様、クライアントがデータベースに同時にアクセスするときには、データベース競合を最小限にします。同期のたびに修正が必要なデータベースのローがあると、競合の発生率が高くなります。たとえば、あるローのカウンタを増分するスクリプトがある場合、カウンタを更新することがボトルネックになる可能性があります。

同期要求は (-sm オプションで指定された制限数まで) 受け付けられ、アップロードされたデータの読み込みとアンパックが行われて、データベースワーカスレッドで使用できるようになります。データベースワーカスレッドの数より同期の数が多い場合は、超過分はキューに追加されて、データベースワーカスレッドに空きが出るのを待機します。

データベースワーカスレッド数と接続数は制御できますが、Mobile Link は、1 つのデータベースワーカスレッドに最低 1 つの接続があることを常に確認します。データベースワーカスレッドよりも多くの接続がある場合、余分な接続はアイドル状態です。スクリプトバージョンが複数ある場合には、後述するように、余分な接続が役に立つこともあります。

データベースワーカスレッド数

同期スクリプトで発生する競合以外に、同期スループットの最も重要な要因になるのがデータベースワーカスレッド数です。データベースワーカスレッド数は、統合データベースで同時に進行する同期の数を制御します。

最適なデータベースワーカスレッド数を判別するにはテストが不可欠です。

データベースワーカスレッドの数を増やすと、統合データベースにアクセスできる同期の重複を多くしたり、スループットを向上させたりすることが「できます」。その反面、重複する同期の間でリソースやデータベースの競合が増えて、1 つの同期にかかる時間も長くなります。データベースワーカスレッドの数が増えると、1 つの同期の時間が長くなるというデメリットがあるので、同時同期をより多く実行することが重要になります。また、さらにデータベースワーカスレッドを追加すると、スループットが低下します。使用する環境に最も適したデータベースワーカスレッド数を決定するには実験が必要ですが、次の情報も参考になります。

パフォーマンステストで、最高のアップロードスループットは、一般的にデータベースワーカスレッドの数が 3 ～ 10 のときに得られることがわかりました。統合データベース、データ容量、データベーススキーマ、同期スクリプトの複雑さ、使用したハードウェアなどの要因によって結果は変動します。ボトルネックの一般的な原因は、統合データベース内でアップロードスクリプトの SQL を同時に実行するデータベースワーカスレッド間の競合です。

-w mlsrv16 オプションを使用して、データベースワーカスレッド数を設定します。また、-wm mlsrv16 オプションでは、現在の -w と -wm の間の数字を基に、スループットが最大になるように Mobile Link サーバによってデータベースワーカスレッド数が自動的に調整されるようにもで

きます。ただし、-wm は便利ではあっても、常にうまく機能するとは限りません。いつの場合でも、最適なデータベースワーカスレッド数を判別するにはテストが不可欠です。

ダウンロード確認を使用していない場合 (デフォルト)、別のスレッドでダウンロードを送信している間に特定のデータベースワーカスレッドで同期を処理できるので、クライアントから Mobile Link への帯域幅の影響が小さくなります。したがって、データベースワーカスレッド数は重要ではありません。

データベースワーカスレッド数よりも多くのダウンロードを同時に送信できます。ダウンロードのパフォーマンスを最適化するには、これらのダウンロードのバッファ用の RAM が Mobile Link サーバに十分にあることが重要です。RAM が十分になかった場合、ダウンロードはディスクにページングされるので、ダウンロードのパフォーマンスが低下する可能性があります。Mobile Link サーバのメモリキャッシュサイズを指定するには、-cmax オプションを使用します。

「-cmax mlsrv16 オプション」54 ページを参照してください。

Mobile Link サーバでディスクへのページングが開始されたら (同時に処理されるダウンロードが多すぎる場合など)、-sm オプションを使用して、データベースワーカスレッドの数を減らすか、アクティブに処理される同期の合計数を制限することを検討してください。

「-sm mlsrv16 オプション」79 ページを参照してください。

ダウンロード確認をオフ (デフォルト) のままにしておくと、ダウンロードに使用できるデータベースワーカスレッドの最適数が少なくなります。これは、データベースワーカスレッドがダウンロード確認トランザクションを処理する必要がないからです。

「SendDownloadAck (sa) 拡張オプション」『Mobile Link クライアント管理』を参照してください。

ブロッキングダウンロード確認はサポートされなくなったため、すべてのダウンロード確認は非ブロッキングとして処理され、パフォーマンスは向上します。非ブロッキング確認では、リモートデータベースがダウンロードを適用しているときに、サーバはデータベースワーカスレッドを再利用します。結果、データベースワーカスレッドの数を増やす必要がなく、より高いパフォーマンスを得ることができます。

ダウンロードとアップロードの最高のスループットを得るために、Mobile Link には 2 つのオプションが用意されています。1 つは、ダウンロードを最適化するデータベースワーカスレッドの総数を指定するオプションです。もう 1 つは、アップロードを同時に適用できる数を制限して、アップロードのスループットを最適にするオプションです。

-w オプションは、データベースワーカスレッドの総数を制御します。デフォルトは 5 です。

-wu オプションは、アップロードを同時に統合データベースに適用できるデータベースワーカスレッド数を制限します。デフォルトでは、すべてのデータベースワーカスレッドがアップロードを同時に適用できますが、この場合、統合データベースで重大な競合が発生します。-wu オプションを使用すると、この競合を軽減すると同時に、多数のデータベースワーカスレッドでダウンロードデータのフェッチを最適化できます。-wu オプションは、その数値がデータベースワーカスレッドの合計数未満の場合にだけ有効です。

「-w mlsrv16 オプション」89 ページと「-wu mlsrv16 オプション」91 ページを参照してください。

Mobile Link データベース接続

Mobile Link では、データベースの接続はデータベースワーカスレッドごとに作成されます。-cn オプションを使用すると、Mobile Link に対してデータベース接続のより大きなプールを作成するように指定できますが、Mobile Link が接続を閉じるか、異なるスクリプトバージョンを使用する必要がないかぎり、余分な接続はアイドル状態です。

Mobile Link がデータベース接続を閉じ、新しい接続を開く場合が 2 つあります。1 つはエラーが発生した場合です。もう 1 つは、クライアントがある同期スクリプトバージョンを要求し、現在使用されている接続の中にその同期スクリプトバージョンを利用できる接続がない場合です。

注意
それぞれのデータベース接続はスクリプトバージョンと関連付けられています。スクリプトバージョンを変更するには、接続を閉じて再度開いてください。

定常的に複数のスクリプトバージョンを使用するのであれば、接続数を増やすことで Mobile Link での接続開閉要求を減らすことができます。同期に使用する接続数が、データベースワーカスレッド数とスクリプトバージョン数の積になる場合は、接続数を増やす必要はありません。

次のコマンドラインには、2 つのスクリプトバージョンに対して Mobile Link を調整する例が示されています。

```
mlsrv16 -c "DSN=SQL Anywhere 16 Demo" -w 5 -cn 10
```

同期に使用するデータベース接続の最大数は、スクリプトバージョン数とデータベースワーカスレッド数の積です。したがって、-cn を 10 に設定することで、データベース接続が無駄に閉じられたり開かれたりすることを排除できます。

「-cn mlsrv16 オプション」55 ページを参照してください。

Mobile Link のパフォーマンスのモニタ

さまざまなツールを使用して、同期のパフォーマンスをモニタできます。

Mobile Link プロファイラは、同期をモニタするためのグラフィカルツールです。同期時のさまざまな処理に要した時間を確認できます。

「Mobile Link プロファイラ」191 ページを参照してください。

また、同期をモニタするための Mobile Link スクリプトがいくつかあります。これらのスクリプトにより、ビジネス論理でパフォーマンス統計を使用できます。たとえば、パフォーマンス情報を格納して後で分析したり、同期の実行時間が長すぎる場合に DBA に警告できます。これらの

スクリプトを作成する場合は、他のスクリプトの場合と同様に、競合とブロッキングをできるだけ回避するよう注意してください。 詳細については、次の項を参照してください。

- 「download_statistics 接続イベント」334 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ

リモートデータベースの集中管理

SQL Anywhere を使用すると、Mobile Link 同期に関連するリモートデータベースを集中管理できます。

リモートデータベースの集中管理を使用すると、次のことができます。

- リモートデータベースが Mobile Link と同期するときに集中的に管理する。
- リモートデータベースでスキーマの変更を実行する。
- 特定のリモートデータベースまたは一般的な同期システムに関する問題を診断する。
- ログファイルをアップロードする。

リモートデータベースの集中管理は、バージョン 11.0.0 で導入された SQL パススルー機能に置き換わります。SQL パススルー機能を使用すると、統合データベースから SQL Anywhere または Ultra Light のクライアントに SQL 文のスクリプトをダウンロードし、クライアント上で適切な時間にそれらの SQL 文を実行できます。

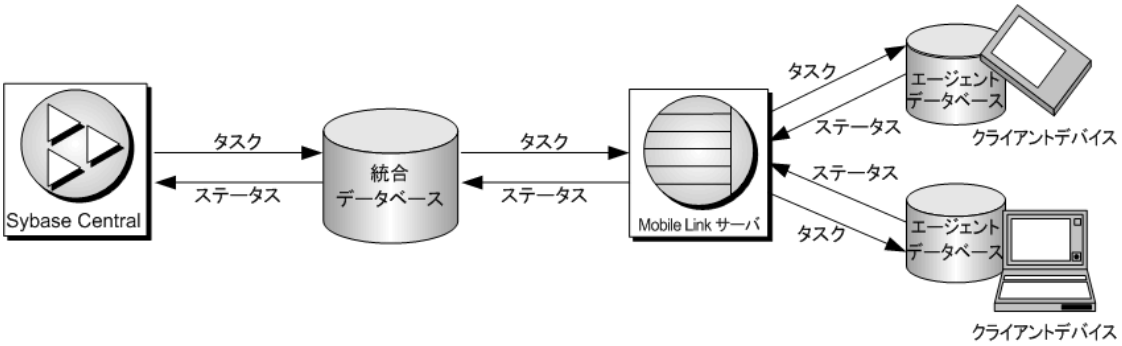
集中管理は、リモートタスクを使用することによって達成されます。リモートタスクは順序付けされたコマンドのコレクション (バッチファイルに似ています) であり、Sybase Central 用の Mobile Link 16 プラグインを使用して作成します。リモートタスクには、実行時期を制御する条件が設定される場合があります。リモートタスクは、1 回のみ実行する、必要に応じて実行する、または定期的に繰り返すように設定できます。リモートタスクは統合データベースに展開され、1 つ以上の Mobile Link エージェントに割り当てることができます。

Mobile Link エージェントは、リモートデバイス上で実行されるアプリケーションです。特定の Mobile Link サーバから集中管理される各データベースセットに対して、Mobile Link エージェントのインスタンスが 1 つ実行されている必要があります。エージェントは、割り当てられたリモートタスクを適切な時間に実行する役割を果たします。Mobile Link エージェントは、エージェントデータベースと呼ばれる Ultra Light データベースを使用して、エージェントに割り当てられたタスク、実行済みタスクの結果、その他の設定情報を保存します。

Mobile Link エージェントは、通常の Mobile Link 同期を使用して、エージェントデータベースを定期的に同期します。同期中に、エージェントは新しく割り当てられたタスクを受信し、オプ

ションとしてタスクの実行結果をアップロードして、システム管理者が Sybase Central 用の Mobile Link 16 プラグインを使用して結果を確認できるようにします。

次の図は、リモートデータベースの集中管理機能を使用している場合の、Sybase Central、統合データベース、クライアントデバイス間での情報のフローを示します。



集中管理の概念

次の概念は、リモートデータベースの集中管理を理解するために重要です。

Mobile Link プロジェクト

Mobile Link プロジェクトは、Sybase Central 用の Mobile Link 16 プラグインを使用して管理者が作成します。まず、Mobile Link プロジェクトを定義してから、リモートタスクを操作してください。

Mobile Link プロジェクトは、次の要素で構成されるコレクションです。

●0 以上のリモートタスク

●統合データベースへの少なくとも 1 つの接続

●0 以上の同期モデル

サンプルの Mobile Link プロジェクトは *%SQLANYSAMP16%¥MobiLink¥CustDB¥project.mlp* にあります。

Mobile Link エージェント

Mobile Link エージェントは、クライアントデバイスで実行されるアプリケーションです。エージェントは、Mobile Link サーバからタスクを受信して実行し、それらのタスクのステータスを Mobile Link サーバに返します。

Mobile Link エージェントでは、クライアントデバイス上の複数のリモートデータベースを管理できます。クライアントデバイス上のリモートデータベースを異なる統合データベースと同期する必要がある場合は、アプリケーションで同期する必要がある統合データベースごとに、異なる Mobile Link エージェントが必要となります。

Mobile Link エージェント ID

Mobile Link エージェント ID は、クライアントデバイスで実行中のエージェントを Mobile Link サーバが識別するための文字列です。Mobile Link プロジェクトを操作する管理者は、Mobile Link エージェント ID を表示できます。また、各エージェントは 1 つのクライアントデバイスで動作するため、管理者はこの ID からデバイスも識別できます。

各 Mobile Link エージェントにはユニークな ID が必要です。エージェント ID は、mlagent コマンドを使用して起動時に指定できます。デフォルト値は、Agent_computername_UUID (*computername* はエージェントが実行されているコンピュータのホスト名、*UUID* はユニバーサルユニーク識別子) の形式で割り当てられます。

異なるエージェント ID を表すのに、大文字と小文字を使用して区別しないことを強くおすすめします。たとえば、エージェント ID **Agent_XYZ** と **agent_xyz** を作成して、異なるエージェント ID として使用しないでください。統合データベースで大文字と小文字が区別されないときは、この推奨事項は必須条件です。統合データベースで大文字と小文字が区別されるときには、この推奨事項は強制ではありません。

リモートタスク

作業の単位は、リモートタスクと呼ばれます。リモートタスクは、コマンドのコレクションです。Mobile Link エージェントは、実行する作業をリモートタスクの形式で受信し、作業の試行ステータスを管理者に返します。

コマンド

コマンドは、特定のアクションを実行するタスク内の命令です。タスクには、複数のコマンドを含めることができ、コマンドには定められた順序があります。コマンドには、実行するアクション、入力パラメータ、コマンドが失敗した場合の処理に関する命令が含まれています。

展開済みリモートタスク

展開済みリモートタスクは、統合データベースにコピーされたタスクです。エージェントに割り当てて実行できるのは、展開済みタスクのみです。

ステータス情報

ステータス情報は、タスクが正常に完了したかどうかなどの、リモートタスクに関する情報です。この情報は、タスクが実行されるとクライアント上でエージェントデータベース内に保存され、管理者がリモートタスクのステータスをシステムで確認できるようにするため、さまざまな時点でサーバに送信されます。

エージェントデータベース

エージェントデータベースは、タスクや設定に関する情報を保存するために Mobile Link エージェントが使用する、リモートデバイス上の Ultra Light データベースです。

エージェントデータベースのデフォルトの場所は、Windows では *%ALLUSERSPROFILE%¥Application Data¥SQL Anywhere 16¥MobiLink Agent*、Windows Mobile では *My Device¥Application Data¥SQLAny16¥MLAgent* です。

エージェントデータベースのファイル名は、*mlagent.exe* の -n オプションで指定された名前に拡張子 *.udb* が付けられたものになります。-n オプションを指定しなかった場合のデフォルト名は *mlagent.udb* です。

リモートデータベース

リモートデータベースは、アプリケーションデータが含まれるリモートデバイス上の Ultra Light データベースまたは SQL Anywhere データベースであり、Mobile Link 同期に関係し、Mobile Link エージェントによって管理されます。各リモートデータベースには、そのスキーマを識別するリモートスキーマ名があります。

リモートスキーマ名

リモートスキーマ名は、同じスキーマを持つデータベースのグループを識別します。一般的に、同じリモートスキーマ名を持つすべてのデータベースは、同じバージョンのアプリケーションのデータベースになります。スキーマには、テーブル定義、ストアドプロシージャ、トリガ、パブリケーション、同期プロファイルなどが含まれます。スキーマには、同期ユーザやデータベースユーザのような、データベースのインスタンスごとに異なる項目は含まれません。

リモートデータベースにリモートスキーマ名がないと、リモートデータベースをリモート管理できません。したがって、少なくとも 1 つのリモートスキーマ名を定義してから、エージェントを Sybase Central で作成してください。

プロジェクト作成ウィザードまたは**統合データベース追加ウィザード**のいずれを使用する場合も、プロジェクトに統合データベースを追加するときは、ウィザードによって、プロジェクトに存在しないリモートスキーマ名が統合データベースに定義されているかどうかが自動的にチェックされます。定義されている場合は、それをインポートするかどうか尋ねられます。

サーバ起動リモートタスク (SIRT)

サーバ起動リモートタスクは、エージェントがタスクの実行指示通知をサーバから受信したときに実行される任意のリモートタスクです。タスクがスケジュールされている場合でも、サーバによって起動されます。

設定概要

次の手順では、サーバとクライアントでリモートデータベースの集中管理を設定するプロシージャの概要を説明します。

サーバでの集中管理の設定

1. Sybase Central で Mobile Link プロジェクトを作成します。 「Mobile Link プロジェクトの作成」『Mobile Link クイックスタート』を参照してください。

2. Sybase Central を使用して、システムに対してリモートデータベースを識別するために、1 つ以上のリモートスキーマ名を定義します。

3. Sybase Central を使用して、リモートデータベースを管理しているすべてのエージェントをシステムで認識します。

4. Sybase Central を使用して、タスクを作成し、エージェントに割り当てます。 「リモートタスク」169 ページを参照してください。

クライアントでの集中管理の設定

1. アプリケーションが実行されているデバイスごとに Mobile Link エージェントを設定します。これにより、システム内で ID が付与され、Mobile Link 接続情報が提供されます。

2. デバイス上で Mobile Link エージェントを実行して、サーバからタスクを受信し実行できるようにします。 「クライアントデバイスでの Mobile Link エージェント」157 ページ を参照してください。

Mobile Link エージェント

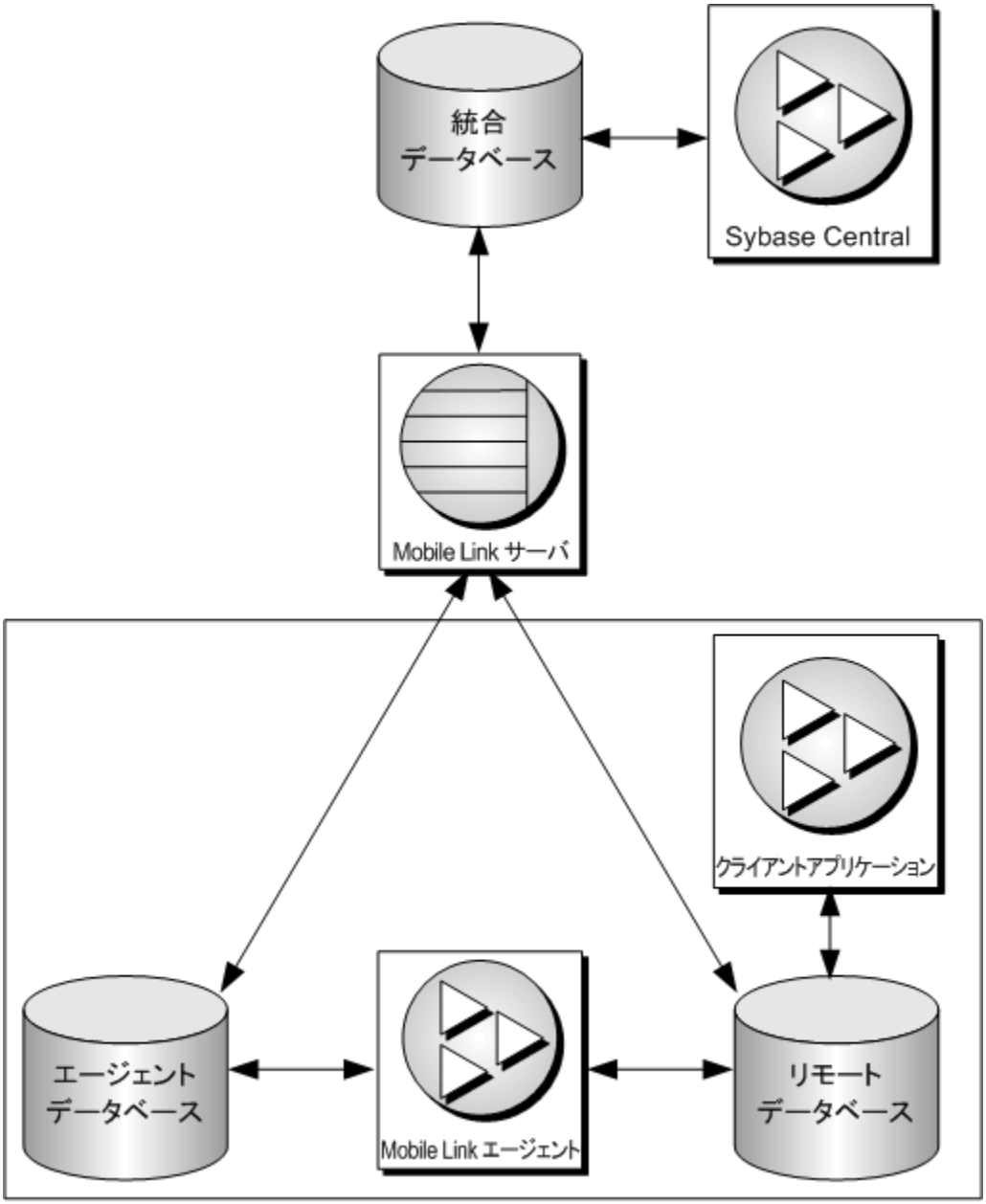
エージェントは、デバイス上のリモートタスクの実行を管理します。実行予定のタスクと、タスクの実行結果をエージェントデータベースに保存します。エージェントは、通常の Mobile Link 同期を使用して、エージェントデータベースを同期します。同期中には、実行する新しいタスクを受信し、実行されたタスクに関する情報をアップロードします。

エージェントは、次の時点でエージェントデータベースを同期します。

- エージェントが起動される。
- エージェントが、エージェントデータベースの同期を指示する通知をサーバから受信する。エージェントはサーバからの通知を受信し、その時点でエージェントデータベースを起動します。
- 最後の同期の実行後に、ユーザが指定した時間が経過した。
- タスクの完了直後にステータスを送信するよう設定されたタスクが完了した。

エージェントはマルチスレッドであるため、複数のタスクを並列して実行できます。タスクは、「最終」ステータスに到達した後にのみ削除されます。最終ステータスには、成功、失敗、失効、キャンセルがあります。タスクが最終ステータスに到達するのは、「排他実行」または「即時実行」のタスクが成功または失敗するか、スケジュールされたタスクが失効するかサーバでキャンセルされた場合のみです。オンデマンドタスクは、キャンセルされないかぎり最終ステータスには到達しません。オンデマンドタスクはエージェントに残り、サーバ起動要求 (SIRT) によって実行されます。

次の図は、エージェントとの間の通信のフローを示します。



クライアントデバイスでの Mobile Link エージェント

エージェントは、2 つのモード (設定モードと通常モード) で実行できます。設定モードでは、コマンドラインで指定したオプションがエージェントデータベースに保存され、次回の通常モードでの実行に使用されます。指定したオプションが保存されると、エージェントは終了します。

通常モードの実行では、エージェントはエージェントデータベースに保存された設定オプションを読み込み、実行を継続します。エージェントは、実行中に受信したリモートタスクを適切な時期に実行し、エージェントデータベースをさまざまな時点で同期して、新しいリモートタスクを受信したり、実行したリモートタスクの結果をアップロードしたりします。

通常モードの実行では、エージェントは常に起動時に同期を試みます。このことは、Mobile Link からの最新情報がエージェントで強制的に取得されるようにする場合に便利です。

参照

● 「mlagent コマンド」 158 ページ

mlagent コマンド

クライアントデバイスで Mobile Link エージェントを設定モードまたは通常モードのいずれかで実行します。

構文

mlagent [*options*]
設定モードで実行するには、mlagent コマンドラインで -c または -cr を指定します。

オプション	説明
@data	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。 「設定ファイル」『SQL Anywhere サーバ データベース管理』を参照してください。
-c	設定オプションを設定し、エージェントを停止します。これらのオプションは、現在のオプションと異なる場合にのみ更新されます。
-cr	設定オプションを設定し、エージェントを停止します。既存のオプションは、すべてデフォルトにリセットされます。このオプションを使用すると、エージェントデータベースのすべての情報がリセットされます。このため、このオプションは、エージェントデータベースが回復不能な状態にある場合にのみ使用してください。

オプション	説明
-a *agentid*	-c または -cr と併用した場合にのみ有効です。このエージェントの ID を指定します。 -a オプションを指定しない場合のデフォルトは Agent_computername_UUID です。*computername* はエージェントが実行されているマシンのコンピュータ名、*UUID* はユニバーサルユニーク識別子です。 異なるエージェント ID を表すのに、大文字と小文字を使用して区別しないことを強くおすすめします。たとえば、エージェント ID **Agent_XYZ** と **agent_xyz** を作成して、異なるエージェント ID として使用しないでください。統合データベースで大文字と小文字が区別されないときは、この推奨事項は必須条件です。統合データベースで大文字と小文字が区別されるときには、この推奨事項は強制ではありません。
-db *database location*	-c または -cr と併用した場合にのみ有効です。リモートデータベースが保存されるクライアントデバイス上のパスを指定します。
-x *protocol*[*protocol-options*] ... *protocol* : **tcpip** | **tls** | **http** | **https** *protocol-options*:(*option*=*value*; ...)	-c または -cr と併用した場合にのみ有効です。Mobile Link クライアントネットワークプロトコルオプションを指定します。これらのオプションにより、エージェントデータベースを同期するときのエージェントから Mobile Link サーバへの接続方法が決まります。
-ap *parameters*	エージェントデータベースを同期するときに使用される Mobile Link 認証パラメータを指定します。
-ek *key*	エージェントデータベースにアクセスするときに使用される暗号化キーを指定します。
-n *name*	エージェントデータベースの名前を指定します。デフォルトは taskdb です。
-o *file*	指定したファイルに出力を書き込みます。

オプション	説明
-on *size*	ログが指定の *size* に達したときに、ログファイルに *.old* を追加し、新しいファイルを元の名前で開始します。サイズは 10K 以上にしてください。このオプションは、-os オプションと同時に使用できません。
-os *size*	ログが指定の *size* に達したときに、ログファイルの名前を *YYMMDDxx.mla* に変更し、新しいファイルを元の名前で開始します。サイズは 10K 以上にしてください。このオプションは、-on オプションと同時に使用できません。
-ot *file*	ファイルをトランケートし、そのファイルに出力メッセージのログを取ります。
-p *password*	エージェントデータベースを同期するときに使用される Mobile Link パスワードを指定します。
-pi	エージェントが同期できるかどうかをテストします。このオプションでは、現在設定されている Mobile Link クライアントのネットワークプロトコルオプションとユーザ認証パラメータを使用して、Mobile Link サーバに対してエージェントデータベースの ping 同期をエージェントで実行します。ping 同期に成功すると mlagent プロセスから 0 が直ちに返され、失敗すると同期要求の SQL エラーコードが返されます。 ping 同期の詳細については、「-pi dbmlsync オプション」『Mobile Link クライアント管理』を参照してください。 -c または -cr を指定した mlagent を実行して Mobile Link エージェントデータベースを設定してから、-pi を指定した mlagent を起動してください。 コマンドラインで -pi と、-c または -cr の両方を指定して、Mobile Link エージェントを起動することはできません。
-q	最小化ウィンドウで実行します。
-qi	トレイアイコンまたはウィンドウを表示しません。

オプション	説明
-u *user*	エージェントデータベースを同期するときに使用される Mobile Link ユーザ名を指定します。
-v*level*	冗長出力レベルを 0 ～ 9 で指定します。デフォルトは 3 です。

例

次の例は、設定モードでのエージェントの実行方法を示します。デフォルトのエージェントデータベースを使用し、エージェント ID を Mobile Link ユーザ ID と同じ値に設定します。

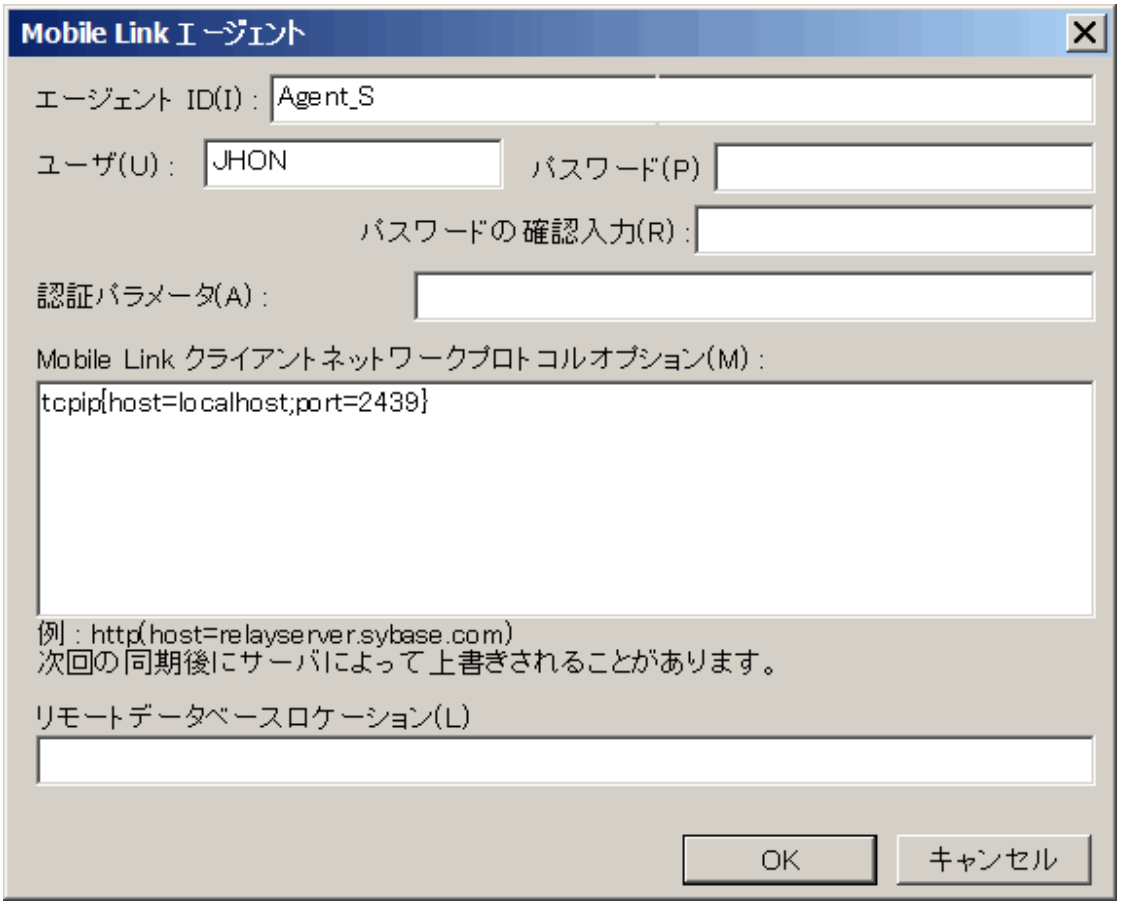
```
mlagent -c -a username -u username -p password -x http{host=myhost.example.com;port=8080} -o
logfile.mla
```

次の例は、通常モードでのエージェントの実行方法を示します。デフォルトのエージェントデータベースから、すべての設定を受け入れます。

```
mlagent
```

Mobile Link エージェントの対話型設定

Mobile Link エージェントは、コマンドラインだけではなく、設定ウィンドウを使用することによっても設定できます。エージェントが実行されており、エージェントデータベースが正しく動作するために十分な情報が提供されていない場合には、設定ウィンドウが自動的に表示されます。または、Mobile Link エージェントウィンドウのメニューを使用することによっても設定ウィンドウを表示できます。Windows では、**[Mobile Link エージェント]** ウィンドウの左上の [システム] メニューをクリックして、**[設定]** をクリックします。Windows Mobile では、メニューバーの [設定] 項目を選択します。



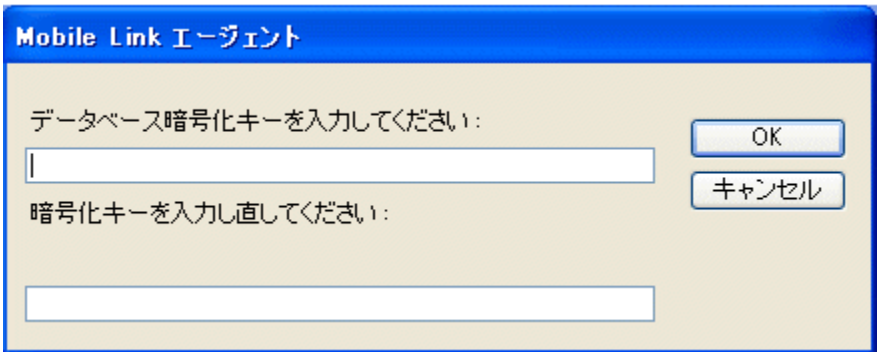
次の表では、設定ウィンドウの項目と Mobile Link エージェントの設定オプションの対照表を示します。

設定ウィンドウのフィールド名	対応する mlagent 設定オプション
[エージェント ID]	mlagent の -a オプション
[ユーザ]	mlagent の -u オプション
[パスワード]	mlagent の -p オプション
[認証パラメータ]	mlagent の -ap オプション
[Mobile Link クライアントのネットワークプロトコルオプション]	mlagent の -x オプション
[リモートデータベースのロケーション]	mlagent の -db オプション

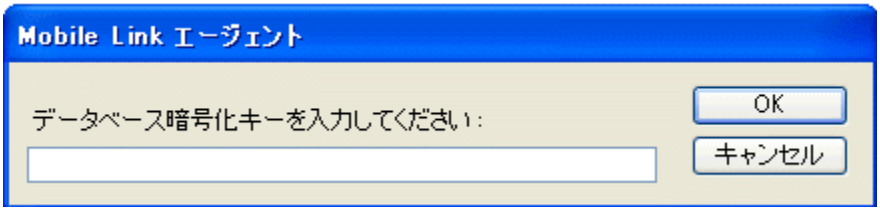
Mobile Link エージェントの設定オプションの詳細については、「クライアントデバイスでの Mobile Link エージェント」157 ページ を参照してください。

暗号化

エージェントデータベースが存在しない場合は、デフォルトの暗号化キーを使用するかどうかを尋ねるメッセージが表示されます。デフォルトの暗号化キーはユーザからは判別しにくいですが、エージェントにハードコードされているため検出されます。デフォルトの暗号化キーを選択しない場合は、表示される次のウィンドウで暗号化キーを入力し、確認します。



エージェントデータベースが存在しているのに、-ek オプションでデフォルト以外の暗号化キーが指定されているために、Mobile Link エージェントが接続できない場合は、表示される次のウィンドウでキーを入力できます。



Mobile Link エージェント停止ユーティリティ

Mobile Link エージェント停止ユーティリティでは、停止ユーティリティが実行されているデバイスで実行中の、Mobile Link エージェントのインスタンスを停止できます。

構文

mlastop [*options*]

オプション	説明
-n *name*	停止するエージェントの名前。-n を指定しないと、デバイス上のすべてのエージェントが停止されます。

参照

- 「mlagent コマンド」158 ページ
- 「SQL Anywhere Mobile Link クライアントの配備」776 ページ
- 「Ultra Light Mobile Link クライアントの配備」779 ページ

Sybase Central の Mobile Link エージェント

mlagent コマンドを使用してクライアントデバイスでエージェントを作成および設定した後、このエージェントを Sybase Central でも作成する必要があります。その後、このエージェントにタスクを割り当てることができます。**Mobile Link エージェント作成ウィザード**では、エージェントの作成手順を説明します。エージェントを正しく識別するために必要な情報を指定してください。

Sybase Central でエージェントを作成するには、リモートスキーマ名が定義済みである必要があります。

参照：

- 「リモートスキーマ名の追加」164 ページ
- 「リモートスキーマ名のインポート」165 ページ
- 「エージェントの追加」165 ページ
- 「エージェントプロパティ」166 ページ
- 「エージェントのプロパティの表示または変更」166 ページ
- 「管理対象のリモートデータベースの追加」167 ページ
- 「グループの追加」167 ページ

リモートスキーマ名の追加

リモートスキーマ名作成ウィザードを使用してリモートスキーマ名を追加します。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

Sybase Central でエージェントを作成する前に、リモートスキーマ名を定義しておく必要があります。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。

   [リモートスキーマ名] をダブルクリックし、**[新規]** » **[リモートスキーマ名]** をクリックします。

2. **リモートスキーマ名作成ウィザード**の指示に従い、**[完了]** をクリックします。

結果

リモートスキーマ名が追加されます。

リモートスキーマ名のインポート

スキーマ名は、別のデータベースからインポートできます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[リモートスキーマ名]** を右クリックし、**[インポート]** をクリックします。
3. リモートスキーマ名をインポートしたいデータベースを、表示された統合データベースのリストから選択し、**[OK]** をクリックします。

結果

選択したデータベースに Mobile Link プロジェクトに存在しないリモートスキーマ名がある場合、それらはインポートされます。

エージェントの追加

エージェントを追加してリモートデータベースの集中管理を使用します。

前提条件

Sybase Central でエージェントを作成する前に、リモートスキーマ名を定義しておく必要があります。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[統合データベース]** をダブルクリックします。
3. **[エージェント]** をダブルクリックし、**[ファイル]** » **[新規]** » **[エージェント]** をクリックします。
4. **Mobile Link エージェント作成ウィザード**の手順に従います。

結果

エージェントが作成されます。

次の手順

Sybase Central でエージェントが作成された後、以下のことを実行できます。

●エージェントのプロパティの表示と変更

●管理するエージェントのリモートデータベースの追加

●エージェントの同期

●エージェントの削除

●エージェントのイベントの表示

●エージェントのタスクの表示

●エージェントで管理されるリモートデータベースに関する情報の表示

エージェントプロパティ

エージェントのプロパティは、Sybase Central の **[エージェントのプロパティ]** ウィンドウで表示、編集できます。また、エージェントを右クリックし **[設定]** を選択して、次のプロパティを設定することもできます。

- **[同期間隔]**　同期間隔は、エージェントがエージェントデータベースを同期する頻度を制御します。

- **[管理ポーリング間隔]**　管理ポーリング間隔によって、サーバからの同期要求や他のアクションの実行要求をエージェントがチェックする頻度が決まります。

- **Mobile Link クライアントのネットワークプロトコルオプション**　Mobile Link クライアントのネットワークプロトコルオプションは、クライアントで指定されエージェントを初めて同期するときにサーバに送信されたエージェントのプロパティです。管理者がエージェントの Mobile Link プロトコルオプションを変更すると、新しい値が同期時にエージェントに送信され、以降の Mobile Link とのすべての通信で新しい値が使用されます。

  管理者が無効な Mobile Link 通信オプション (誤って設定されたサーバホスト名など) を送信した場合は、その値を受信したエージェントが Mobile Link サーバと通信できなくなることがあります。この場合、管理者は Sybase Central で Mobile Link クライアントのネットワークプロトコルオプションを修正し、その後、デバイス上でエージェントに正しいオプションを再設定する必要があります。

- **[管理対象データベースの接続文字列]**　Mobile Link エージェントがリモートデータベースとの接続に使用できる接続文字列。

エージェントのプロパティの表示または変更

エージェントプロパティは、表示または更新することができます。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. 統合データベースをダブルクリックします。
2. **[エージェント]** をダブルクリックし、操作するエージェントを右クリックして、**[プロパティ]** をクリックします。
3. 必要に応じてプロパティに変更を加え、**[適用]** をクリックします。

結果

エージェントのプロパティが更新されます。

管理対象のリモートデータベースの追加

エージェントでリモートデータベースを管理するには、Sybase Central でデータベースがエージェントに関連付けられている必要があります。このことは、リモートスキーマ名を指定することによって行われます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. 統合データベースをダブルクリックします。
2. 操作するエージェントをダブルクリックして、**[管理対象のリモートデータベースの追加]** をクリックします。
3. 使用可能なリモートデータベースが、リモートスキーマ名で表示されます。ドロップダウンリストから **[リモートスキーマ名]** を選択します。
4. リモートデータベースの接続文字列を入力し、**[OK]** をクリックします。

   接続文字列はクライアントデバイスで使用されます。すべての ODBC データソース、パス、ファイルは、デバイスに対して有効である必要があります。

結果

データベースが追加されます。

グループの追加

グループは、Mobile Link エージェントのコレクションです。

前提条件

グループを作成するには、1 つ以上のエージェントが定義されている必要があります。

♦ タスク

1. 統合データベースが稼働していることを確認します。
2. Mobile Link オブジェクトをダブルクリックします。
3. プロジェクト名をダブルクリックし、**[新規]** » **[グループ]** をクリックします。
4. **グループウィザード**の指示に従い、**[完了]** をクリックします。

結果

グループが作成されます。

エージェントの認証

エージェントデータベースを同期するとき、エージェントは Mobile Link 同期クライアントとして動作します。エージェントをサポートする同期スクリプトでは、*ml_ra_agent_16* スクリプトバージョンが使用されます。エージェント同期スクリプトは、統合データベース用に Mobile Link を設定したときに自動的にインストールされます。ただし、エージェントに対して認証スクリプトは提供されません。 「統合データベースの設定」3 ページを参照してください。

セキュアシステムにするには、少なくとも次のいずれかが定義されている必要があります。

- 「authenticate_user 接続イベント」
- 「authenticate_user_hashed 接続イベント」
- 「authenticate_parameters 接続イベント」

このことは、次の方法で実行できます。

- ml_add_connection_script ストアドプロシージャを呼び出す。次に例を示します。

```
ml_add_connection_script( 'ml_ra_agent_16', 'authenticate_user', '<DEFINE YOUR
AUTHENTICATION LOGIC>' )
```

- Sybase Central で **[接続文字列]** を使用する。ml_ra_agent_16 スクリプトバージョンまたは ml_global を使用できます。一般的に、アプリケーションデータの同期とエージェントには、どちらにも同じ認証を使用します。ml_global を使用することによって、両方に使用できる認証セットを 1 つ定義するのみとなります。認証には、この方法をおすすめします。 「ml_global スクリプトバージョン」253 ページを参照してください。

リモートタスク

リモートタスクは、リモートタスクの集中管理を実行するときの作業単位です。リモートタスクは、次の要素で構成されます。

●1 つ以上のトリガメカニズム

●オプションの条件

●順序付けされたコマンドのコレクション

●その他のプロパティ

タスクは、Sybase Central 用の Mobile Link 16 プラグインを使用して管理者が作成します。設計時には、プロジェクト内の管理者のコンピュータでローカルに保存されます。 タスクの構築に使用できるコマンドのリストについては、「コマンド」180 ページを参照してください。

一般的に、タスクは別のタスクの完了に依存しません。ただし、1 つのタスクでリモートデータベースに書き込みを行い、別のタスクの条件でその値を問い合わせるようにすることで、他のタスクに依存するタスクを作成することもできます。

管理者は、エージェントでタスクの受信準備ができると、タスクを展開します。展開済みタスクは、統合データベースにコピーされます。この時点ではタスクに 2 つのコピーが存在します (統合データベース内の展開済みタスクとプロジェクト内の設計時のタスク)。展開済みタスクは変更できません。ただし、タスクを作成するために使用された設計時のタスクは、変更して再展開できます (2 つ目の展開済みタスクを作成するため)。展開済みタスクは、キャンセル、起動 (SIRT)、再アクティブ化、新しい受信者への割り当てを行うことができます。

展開済みタスクは、1 つ以上のエージェントに割り当てて実行できます。エージェントに割り当てられたタスクは、エージェントにダウンロードされます。エージェントは、適切な時間にタスクを実行し、オプションとして実行結果を統合データベースにアップロードします。管理者は、Sybase Central で Mobile Link 16 プラグインを使用して結果を確認できます。

タスクには次の属性があります。

- **名前**　リモートタスクには 2 つの名前があります。1 つはプロジェクト内に保存されたタスクの設計時のバージョンを識別し、もう 1 つは展開済みタスクを識別します。通常、2 つの名前は同じです。

  タスクの設計時の名前はタスクを作成するときに割り当てられ、プロジェクト内のタスク間でユニークである必要があります。展開済みタスクの名前はタスクを展開するときに割り当てられ、統合データベースト内の展開済みタスク間でユニークである必要があります。

- **説明**　タスクの説明は、入力可能であり、タスクに関連付ける任意のテキストを含めることができます。この説明は、プロジェクトと統合データベース (タスクの展開後) に保存されますが、エージェントには送信されません。

- **トリガのメカニズム**　リモートタスクのトリガメカニズムによって、エージェントでのタスクの実行時期が決まります。1 つのタスクに複数のトリガメカニズムが存在することがあります。3 つのトリガメカニズムがサポートされています。

○ **[スケジュールに従う]**　タスクは、特定の時刻または特定の時間間隔でトリガされます。このオプションは、タスクに対して明示的に設定します。

○ **[エージェントで受信されたとき]**　タスクは、エージェントで受信されたときにトリガされ、1 回のみ実行されます。このオプションは、タスクに対して明示的に設定します。

○ **オンデマンド**　タスクは、サーバ起動リモートタスク (SIRT) と呼ばれるプロセスでサーバからのメッセージによっていつでもトリガできます。1 回のみ実行するように設定されていないすべてのタスクでは、「オンデマンド」でのトリガがサポートされています。

- **条件**　タスクを実行する前に、すべての条件が満たされている必要があります。タスクがトリガされた時点ですべての条件が満たされていない場合は、実行の試みが失敗したと見なされ、タスクを再度トリガして条件を再評価する必要があります。

- **リモートスキーマ名**　オプションとして、タスクをリモートスキーマ名に関連付けることができます。リモートデータベースにアクセスする必要があるタスクは、リモートスキーマ名に関連付ける必要があります。リモートスキーマ名は、タスクの実行時にエージェントからアクセスする必要があるデータベースを判断するために使用されます。create database、drop database、execute SQL、synchronize のいずれかのコマンドがタスクに含まれている場合は、タスクをリモートスキーマ名に関連付けてください。また、SQL 条件を使用してタスクが実行できるかどうかを判断する場合は、リモートタスクもリモートスキーマ名に関連付ける必要があります。

- **コマンド**　タスクには、必要な操作を実行する順序付けされたコマンドのセットが含まれています。コマンドの指定順序によって、タスク内でのコマンドの実行順序が定義されます。コマンドは互いに依存することがあるため、コマンドの順序には十分に注意してください。「コマンド」180 ページを参照してください。

- **[最大再試行回数]**　各コマンドには、コマンドが失敗した場合にタスクまたはコマンドを再試行するために使用できる「失敗時」のアクションがあります。このオプションを使用して、1 回の実行に有効な再試行回数を制限できます。

- **再試行間隔**　このオプションは、コマンドが失敗した後でコマンドまたはタスクを再試行するまでの待機時間を指定します。この遅延によって、コマンドまたはタスクが再試行される前に、失敗の原因となった一時的な状況 (データベーステーブルやファイルのロックなど) が解消されることがあります。再試行の遅延は、「短」期間であることが前提となります。再試行の間では、タスク条件は再評価されません。

- **最大実行時間**　タスクを実行したとき、管理者が意図したように動作しない場合があります。たとえば、OS 呼び出しがハングする、同期の実行速度が非常に遅い、データベースを使用する別の接続によって SQL 文がブロックされるなどの場合があります。タスクの最大実行時間を設定することによって、タスクの実行時間を制限できます。最大実行時間に達すると、タスクは終了します (タスクが実際にいつ終了するかは、その操作を中断できるかどうかによって異なります)。タスクのステータスがタイムアウトを反映して設定され、そのタスクは再度トリガされるまで再試行されません。タスク内のコマンドが失敗し、タスクまたはコマンドを再試行する必要がある場合は、遅延後に最大実行時間がリセットされます。したがって、最大実行時間は、タスクの実行または再試行ごとの値と見なされます。最大実行時間には、すべての再試行の合計時間およびプロンプトコマンドは含まれません。

- **スキーマ変更**　スキーマ変更タスクは、リモートデータベースのスキーマを変更します。タスクが成功すると、リモートデータベースのリモートスキーマ名も更新されます。スキーマ変更タスクは常に高優先度タスクであり、完了時にステータスを通知します。

- **高優先度**　高優先度タスクは、常に、エージェントによって受信された時点でトリガされ、現在実行中のタスクが完了するとすぐに実行されます。ただし、スケジュールに従って実行されない場合や、実行条件が存在しない場合もあります。高優先度とマークされているタスクは、他のタスクからの干渉を防ぐため、実行中の他のタスクが存在しないときにのみ実行されます。

- **ステータスレポート**　タスクのステータスレポートオプションを使用すると、タスクが正常に完了した場合と失敗した場合の両方において、統合データベースに結果を返すかどうか、返す場合はその時期を指定できます。有効なオプションは次のとおりです。

  - **[ステータスのみ送信]**　タスクの結果はレポートされません。ただし、成功または失敗したタスクの数に関する情報は、保持され、レポートされます。

  - **[エージェントがエージェントデータベースを次回に同期するときに結果を返す]**　タスクの結果とステータスが保持され、エージェントがエージェントデータベースを次回に同期するときにレポートされます。

  - **[タスクの実行が完了した時点で結果をすぐに返す]**　タスクの結果とステータスは、タスクが完了した時点ですぐにレポートされます。

- **ランダムな遅延時間**　指定のタスクによって、実行後に結果を Mobile Link サーバに結果が送信されたり、またはリモートデータベースがサーバと同期する結果となったときに膨大な数のリモートが同時にトリガされた場合に、タスクにランダムな遅延時間を設定することによって、設定可能な一定の期間、サーバの同期負荷を均一に分散させることができます。

  リモートタスクには、ランダムな遅延時間を設定できます。秒数で表される時間 *N* を設定することによって、各エージェントではランダムな秒数 (0 ～ N 秒) が生成され、その値に従って各タスクの実行を遅延させます。スケジュールされたタスクの場合には、最初のタスクが実行される前にランダムな遅延が生成され、それぞれの実行に使用されます。タスクはスケジュールされた時間に実行されますが、ランダムな遅延によってオフセットされます。それによって、タスクの実行時間のデルタ時間とスケジュールとの整合性が維持されます。

  ランダムな遅延時間では、スケジュールされたタスクの最小デルタ時間より大きな値を設定することはおすすめしません。タスクが「オンデマンド」タスクの場合、つまりサーバによって起動される場合には、タスクが起動されるたびに、ランダムな遅延が生成され実行の遅延に使用されます。タスクが「受信時に実行」されるタスクの場合には、最初にタスクが実行されたときにのみ、ランダムな遅延が生成され実行の遅延に使用されます。

参照

- 「リモートタスクのロジック」172 ページ
- 「リモートタスクの作成」172 ページ
- 「リモートタスクの編集」173 ページ
- 「リモートタスクの展開」174 ページ
- 「展開済みリモートタスク」175 ページ

リモートタスクのロジック

次に、リモートタスクの実行に使用されるロジックの概要を示します。

```
current_command = 1;
num_tries = 0;
EXECUTE_TASK:
loop {
    num_tries = num_tries + 1;
    EXECUTE_COMMANDS;
    if( task_success or task_abort ) break EXECUTE_TASK;
    if( task_retry and at maximum tries ) {
        break EXECUTE_TASK;
    } else {
        continue;
    }
}

EXECUTE_COMMANDS:
for each command starting at current_command {
    execute current_command;
    if( command failed ) {
        if( action on failure is "abort task" ) {
            break EXECUTE_COMMANDS, returning task_abort;
        } else if( action on failure is "continue" ) {
            // no action, continue at next command
        } else if( action on failure is "retry task" ) {
            current_command = 1;
            break EXECUTE_COMMANDS, returning task_retry;
        } else if( action on failure is "retry command" ) {
            // no change to current_command
            break EXECUTE_COMMANDS, returning task_retry;
        }
    }
    current_command = next command number;
}
return task_success;
```

最大実行時間は、タスクが開始されたときと、コマンドが失敗したためにコマンドまたはタスク全体を再試行したときにリセットされます。

コマンドが失敗したためにコマンドまたはタスク全体を再試行した場合、条件は再評価されず、新しいトリガも必要ありません。条件で参照されるプロパティがタスクの実行中に変化する可能性があるかどうか、条件が満たされずにコマンドを再試行した場合に望ましくない結果が発生するかどうかを考慮することが重要です。

リモートタスクの作成

タスクは、Mobile Link プロジェクトのコンテキスト内で定義、管理されます。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

管理者は、統合データベースに接続しないで、タスクの作成、変更、および削除を実行できます。未展開のタスクは開発中であると見なされ、Mobile Link プロジェクト内のみでローカルに存在します。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[リモートタスク]** をダブルクリックし、**[新規]** » **[リモートタスク]** をクリックします。
3. **リモートタスク作成ウィザード**の手順に従い、**[完了]** をクリックします。

> **注意**
> 同じスキーマを持つリモートデータベースのグループを識別する新しいリモートスキーマ名を作成する必要がある場合は、**リモートタスク作成ウィザード**の **[ようこそ]** ページで **[リモートスキーマ名を作成]** をクリックします。**リモートスキーマ名作成ウィザード**が表示され、**リモートタスク作成ウィザード**は開いたままになります。新しいリモートスキーマ名を作成すると、**リモートタスク作成ウィザード**の **[リモートスキーマ名]** ドロップダウンに表示されます。

4. 1 つ以上のコマンドをタスクに追加する手順に従います。 「リモートタスクへのコマンドの追加」186 ページを参照してください。

結果

タスクが作成されます。

次の手順

タスクが作成されたら、そのタスクを統合データベースに展開し、受信者 (エージェント) に割り当てることができます。 「リモートタスクの展開」174 ページを参照してください。

リモートタスクの編集

リモートタスクのプロパティは編集できます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[プロパティ]** をクリックします。
3. リモートタスクのプロパティを編集します。

4. リモートタスクのプロパティの編集が完了したら、**[適用]** をクリックします。

結果

リモートタスクのプロパティが更新されます。

次の手順

リモートタスクを展開し、エージェントに割り当てることができます。 「リモートタスクの展開」174 ページを参照してください。

リモートタスクの展開

タスクをシステムに追加する準備ができたら、タスクを展開する必要があります。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

リモートタスクを展開することは、タスクを統合データベースにコピーし、新しいコピーに名前を付けることを意味します。展開済みタスクの名前は、多くの場合、開発時に付けられたタスクの名前と同じです。

また、タスクを展開するとき、エージェントのリストにそのタスクを割り当てることもできます。エージェントはタスクの展開後に追加できるため、タスクを最初に展開する時点でエージェントを追加する必要はありません。このことは、タスクの展開後に新しいエージェントをシステムに追加する場合に便利です。

♦ タスク

リモートタスクを展開するには、少なくとも 1 つのコマンドがリモートタスクに含まれている必要があります。 コマンドの詳細については、「コマンド」180 ページ と 「リモートタスクへのコマンドの追加」186 ページを参照してください。

1. Mobile Link オブジェクトをダブルクリックします。
2. 展開するリモートタスクを右クリックし、**[展開]** をクリックします。
3. **リモートタスク展開ウィザード**の指示に従います。
4. **[完了]** をクリックします。

結果

リモートタスクが展開されます。

次の手順

展開済みタスクでは、キャンセル、起動、再アクティブ化、受信者の追加を実行できます。「展開済みリモートタスク」175 ページを参照してください。

リモートタスクのエクスポート

リモートタスクをファイルにエクスポートできます。

前提条件

リモートタスクが定義済みである必要があります。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. エクスポートするリモートタスクを右クリックし、**[エクスポート]** をクリックします。
3. ファイルの名前とロケーションを指定します。**[保存]** をクリックします。

結果

リモートプロファイルが、指定したファイルにエクスポートされます。

展開済みリモートタスク

展開済みであってもリモートタスクを操作できます。展開済みタスクでは、キャンセル、起動、再アクティブ化、受信者の追加を実行できます。

展開済みリモートタスクは Sybase Central の次の場所にあります。

- 個々のエージェントのレベルで展開済みリモートタスクを操作するには、Sybase Central の Mobile Link 16 プラグインの **[フォルダ]** のビューで、操作する統合データベースが展開されていること確認します。**[エージェント]** を展開し、操作したいエージェントをクリックして、右ウィンドウ枠の **[タスク]** タブをクリックします。そこには、選択したエージェントの展開済みのすべてのリモートタスクがリストされています。
- すべてのエージェントの展開済みリモートタスクを操作するには、左ウィンドウ枠の **[フォルダ]** ビューで、使用している Mobile Link プロジェクトが選択されていることを確認し、**[リモートタスク]** を右クリックして **[展開済みタスク]** をクリックします。すべてのエージェントの展開済みリモートタスクがリストされます。

参照

- 「展開済みリモートタスクのすべてのエージェントでのキャンセル」176 ページ
- 「展開済みリモートタスクの 1 つのエージェントでのキャンセル」176 ページ
- 「展開済みリモートタスクのすべてのエージェントでの起動」177 ページ
- 「展開済みリモートタスクの 1 つのエージェントでの起動」177 ページ
- 「展開済みリモートタスクの 1 つのエージェントでの再アクティブ化」178 ページ
- 「展開済みリモートタスクへの受信者の追加」178 ページ

展開済みリモートタスクのすべてのエージェントでのキャンセル

展開済みリモートタスクは、すでに必要でなくなった場合、またはリモートスキーマを更新して新しいバージョンのタスクを展開した場合には、キャンセルできます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[リモートタスク]** » **[展開済みタスク]** をクリックし、キャンセルする展開済みタスクを選択します。
3. キャンセルしたい展開済みタスクを右クリックし、**[すべての受信者についてキャンセル]** をクリックします。

結果

タスクがキャンセルされます。

展開済みリモートタスクの 1 つのエージェントでのキャンセル

展開済みリモートタスクは、すでに必要でなくなった場合、またはリモートスキーマを更新して新しいバージョンのタスクを展開した場合には、キャンセルできます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. 統合データベースをダブルクリックします。
2. **[エージェント]** をダブルクリックし、操作したいエージェントをクリックします。
3. 右ウィンドウ枠で、**[タスク]** タブをクリックします。
4. キャンセルしたい展開済みタスクを右クリックし、**[キャンセル]** をクリックします。

結果

結果

展開済みリモートタスクのすべてのエージェントでの起動

展開済みリモートタスクが開始されると、サーバはクライアント上のエージェントに、タスクがすぐに実行されることを通知します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[リモートタスク]** » **[展開済みタスク]** をクリックし、開始する展開済みタスクを選択します。
3. 起動したい展開済みタスクを右クリックし、**[すべての受信者が開始]** をクリックします。

結果

タスクがすべてのエージェントで開始されます。

展開済みリモートタスクの 1 つのエージェントでの起動

展開済みリモートタスクが開始されると、サーバはクライアント上のエージェントに、タスクがすぐに実行されることを通知します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. 統合データベースをダブルクリックします。
2. **[エージェント]** をダブルクリックし、操作したいエージェントをクリックします。
3. 右ウィンドウ枠で、**[タスク]** タブをクリックします。
4. 起動したい展開済みタスクを右クリックし、**[開始]** をクリックします。

結果

タスクが、選択したエージェントで開始されます。

展開済みリモートタスクの 1 つのエージェントでの再アクティブ化

一部の展開済みリモートタスクは再アクティブ化できます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. 統合データベースをダブルクリックします。
2. **[エージェント]** をダブルクリックし、操作したいエージェントをクリックします。
3. 右ウィンドウ枠で、**[タスク]** タブをクリックします。
4. 再アクティブ化したい展開済みタスクを右クリックし、**[再アクティブ化]** をクリックします。

結果

展開済みタスクが再アクティブ化されます。

展開済みリモートタスクへの受信者の追加

展開済みリモートタスクに受信者を追加できます。たとえば、タスクが展開された後に新しいエージェントが作成された場合には、展開済みタスクにエージェントを追加する必要があります。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[リモートタスク]** » **[展開済みタスク]** をクリックし、受信者を追加する展開済みタスクを選択します。
3. 受信者を追加したい展開済みタスクを右クリックし、**[受信者の追加]** をクリックします。
4. **[エージェント]** リストからエージェントを選択し、**[追加]** をクリックして **[受信者]** リストに追加するか、または **[すべて追加]** を選択して、すべてのエージェントを選択します。
5. **[OK]** をクリックします。

結果

選択したエージェントが受信者として追加されます。

サーバ起動リモートタスク (SIRT)

サーバ起動リモートタスクは、エージェントがタスクの実行指示通知をサーバから受信したときに実行される任意のリモートタスクです。管理者がリモートタスクの開始を選択すると、Mobile Link サーバは影響を受けるエージェントにメッセージを送信し、指定のタスクを実行するように指示します。特定の時間に実行するようにスケジュールされているタスクであっても、管理者がそのように選択した場合には、サーバ起動できます。

SIRT は、特定のエージェントで **[開始]** または展開済みタスクで **[すべての受信者が開始]** を選択することによって、Sybase Central を通じて起動できます。Mobile Link サーバで ml_ra_notify_task システムプロシージャを使用することによっても SIRT を起動できます。

参照

● 「ml_ra_notify_task システムプロシージャ」 689 ページ

リモートタスク Notifier (RTNotifier)

SIRT 要求を追跡するために、Mobile Link サーバには RTNotifier という Notifier が組み込まれています。RTNotifier は Mobile Link のシステムテーブルをチェックし、SIRT が起動されている場合には、Mobile Link エージェントが Mobile Link サーバをポーリングすると、RTNotifier がクライアントに適切なリモートタスク情報を送信し、リモートタスクが実行されます。

RTNotifier はデフォルトで実行されます。リモートタスクの集中管理を使用しない場合は、次のオプションを使用して RTNotifier を無効にできます。

RTNotifier オプションはオプションと値の組み合わせによって指定され、ml_property system テーブルに挿入されます。次の例では、RTNotifier をオフにする方法を示します。この例では、RTNotifier オプションは **enable** であり、値は **no** に設定されています。**SIRT** はコンポーネント名で、**RTNotifier(RTNotifier1)** は Notifier 名ですが、これらの 2 つのカラムは内部でのみ使用されるため、変更することはできません。

```
call ml_add_property( 'SIRT', 'RTNotifier(RTNotifier1)', 'enable', 'yes' );
```

次の表には、ml_property system テーブルに指定できる RTNotifier オプションがリストされています。

オプション	値	説明
autoset_poll_every	{ **yes** | **no** }	エージェントによる Mobile Link サーバへのリモートタスク要求のポーリングの頻度に基づいて、poll_every プロパティが自動的に調整されるかどうかを指定します。poll_every が調整される場合には、RTNotifer が update_poll_every 値に基づいて更新をチェックすると、更新内容が表示されます。
enable	{ **yes** | **no** }	Mobile Link サーバが起動されるときに、RTNotifier を有効にするかどうかを指定します。このプロパティは動的には更新できません。
poll_every	*秒数*	リモートタスク要求を使用してインメモリキャッシュを再設定するために、RTNotifier が request_cursor を実行する頻度を秒数で指定します。このプロパティの値が 2147483647 の場合は、RTNotifier は request_cursor を実行しません。
update_poll_every	*秒数*	RTNotifier がプロパティの更新をチェックする頻度を秒数で指定します。
request_cursor		このオプションは内部でのみ使用されます。統合データベースからリモートタスク要求を取り出すのに使用するクエリを指定します。

コマンド

コマンドは、特定のアクションを実行するタスク内の命令です。タスクには、複数のコマンドを含めることができ、コマンドには定められた順序があります。コマンドには、実行するアクション、入力パラメータ、コマンドが失敗した場合の処理に関する命令が含まれています。

copy file コマンド

リモートデバイスでファイルのコピーを作成します。

パラメータ

- **[元のファイル名]** コピーするファイルの名前。
- **[新しいファイル名]** 元のファイルのコピー先ファイルの名前。
- **[必要な場合は既存のファイルを上書き]** このオプションを使用すると、「新しいファイル名」パラメータで指定された名前がすでに存在していても、ファイルがコピーされます。
- **[読み取り専用属性を無視]** このパラメータは、「必要な場合は既存のファイルを上書き」 パラメータが使用されている場合にのみ使用できます。このオプションを使用すると、「新しいファイル名」 パラメータで指定された名前を持つファイルがすでに存在し、読み取り専用である場合にも、コピーが実行されます。

備考

ファイル名の指定は絶対パスまたは相対パスのどちらでもかまいません。ファイル名を相対パスで指定すると、ファイル名はエージェントの現在の作業ディレクトリからの相対パスになります。

create database コマンド

エージェントで管理されるリモートデバイスで新しいリモートデータベースを作成します。

パラメータ

- **[ファイル名]** 新しいデータベースのファイル名。通常、{db_location} マクロを使用して、データベースのパスを指定します (*{db_location}¥mydatabase.db* など)。エージェントで複数の SQL Anywhere データベースを管理する場合は、各データベースを別個のディレクトリに作成してください。*{db_location}* ディレクトリの異なるサブフォルダに各データベースを配置することをおすすめします。
- **[CHAR 照合]** 新しいデータベースで、CHAR、VARCHAR、LONG VARCHAR データ型の照合を指定します。
- **[NCHAR 照合]** SQL Anywhere の場合にのみ、新しいデータベースで、NCHAR、NVARCHAR、LONG NVARCHAR データ型の照合を指定します。

備考

このコマンドは、**[リモートデータベースが必要、またはリモートデータベースを作成]** としてマーク付けされたリモートタスクでのみ使用できます。

コマンドで作成されるデータベースのタイプ (Ultra Light または SQL Anywhere) は、リモートタスクに指定されたリモートスキーマ名によって決まります。指定されたファイル名で、リモートデバイスに存在しないディレクトリを使用すると、ディレクトリが作成されます。ファイル名の指定は絶対パスまたは相対パスのどちらでもかまいません。ファイル名を相対パスで指定すると、ファイル名はエージェントの現在の作業ディレクトリからの相対パスになります。このことは、Windows Mobile の SQL Anywhere ではサポートされていません。

CREATE DATABASE 文を使用してデスクトップコンピュータのデータベースを初期化し、これを後で Windows Mobile デバイスにコピーできます。

delete file コマンド

リモートデバイスでファイルを削除します。

パラメータ

- **[ファイル名]** リモートデバイスで削除するファイルの名前。
- **[読み取り専用属性を無視]** このオプションを選択すると、読み取り専用とマークされていてもファイルが削除されます。

備考

ファイル名の指定は絶対パスまたは相対パスのどちらでもかまいません。ファイル名を相対パスで指定すると、ファイル名はエージェントの現在の作業ディレクトリからの相対パスになります。

デフォルトでは、削除されるファイルが存在しない場合には、タスクはエラーになります。削除しようとしたファイルが存在しない場合でも、タスクがエラーにならないようにするには、Sybase Central の Mobile Link 16 プラグインのファイル削除コマンドの **[コマンド]** プロパティページで、**[ファイルが存在しなければエラーにする]** オプションをクリアします。

download file コマンド

サーバからリモートデバイスにファイルをダウンロードします。

パラメータ

- **[サーバファイル名]** サーバからリモートデバイスにダウンロードするファイルの名前。ファイル名は絶対名にできません。ファイル名は、Mobile Link サーバのダウンロードルートディレクトリからの相対パスになります。このディレクトリは、Mobile Link サーバで -ftr オプションを使用して指定されます。 「-ftr mlsrv16 オプション」62 ページを参照してください。
- **[リモートファイル名]** ファイルを保存するリモートデバイス上の場所を指定します。ファイル名には絶対パスまたは相対パスのどちらでも指定できます。ファイル名を相対パスで指定すると、ファイル名はエージェントの現在の作業ディレクトリからの相対パスになります。

drop database コマンド

管理対象のリモートデータベースを削除します。

パラメータ

なし

備考

このコマンドは、[リモートデータベースが必要、またはリモートデータベースを作成] としてマーク付けされたリモートタスクでのみ使用できます。削除されるデータベースは、リモートタスクに対して指定したリモートスキーマ名に関連付けられたデータベースです。

drop database コマンドを含むタスクに関連付けられたリモートスキーマ名の接続文字列には、DBF パラメータが必要です。

drop database コマンドを実行する場合、削除されるデータベースは停止している必要があります。

データベースが削除されると、リモートデータベース内のすべてのデータが失われます。

Windows Mobile では、データベースの削除はサポートされていません。

参照

- 「Ultra Light DBF 接続パラメータ」『Ultra Light データベース管理とリファレンス』
- 「DatabaseFile (DBF) 接続パラメータ」『SQL Anywhere サーバ データベース管理』

SQL コマンドの実行

リモートデータベースに対して SQL を実行します。

パラメータ

- **[SQL]**　実行される SQL。GO を含む SQL 文は、別の行に単独で指定します。BEGIN と END 文で囲まれた SQL 文では、BEGIN ～ END ブロック内に GO を指定しないでください。次に、文を区切る GO の正しい使用例を示します。

```
SELECT * FROM systable
GO
CREATE PROCEDURE p1()
BEGIN
   CREATE TABLE t1( pk INTEGER PRIMARY KEY );
   INSERT INTO t1 VALUES( 5 );
   COMMIT;
END
GO
SELECT * FROM sysprocedure
```

備考

このコマンドは、[リモートデータベースが必要、またはリモートデータベースを作成] としてマーク付けされたリモートタスクでのみ使用できます。SQL は、リモートタスクに指定したリモートスキーマ名に関連付けられたデータベース対して実行されます。

SQL を実行するとき、エージェントでは文を COMMIT しません。実行中の SQL に COMMIT が存在しない場合、文はロールバックされます。このことは、SQL が INSERT、UPDATE、DELETE 文であるか、COMMIT を明示的に発生させない他の文である場合に重要です。

コマンドのステータスと結果には、実行された SQL の結果が保存されます。DDL 文は結果を返しません。INSERT/UPDATE/DELETE 文の影響を受けるロー数は、単一の値として 1 行で返され

ます。SELECT 文の結果は *.csv* 形式で返されます。最初のローはカラム見出しになります。複数の文の結果はすべて、1 つの大きい結果に追加されます。

prompt コマンド

リモートデバイスでメッセージボックスを表示します。

パラメータ

- **[メッセージ]**　メッセージボックスに表示するメッセージ。

備考

リモートデバイスでは、**[OK]** をクリックして閉じるまでメッセージが表示されます。

タスクで prompt コマンドの後に他のコマンドが続く場合、メッセージを閉じるまで他のコマンドは実行されません。プロンプトが表示されてからユーザが **[OK]** をクリックするまでの時間は、タスクの実行時間としては計算されません。

rename file コマンド

リモートデバイスでファイル名を変更します。

パラメータ

- **[元のファイル名]**　名前を変更するファイルの現在の名前。
- **[新しいファイル名]**　名前を変更した後のファイル名。
- **[必要な場合は既存のファイルを上書き]**　このオプションを選択すると、新しいファイル名を持つファイルがすでに存在していても、ファイル名が変更されます。
- **[読み取り専用属性を無視]**　このオプションは **[必要な場合は既存のファイルを上書き]** を選択した場合にのみ選択できます。このオプションを選択すると、新しいファイル名を持つファイルがすでに存在し、読み取り専用である場合にも、ファイル名の変更が実行されます。

備考

ファイル名の指定は絶対パスまたは相対パスのどちらでもかまいません。ファイル名を相対パスで指定すると、ファイル名はエージェントの現在の作業ディレクトリからの相対パスになります。

run program コマンド

リモートデバイスでプログラムを実行します。

パラメータ

- **[コマンドライン]**　実行されるコマンドライン。

備考

タスクの実行は、プログラムの実行が完了するまで続行されません。実行されたプログラムの終了コードが 0 の場合は、コマンドが成功したと見なされます。

synchronize コマンド

リモートデータベースを同期します。

パラメータ

- **[同期プロファイル]**　同期に使用するオプションが含まれた同期プロファイルで、リモートデータベース内に定義されているものを指定します。

- **[その他のオプション]**　同期に使用する追加オプションを指定します。その他のオプションと同期プロファイルの両方でオプションが指定された場合、その他のオプションの設定が、同期プロファイルの設定よりも優先されます。このオプションは空白でもかまいません。

備考

このコマンドは、[リモートデータベースが必要、またはリモートデータベースを作成] としてマーク付けされたリモートタスクでのみ使用できます。同期されるデータベースは、リモートタスクに対して指定したリモートスキーマ名に関連付けられたデータベースです。

upload file コマンド

リモートデバイスからサーバにファイルをアップロードします。

パラメータ

- **[リモートファイル名]**　リモートデバイスからサーバにアップロードするファイルの名前。ファイル名には絶対パスまたは相対パスのどちらでも指定できます。ファイル名を相対パスで指定すると、ファイル名はエージェントの現在の作業ディレクトリからの相対パスになります。

- **[サーバファイル名]**　ファイルを保存するサーバ上の場所を指定します。このファイル名は絶対パスで指定できず、複数の円記号 (¥) を含めることもできません。ファイル名は、Mobile Link サーバのアップロードルートディレクトリからの相対パスになります。このディレクトリは、Mobile Link サーバで -ftru オプションを使用して指定されます。

  コマンドを実行する各エージェントがサーバ上の異なる場所にファイルをアップロードできるようにするため、サーバのファイル名にマクロを使用することをおすすめします。このようにしない場合、複数のエージェントが互いのファイルを上書きする問題が発生することがあります。各エージェントのファイルは、ディレクトリ名にエージェント ID を使用した異なるディレクトリに配置することをおすすめします。このことを行うには、{agent_id} マクロを使用します。たとえば、*myuploadfile.txt* というファイルをアップロードする場合は、アップロード先ファイル名を {agent_id}¥myuploadfile.txt に設定します。「パラメータの変数」187 ページを参照してください。

コマンドの使用法

次の方法の 1 つを使用して、コマンドをタスクに追加します。

- 左ウィンドウ枠の **[フォルダ]** ビューでタスクを右クリックし、**[コマンドを追加]** をクリックします。
- 左ウィンドウ枠の **[フォルダ]** 表示でタスクを選択し、**[コマンドを追加]** ツールバーボタンをクリックします。
- タスクを作成すると、右ウィンドウ枠にコマンドが自動的に表示されます。[Tab] キーを押して、パラメータ間を移動します。[Tab] キーを押し続けると、新しいコマンドがタスクに自動的に追加されます。
- 既存のコマンドを右クリックし、**[コマンドを追加]** をクリックします。
- 既存のコマンドの下にあるホワイトスペースを右クリックし、**[コマンドを追加]** をクリックします。
- 左ウィンドウ枠の **[フォルダ]** 表示でタスクを選択します。**[ファイル]** » **[コマンドを追加]** をクリックします。

「リモートタスクへのコマンドの追加」186 ページを参照してください。

リモートタスクへのコマンドの追加

リモートタスクは、少なくとも 1 つのコマンドがリモートタスクに含まれるまで展開できません。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Mobile Link オブジェクトをダブルクリックします。
2. **[リモートタスク]** をダブルクリックし、操作するリモートタスクを右クリックして、**[コマンドを追加]** をクリックします。**[コマンド]** ウィンドウ枠が、右ウィンドウ枠に表示されます。
3. **[コマンドタイプ]** ドロップダウンリストから、必要なコマンドのタイプを選択します。 使用可能なコマンドの詳細については、「コマンド」180 ページを参照してください。
4. 選択したコマンドに、適切なパラメータを入力します。
5. **[失敗時]** ドロップダウンリストから、次のいずれかのオプションを選択して、コマンドが失敗した場合の処理方法を指定します。
   - **[タスクをアボート]**　タスクを実行する現在の試みが終了し、失敗のマークが付きます。
   - **[続行]**　次のコマンドに移動して、タスクの実行を続行します。

- **[コマンドを再試行]**　失敗したコマンドから、タスクが再試行されます。タスクの最大再試行回数に達すると、コマンドは再試行されません。
- **[タスクを再起動]**　最初のコマンドからタスクが再試行されます。タスクの最大試行回数に達すると、タスクは再試行されません。

結果

結果

次の手順

タスクは展開できます。 「リモートタスクの展開」174 ページを参照してください。

パラメータの変数

次のマクロを使用して、リモートデバイス間で異なる情報にアクセスできます。これらの値は、リモートタスク条件、リモートテタスク内のコマンドのパラメータ、接続文字列で使用できます。

変数	置換
{agent_db}	エージェントデータベースファイルのフルパスとファイル名。このファイルを必要に応じてデバイスからアップロードして、問題の診断に役立てることができます。
{agent_id}	エージェント ID。
{agent_log}	エージェントログファイルのフルパスとファイル名。このファイルを必要に応じてデバイスからアップロードして、問題の診断に役立てることができます。エージェントがログファイルなしで実行されしている場合、この変数は空の文字列になります。
{battery_level}	リモートデバイスのバッテリー残量。範囲は 0 ～ 100 です。
{db_location}	mlagent -db オプションで指定される、エージェントのリモートデータベースディレクトリ。
{is_on_ac_power}	リモートデバイスで交流電力を使用しているかどうかを示します。1 はデバイスのプラグが差し込まれていることを示し、0 はデバイスがバッテリー電力で稼働していることを示します。
{is_online}	Mobile Link サーバの IP アドレスへのルートが存在するネットワークに、クライアントデバイスが接続されている場合にのみ、この変数は 1 (true) と評価されます。ホストコンピュータがオフラインであっても、変数は 1 (true) と評価されます。

変数	置換
{ml_password}	エージェントデータベースを同期するときにエージェントで使用される Mobile Link パスワード (pwd)。
{ml_stream}	Mobile Link サーバに接続するためのプロトコルパラメータ (HTTP{host=Sybase.com;port=9376} など)。
{ml_username}	エージェントデータベースを同期するときにエージェントで使用される Mobile Link ユーザ名 (uid)。
{network_conn_name}	Mobile Link サーバと通信を行うエージェントによって使用されるネットワーク接続の名前を評価します。Mobile Link サーバと通信を行うエージェントによって使用が可能なネットワーク接続がない場合には、この変数は **?**.と評価されます。
{remote_id}	このタスクに関連付けられたリモートデータベースのリモート ID。この値は、**[リモートデータベースを必要とするもの、またはリモートデータベースを作成するもの]** としてマーク付けされたリモートタスクでのみ意味があります。
{rows_to_upload}	Ultra Light データベース専用です。完全な同期が行われた場合に、アップロードされるリモートデータベースのロー数。

ステータス

エージェントによってタスクが実行されると、ステータス情報をレポートしないというマークがタスクに付けられていないかぎり、その実行に関するステータス情報がエージェントデータベースに保存されます。エージェントデータベース内のステータス情報は、エージェントデータベースが同期されるたびに、サーバにアップロードされます。

ステータス情報には、Sybase Central の Mobile Link 16 プラグインを使用してアクセスできます。左ウィンドウ枠の **[フォルダ]** 表示でタスクを選択し、右ウィンドウ枠で **[受信者]** タブまたは **[結果]**タブのいずれかを表示すると、特定の展開済みタスクのステータスが表示されます。左ウィンドウ枠の **[フォルダ]** 表示でエージェントを選択し、右ウィンドウ枠で **[タスク]** タブを表示すると、特定のエージェントに割り当てられたすべてのタスクのステータスが表示されます。

ステータス情報は、次のストアドプロシージャからでも返されます。

- 「ml_ra_get_agent_events システムプロシージャ」
- 「ml_ra_get_agent_ids システムプロシージャ」
- 「ml_ra_get_agent_properties システムプロシージャ」
- 「ml_ra_get_latest_event_id システムプロシージャ」
- 「ml_ra_get_orphan_taskdbs システムプロシージャ」
- 「ml_ra_get_remote_ids システムプロシージャ」
- 「ml_ra_get_task_results システムプロシージャ」
- 「ml_ra_get_task_status システムプロシージャ」

システムプロシージャ

Sybase Central のリモートタスクの管理機能に加え、統合データベースの Mobile Link システムプロシージャを使用して管理タスクを自動化することもできます。ml_ra_cancel_notification システムプロシージャと修復プロシージャを除き、システムプロシージャで可能なすべての作業は、Sybase Central の Mobile Link 16 プラグインでも実行できます。ただし、次のタスクは Sybase Central の Mobile Link 16 プラグインを使用してのみ実行できます。

●新しいタスクの作成

●新しいリモートスキーマ名の作成

●エージェント、リモート、リモートスキーマ名、タスクへの説明の追加

新しいすべての Mobile Link テーブル、システムプロシージャ、エージェントスクリプトバージョンは、プレフィクス **ml_ra_** で始まります。

次に、リモートデータベースの集中管理に使用されるシステムプロシージャのリストを示します。

- 「ml_ra_add_agent_id システムプロシージャ」
- 「ml_ra_assign_task システムプロシージャ」
- 「ml_ra_cancel_notification システムプロシージャ」
- 「ml_ra_cancel_task_instance システムプロシージャ」
- 「ml_ra_clone_agent_properties システムプロシージャ」
- 「ml_ra_delete_agent_id システムプロシージャ」
- 「ml_ra_delete_events_before システムプロシージャ」
- 「ml_ra_delete_remote_id システムプロシージャ」
- 「ml_ra_delete_task システムプロシージャ」
- 「ml_ra_get_agent_events システムプロシージャ」
- 「ml_ra_get_agent_ids システムプロシージャ」
- 「ml_ra_get_agent_properties システムプロシージャ」
- 「ml_ra_get_latest_event_id システムプロシージャ」
- 「ml_ra_get_orphan_taskdbs システムプロシージャ」
- 「ml_ra_get_remote_ids システムプロシージャ」
- 「ml_ra_get_task_results システムプロシージャ」
- 「ml_ra_get_task_status システムプロシージャ」
- 「ml_ra_manage_remote_db システムプロシージャ」
- 「ml_ra_notify_agent_sync システムプロシージャ」
- 「ml_ra_reassign_taskdb システムプロシージャ」
- 「ml_ra_set_agent_property システムプロシージャ」
- 「ml_ra_unmanage_remote_db システムプロシージャ」

展開と構成

ここでは、展開と構成について説明します。 Mobile Link エージェントの展開に必要なファイルのリストについては、「SQL Anywhere Mobile Link クライアントの配備」776 ページと「Ultra Light Mobile Link クライアントの配備」 779 ページを参照してください。

エージェントの展開に関する考慮事項

Mobile Link 同期システムでリモートデータベースを正しく管理するには、いくつかの技術的なポイントについて考慮する必要があります。

- Mobile Link エージェントは、Windows デバイスと Windows Mobile デバイスでのみ動作します。それ以外のプラットフォームのリモートデータベースは、現在、Mobile Link エージェント経由で管理できません。
- Ultra Light リモートデータベースの管理には、Ultra Light エンジンを使用する必要があります。結果、これらのリモートデータベースにアクセスするアプリケーションも、Ultra Light エンジン経由でアクセスする必要があります。Ultra Light のインプロセスバージョンを使用しようとすると、「ファイル使用中」エラーになります。 「Windows Mobile 用の Ultra Light データ管理コンポーネント」『Ultra Light データベース管理とリファレンス』を参照してください。
- 集中管理は、Mobile Link エージェントがデバイスで実行されている場合にのみ可能です。一般的に、エージェントは常にデバイスで実行されていると想定されます。*mlastop.exe* を使用してエージェントを停止することはできますが、集中管理を再び有効にするには、エージェントを再起動する必要があります。 「Mobile Link エージェント停止ユーティリティ」163 ページを参照してください。

Mobile Link エージェントを使用した Windows Mobile での Ultra Light アプリケーションと Ultra Light データベースの展開

Mobile Link エージェントを使用して Windows Mobile デバイスで Ultra Light を展開するためのさまざまなメカニズムがあります。

SQL Anywhere **Deployment ウィザード**を使用して、Windows Mobile デバイスでの SQL Anywhere の展開に使用できる *.CAB* キャビネットファイルを構築できます。ただし、**Deployment ウィザード**では、ユーザアプリケーションとデータベースの展開の作成がサポートされていません。

Deployment ウィザードを完了すると、*.INF* ファイルが作成されます。*.INF* ファイルは、ファイルのターゲットロケーション、ショートカット、*.CAB* ファイルに含まれるレジストリ設定を記述するいくつかのセクションで構成されます。エージェントでユーザアプリケーションとデータベースをインストールするロジックが含まれるように、この *.INF* ファイルを変更できます。

SQL Anywhere for Windows Mobile Deployment ウィザードを使用した展開

SQL Anywhere for Windows Mobile Deployment ウィザードを使用して、リモートデータベースの集中管理に必要なファイルを展開できます。

SQL Anywhere for Windows Mobile Deployment ウィザードには、**[SQL Anywhere のリモートデータベースの管理]** または **[Ultra Light のリモートデータベースの管理]** のオプションがあります。

エージェントの構成 (Windows デスクトップの場合)

Windows コンピュータにエージェントを構成するには、インストールプログラム (アプリケーションをインストールするのと同じプログラム) からエージェントをインストールし、エージェントを構成、検証、起動するコマンドを実行する方法があります。インストールプログラムでは、Mobile Link アプリケーションの識別や認証パラメータを要求するプロンプトが表示されることがあります。これらのパラメータを使用し、コマンドラインで mlagent -cr を使用して、エージェントを構成できます。

エージェントが構成された後、インストールプログラムで **mlagent** *-pi* ... を実行して、認証パラメータが有効であることを検証できます。このコマンドを実行して認証を正しく検証するには、Mobile Link サーバへの接続が必要です。

最後に、インストールプロセスの最終ステップとしてエージェントを起動します。これで、エージェントの受信準備が整い、ターゲットデバイスでタスクを実行できるようになります。エージェントのプロセスのリターンコードは、エージェントの実行に関する情報を提供するために使用できます。たとえば、mlagent で Mobile Link サーバの ping に失敗すると、*mlagent.exe* から返されるリターンコードは、構成済み mlagent オプションを使用したエージェントデータベースの同期結果である SQLCODE になります。

エージェントの構成 (Windows Mobile の場合)

Windows Mobile での Mobile Link エージェントの構成方法は、ユーザのアプリケーションがデバイスにどのようにインストールされているかによって異なります。

Mobile Link エージェントを構成して実行する 1 つの方法として、Mobile Link の識別や認証パラメータを要求するプロンプトをユーザアプリケーションから表示し、その起動プロセスの一環として Mobile Link エージェントの構成と起動を行う方法があります。

- アプリケーションではエージェントの実行を試行できます。エージェントが正しく構成されていない場合は、mlagent 実行プログラムからエラーコードが返されます。
- このコードが返された場合は、エージェントを設定モードで実行して設定してから、通常モードでエージェントを再実行します。 「クライアントデバイスでの Mobile Link エージェント」157 ページを参照してください。

Mobile Link プロファイラ

Mobile Link プロファイラは、同期のパフォーマンスに関する詳細情報を提供する Mobile Link 管理ツールです。このツールを使用することにより、ボトルネックを分析し、パフォーマンスを最大限に高めることができます。基本的なパフォーマンス情報については SQL Anywhere モニタを使用できます。同期について、イベントレベルを掘り下げて、詳細を取得するには Mobile Link プロファイラを使用します。

プロファイリングセッションで得られた同期データは、データディレクトリ内にデフォルトのファイル名、ユーザ、パスワードで作成されたプロファイリングデータベースファイルに保存されます。**[オプション]** ウィンドウの **[一般]** タブで、別のプロファイリングデータベースを指定できます。

Mobile Link プロファイラを起動して Mobile Link サーバに接続すると、そのプロファイリングセッションで発生するすべての同期に関する統計情報の収集が開始されます。プロファイリングセッションを終了するか、Mobile Link サーバを停止するまで、Mobile Link プロファイラはデータを収集し続けます。Mobile Link プロファイラのインタフェースでは、表形式またはグラフィカル形式でデータを表示できます。

同期に関するさまざまな情報を、Mobile Link プロファイラの出力で確認できます。たとえば、エラーが発生した同期またはイベントや、指定したその他の条件に一致する同期またはイベントをすばやく特定できます。期間の異なる同期においてほぼ同時に終了したフェーズがあるかど

うかをチェックすると、(同期は前のフェーズが完了するのを待ってから処理を続行するため) 可能性のある同期スクリプトの競合を特定できます。Mobile Link サーバで検出されたブロックの対象であるイベントを特定することもできます。

運用システムへの展開前に開発環境でパフォーマンスをテストするには、主にプロファイラを使用することをおすすめします。

SQL Anywhere モニタ

SQL Anywhere モニタは、SQL Anywhere データベースや Mobile Link サーバの正常性や可用性に関する情報を示す、ブラウザベースの管理ツールです。システム全体の正常性や可用性を評価するときや、全体的な同期統計情報を分析するときに便利です。SQL Anywhere モニタは個々の同期に関する情報は提供しません。タイミングなどの同期別の統計を含む、個々の同期に関する詳細情報を取得するには、Mobile Link プロファイラを使用します。

SQL Anywhere モニタの詳細については、「SQL Anywhere モニタ」『SQL Anywhere サーバ データベース管理』を参照してください。

Mobile Link プロファイラ (管理ツール) の起動

Mobile Link サーバごとに、Mobile Link プロファイラのインスタンスを複数実行できます。ただし、1 つの Mobile Link サーバで実行する Mobile Link プロファイラのインスタンスは 1 つのみとすることをおすすめします。

前提条件

新しいプロファイリングセッションでは、統合データベースと Mobile Link サーバを起動します (起動していない場合)。

内容と備考

注意
Mobile Link プロファイラのバージョンは、使用する Mobile Link サーバのバージョンと一致している必要があります。

♦ タスク

- **[スタート]** » **[プログラム]** » **[SQL Anywhere 16]** » **[管理ツール]** » **[Mobile Link プロファイラ]** をクリックします。

結果

Mobile Link プロファイラが起動します。

次の手順

プロファイリングセッションを開始して、データの収集を始めます。「プロファイリングセッションの開始」193 ページを参照してください。

コマンドラインでの Mobile Link プロファイラ (mlprof)

コマンドラインオプションを使用すると、Mobile Link プロファイラを開いたり、起動時に Mobile Link サーバへ接続することができます。これは、テストセッションのプロファイリングを自動的に無人で行う場合に役に立ちます。

次の構文を使用します。

mlprof [*options*]

オプション	説明
-c	プロファイリングセッションの終了時に Mobile Link プロファイラを閉じます。
-p *password*	Mobile Link ユーザのパスワード。
-r	プロファイリングデータベースを再作成します。 以前のプロファイリングセッションをすべて削除する場合、またはプロファイリングデータベーススキーマに問題がある場合は、このオプションを使用します。
-u *ml_username*	Mobile Link ユーザ名。コマンドラインから開始するプロファイリングセッションの開始には、このオプションが必要となります。
-x {**tcpip**|**tls**|**http**|**https**} [(*keyword=value*;...)]	Mobile Link サーバに接続するためのネットワークプロトコルおよびパラメータ。keyword=value のペアは、ホスト、ポート、追加のネットワークパラメータに指定できます。コマンドラインから開始するプロファイリングセッションの開始には、このオプションが必要となります。 「-x mlsrv16 オプション」92 ページを参照してください。

mlprof -? と入力すると、mlprof の構文が表示されます。

プロファイリングセッションの開始

Mobile Link プロファイラのセッションを開始すると、データの収集が始まり、データはプロファイリングデータベースに保存されます。

前提条件

統合データベースと Mobile Link サーバを起動します (起動していない場合)。

♦ **タスク**

1. Mobile Link プロファイラで、**[ファイル]** » **[プロファイリングセッションの開始]** をクリックします。データの収集が始まります。

2. Mobile Link プロファイラ接続は、Mobile Link サーバへの同期接続と同じように開始されます。すべての Mobile Link プロファイラセッションに対して、スクリプトバージョンが **for_ML_Monitor_only** に設定されます。

   [Mobile Link サーバへの接続] ウィンドウで次の情報を入力してください。

   - **[ユーザ]**　接続用の Mobile Link ユーザ名を入力します。ユーザ名を入力する必要がありますが、Mobile Link サーバを -zu+ で起動した場合は、認識されていない Mobile Link ユーザ名が同期時に ml_user に自動的に追加されるため、どの Mobile Link ユーザを入力してもかまいません。
   - **[パスワード]**　接続用のパスワードを入力します。Mobile Link ユーザに対する正しいパスワードを指定してください。Mobile Link ユーザにパスワードがない場合、このフィールドは空白のままとします。
   - **[ホスト]**　Mobile Link サーバが稼働しているコンピュータのネットワーク名または IP アドレス。デフォルトでは、ホストは Mobile Link プロファイラが稼働しているコンピュータです。Mobile Link サーバが Mobile Link プロファイラと同じコンピュータで稼働している場合は、**localhost** を使用できます。
   - **[プロトコル]**　Mobile Link サーバが同期要求に使用しているのと同じネットワークプロトコルに設定してください。
   - **[ポート]**　Mobile Link サーバが同期要求に使用しているのと同じネットワークポートに設定してください。
   - **[暗号化]**　プロトコルとして HTTPS または TLS を選択すると、このボックスが有効になります。ドロップダウンリストから暗号タイプを選択します。

     HTTPS および TLS を使用するには、Mobile Link プロファイラが稼働しているコンピュータに Mobile Link クライアント側データストリーム暗号化をインストールする必要があります。 セキュリティの詳細については、「Mobile Link サーバ／クライアント通信の暗号化」『SQL Anywhere サーバ データベース管理』を参照してください。

     > **注意**
     > 別途ライセンスが必要な必須コンポーネント
     >
     > FIPS 認定の暗号化には別のライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。
     >
     > 「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

   - **[信頼できる証明書ファイル]**　プロトコルに [HTTPS] または [TLS] を選択した場合は、Mobile Link サーバとのセキュア接続に使用する、信頼できる証明書ファイルの名前を指定します。Windows プラットフォームの場合、信頼できる証明書ファイルを指定しないと、信頼できる証明書ストアが使用されます。Windows 以外のプラットフォームの場合、セキュア接続には信頼できる証明書ファイルが指定されていることが必要です。

- **[追加のプロトコルオプション]**　このフィールドにはオプションのネットワークパラメータを指定します。指定できる値は、接続ストリームのタイプによって異なります。複数のパラメータは、セミコロンで区切る必要があります。

  Mobile Link クライアントネットワークプロトコルのすべての有効なオプションがサポートされています。ただし、このウィンドウですでに設定されているホスト、ポート、信頼できる証明書などのオプションは除きます。

  「Mobile Link クライアントネットワークプロトコルオプション」『Mobile Link クライアント管理』を参照してください。

3. 同期を開始します。
4. **[一時停止]** ボタンをクリックして、チャートと使用率グラフの自動スクロールを停止します。

結果

Mobile Link プロファイラはデータの収集を開始し、収集されたプロファイリングデータが表示されます。

プロファイリングセッションの終了

Mobile Link プロファイラセッションを終了すると、データの収集は終了しますが、Mobile Link プロファイラは引き続き稼動します。このため、データを表示したり新しいプロファイリングセッションを表示したりすることができます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. **[ファイル]** » **[プロファイリングセッションの終了]** をクリックします。これにより、データの収集が停止され、プロファイラが Mobile Link サーバから切断されます。

   Mobile Link サーバをシャットダウンするか、Mobile Link プロファイラを終了して、データの収集を停止することもできます。

2. Mobile Link プロファイラを終了する準備ができたら、**[ファイル]** » **[閉じる]** をクリックします。

結果

Mobile Link プロファイラがプロファイリングデータの収集を停止します。

以前のプロファイリングセッションを開くまたは削除する

[Mobile Link プロファイラセッションを開く] ウィンドウで、開くか、または削除する以前の Mobile Link プロファイラセッションを選択します。

前提条件

プロファイリングデータベースに以前のプロファイラセッションが存在する必要があります。

♦ タスク

1. Mobile Link プロファイラで、**[ファイル]** » **[セッションを開く]** をクリックします。プロファイリングデータベースにある以前のプロファイリングセッションが表示されます。

2. 開くか、削除するプロファイリングセッションを選択します。

3. **[OK]** をクリックして選択したセッションを開くか、**[削除]** をクリックして、選択したセッションをプロファイリングデータベースから削除します。

結果

[OK] をクリックすると、選択したセッションのデータが表示されます。

[削除] をクリックすると、選択したセッションがプロファイリングデータベースから削除され、リストから消えます。

次の手順

以前のプロファイラセッションを開いた場合は、データを確認できます。

プロファイリングデータベース

Mobile Link プロファイラは、はじめて起動したときに、デフォルトのファイル名、ユーザ、パスワードでプロファイリングデータベースを作成します。プロファイリングデータベースは、ドキュメントディレクトリの *MLProfiler16* フォルダに格納されます。プロファイリングデータベースに対してデフォルトのユーザおよびパスワードを使用しない場合は、データベースのロケーションを、使用する既存の SQL Anywhere データベースに変更するか、そのデータベースをデフォルトのロケーションにコピーします。Mobile Link プロファイラを次に起動するときに、そのデータベースのユーザ ID およびパスワードを入力して、そのデータベースをプロファイリングデータベースとすることができます。接続に成功すると、接続情報が保存されます。

プロファイリングデータベースに接続された後で、互換性のないスキーマが検出された場合、プロファイリングデータベーススキーマを再作成するかどうかを問われ、Mobile Link プロファイラは閉じます。プロファイルデータベースの再作成を選択した場合は、プロファイラを再起動すると、スキーマが再作成され、さらに以前のプロファイリングセッションデータが削除されます。Mobile Link プロファイラを再起動する前にデータのバックアップを行わないと、データが失われる可能性があります。

プロファイルデータベースの名前とロケーションを決定するには、**[オプション]** ウィンドウの **[一般]** ページに移動します。

Mobile Link プロファイラのインタフェース

Mobile Link プロファイラには次のウィンドウ枠があります。

- **[詳細テーブル]**　**[詳細テーブル]** は一番上のウィンドウ枠です (有効な場合)。これは、デフォルトでは各同期フェーズの合計所要時間と、同期の各部分に要した時間を示すスプレッドシートです。

  「[詳細テーブル] ウィンドウ枠」197 ページを参照してください。

- **[使用率グラフ]**　**[使用率グラフ]** は 2 番目のウィンドウ枠です (有効な場合)。このウィンドウ枠には、各フェーズの同期数がグラフィックで表示されます。**[使用率グラフ]** ウィンドウ枠と **[チャート]** ウィンドウ枠で同じ横目盛りが使用されます。**[チャート]** ウィンドウ枠の下の目盛りは時間を表します。下の **[概要]** ウィンドウ枠でデータをドラッグして選択するか、**[表示]** » **[移動]** をクリックして、[使用率グラフ] ウィンドウ枠に表示されているデータを選択できます。

  「[使用率グラフ] ウィンドウ枠」199 ページを参照してください。

- **[チャート]**　**[チャート]** は、3 番目のウィンドウ枠で、常に表示されます。このウィンドウ枠には、同期が同期フェーズごとに色付けされてグラフィックで表示されます。このウィンドウ枠の下の目盛りは時間を表します。下の **[概要]** ウィンドウ枠でデータをドラッグして選択するか、**[表示]** » **[移動]** をクリックして、[チャート] ウィンドウ枠に表示されているデータを選択できます。

  「[チャート] ウィンドウ枠」200 ページを参照してください。

- **[概要]**　**[概要]** は一番下のウィンドウ枠です (有効な場合)。ここには、セッション内のすべての同期の概要が表示されます。このウィンドウ枠には**マーキーツール**と呼ばれるボックスの枠があり、**[チャート]** および **[使用率グラフ]** ウィンドウ枠に表示される領域が示され、この領域は選択できます。

  「[概要] ウィンドウ枠」201 ページを参照してください。

また、表示をカスタマイズするための **[オプション]** ウィンドウと、表示できるプロパティシートがあります。 「[オプション] ウィンドウ」203 ページを参照してください。

[詳細テーブル] ウィンドウ枠

[詳細テーブル] は、フェーズ時間など、同期に関する情報を提供します。時間はすべて Mobile Link サーバによって測定されます。対応するスクリプトが定義されていなくても、フェーズ時間によっては 0 ではない場合があります。

[ツール] » **[オプション]** をクリックし、次に **[テーブル]** タブを開くと、**[詳細テーブル]** ウィンドウ枠に表示されるカラムを選択できます。 使用できる統計の詳細については、「Mobile Link の同期統計のプロパティ」208 ページを参照してください。

デフォルトでは、次のカラムが表示されます。

- **number**　各同期を識別します。この番号は、Mobile Link プロファイラではなく Mobile Link サーバによって割り当てられるので、どの Mobile Link プロファイラセッションでも、1 から始まるとは限らず、続き番号になるとも限りません。この番号は、Mobile Link サーバの警告、エラー、ログに表示される同期番号と同じです。同じ番号を **[同期プロパティ]** ウィンドウで見ることができます。 「同期プロパティ」203 ページを参照してください。

- **remote_id**　リモートデータベースのユニークな識別子。 「リモート ID」『Mobile Link クライアント管理』を参照してください。

- **user**　Mobile Link 同期ユーザ名。 「Mobile Link ユーザ」『Mobile Link クライアント管理』を参照してください。

- **version**　同期スクリプトのバージョン。

  「スクリプトバージョン」252 ページを参照してください。

- **start_time**　Mobile Link サーバが同期を開始した日時 (これは、クライアントが同期を要求した時点よりも遅い場合があります)。

- **duration**　同期の合計時間 (秒)。

次の時間はすべて秒数単位です。

- **[sync_request] フェーズ**　リモートデータベースと Mobile Link サーバの間のネットワーク接続を確立してから、アップロードストリームの最初のバイトを受信するまでの時間。-sm を -nc よりも小さい値に設定した場合以外はこの時間はわずかです。-sm を -nc よりも小さい値に設定した場合は、-sm で指定したアクティブな同期の最大数よりも同期の数が多かった場合に同期が一時停止した時間がこの時間に含まれる可能性があります。

- **[receive_upload] フェーズ**　Mobile Link サーバでアップロードストリームの最初のバイトが受信されてから、リモートデータベースからのアップロードストリームが完全に受信されるまでの時間。アップロードストリームには、テーブルの定義や、アップロードされているリモートデータベースのローが含まれるので、ダウンロード専用同期でもこの時間が長くなる場合があります。時間は、アップロードストリームのサイズと、転送用のネットワーク帯域幅によって異なります。

- **[get_db_worker] フェーズ**　空いているデータベースワーカスレッドを取得するために必要な時間。

- **[connect] フェーズ**　新しいデータベース接続が必要な場合に、データベースワーカスレッドでデータベース接続を確立するために必要な時間。たとえば、エラーの後や、スクリプトのバージョンが変更された場合に新しい接続が必要になります。

- **[authenticate_user] フェーズ**　認証を必要とする同期設定の場合に、Mobile Link が同期要求、ユーザ名、パスワードを検証するのに要する時間。これは、ユーザ認証トランザクション (認証の開始から begin_synchronization イベントの直前まで) の長さです。

- **[begin_sync] フェーズ**　begin_synchronization スクリプトが実行されている場合、実行に要する時間。

- **[apply_upload] フェーズ**　アップロードを統合データベースに適用するのに要する時間。これは、begin_upload スクリプトから end_upload スクリプトまでの時間です。
- **[prepare_for_download] フェーズ**　prepare_for_download スクリプトが実行されている場合、実行に要する時間。
- **[fetch_download] フェーズ**　統合データベースからダウンロードするローをフェッチするために必要な時間。これは、begin_download スクリプトから end_download スクリプトまでの時間です。
- **[end_sync] フェーズ**　end_synchronization スクリプトが実行されている場合、実行に要する時間。
- **[send_download] フェーズ**　ダウンロードストリームをリモートデータベースに送信するために必要な時間。
- **[wait_for_download_ack] フェーズ**　ダウンロード確認が有効な場合、これには、ダウンロードがリモートデータベースに適用され、リモートデータベースからダウンロード確認が送信されるのを待つ時間が含まれます。
- **[get_db_worker_for_download_ack] フェーズ**　ダウンロード確認が有効な場合、これには、ダウンロード確認を受信してから、データベースワーカスレッドが空くのを待つ時間が含まれます。
- **[connect_for_download_ack] フェーズ**　ダウンロード確認が有効な場合、これには、新しいデータベース接続が必要なときに、データベースワーカスレッドでデータベース接続を確立するために必要な時間が含まれます。
- **[nonblocking_download_ack] フェーズ**　ダウンロード確認が有効な場合、これには、publication_nonblocking_download_ack connection 接続イベントと nonblocking_download_ack 接続イベントに必要な時間が含まれます。

テーブルを特定のカラムでソートするには、カラムの見出しをクリックします。Mobile Link モニタに表示される新しいデータがある場合は、ソートされて追加されます。

[ビュー] メニューで **[詳細テーブル]** オプションのチェックを外すことで、**[詳細テーブル]** ウィンドウ枠を閉じることができます。

[使用率グラフ] ウィンドウ枠

[使用率グラフ] は、上から 2 番目のウィンドウ枠です。このウィンドウ枠には、時間グラフの各フェーズにおける同期数が表示されます。

このウィンドウ枠に表示されるデータの詳細については、「[使用率グラフ]の動作方法」200 ページを参照してください。

[使用率グラフ] ウィンドウ枠では、**[チャート]** ウィンドウ枠と同じ水平スクロールバー、水平時間ラベル、水平ズームレベルが使用されます。**[使用率グラフ]** ウィンドウ枠と **[チャート]** ウィンドウ枠で同じ時間帯が上下に並びます。

[使用率グラフ] と **[チャート]** に表示される時間範囲を選択するには、次の 2 つの方法があります。

- **[ビュー]** メニューの **[移動]** をクリックします。
- **[概要]** ウィンドウ枠で、**マーキーツール**を動かします。**マーキーツール**とは、**[概要]** ウィンドウ枠に表示される小さなボックスです。

[使用率グラフ] の領域内をダブルクリックすると、**[サンプルプロパティ]** ウィンドウが開き、表示されているサンプル間隔の詳細が表示されます。サンプル間隔は約 1 秒です。

[使用率グラフ] ウィンドウ枠でマウスをドラッグして、サンプルの範囲に関するデータを表示します。**[サンプル範囲プロパティ]** ウィンドウが表示されます。

参照

- 「[使用率グラフ]の動作方法」200 ページ
- 「マーキーツール」202 ページ
- 「サンプルプロパティ」203 ページ

[使用率グラフ]の動作方法

[使用率グラフ] をカスタマイズするには、**[ツール]** » **[オプション]** をクリックし、**[グラフ]** タブをクリックします。このタブには **[使用率グラフ]** の時間が色分けして表示されます。また、ここでグラフをカスタマイズすることもできます。

フェーズカウンタ

各プロパティでは、現在そのフェーズにある同期の数が示されます。

アンチエイリアス処理

カスタマイズできるものの 1 つは、アンチエイリアス処理です。アンチエイリアス処理により、グラフの見栄えがよくなりますが、描画に時間がかかることもあります。

[チャート] ウィンドウ枠

[チャート] ウィンドウ枠には、**[詳細テーブル]** のデフォルトカラムと同じ情報がグラフィックで表示されます。**[チャート]** 内のバーは、各同期で要した時間を表し、バーの各部分は同期のフェーズを表します。

データの表示

[詳細テーブル] で同期をクリックすると、その同期が選択されます。

同期をダブルクリックすると、**[同期プロパティ]** ウィンドウが開きます。 「同期プロパティ」203 ページを参照してください。

リモート ID 別またはコンパクトにデータをグループ化

リモート ID 別にデータをグループ化するには、**[ビュー]** » **[リモート ID]** をクリックします。

また、データをコンパクトモードで表示すると、すべてのアクティブな同期ができるだけ少ない行数で表示されます。**[ビュー]** » **[コンパクトビュー]** をクリックします。**[コンパクトビュー]** では、ロー番号は無意味になります。

データの拡大

[チャート] ウィンドウ枠と **[使用率グラフ]** に表示されているデータを選択するには、次の 4 つの方法があります。

- **ズームオプション**　**[ビュー]** メニューのズームオプションとツールバーのズームボタンを使用して、ズームインやズームアウトができます。同期によって使用可能な領域を埋めるには、**[選択範囲にズーム]** を使用します。
- **スクロールバー**　**[チャート]** ウィンドウ枠の下のスクロールバーをクリックしてスライドさせます。
- **[移動] ウィンドウ**　このウィンドウを開くには、**[ビュー]** » **[移動]** を選択します。

  [開始日時] では、**[チャート]** ウィンドウ枠に表示するデータの開始時刻を指定できます。この設定を変更する場合は、少なくとも日付／時刻の年月日を指定してください。

  [チャートの範囲] では、表示する期間を指定できます。チャート範囲は、ミリ秒、秒、分、時間、または日数で指定できます。チャート範囲がデータの詳細度を決定します。時間が短かければ、より詳細に表示できます。
- **マーキーツール**　**[概要]** ウィンドウ枠で、ドラッグして**マーキーツール**を変更します。**マーキーツール**とは、**[概要]** ウィンドウ枠に表示されるボックスです。 「マーキーツール」202 ページを参照してください。

時間軸

[チャート] ウィンドウ枠の下には、期間を示す目盛りがあります。時間のフォーマットは、表示される時間の間隔に応じて自動的に再調整されます。目盛り上にカーソルを置くと、いつでも完全な日付／時刻を表示できます。

デフォルトの色スキーム

[オプション] ウィンドウ (**[ツール]** メニューから開く) では、**[チャート]** ウィンドウ枠の色を表示または設定できます。**[チャート]** ウィンドウ枠の各フェーズには、アップロードにライムグリーン、ダウンロードにコーラルレッド、開始と終了に青の色スキームがデフォルトで使用されます。各フェーズの以前の部分は暗い影で表します。

色の設定については、「[オプション] ウィンドウ」203 ページを参照してください。

[概要] ウィンドウ枠

[概要] ウィンドウ枠には、Mobile Link プロファイラのセッション全体の概要が表示されます。セッション間は**マーキーツール**を使用してナビゲートできます。マーキーツールは、**[概要]** ウィンドウ枠内のボックスです。

デフォルトでは、アクティブ同期、ブロックされた同期、完了同期、失敗同期は、ウォッチを介して色分けされて表示されます。色を設定するには、Mobile Link プロファイラを開き、**[ツー**

ル] » **[オプション]** をクリックし、**[概要]** タブをクリックします。または **[ツール]** » **[ウォッチマネージャ]** をクリックし、**[編集]** をクリックして対応するウォッチを編集します。

次の項を参照してください。

- 「[オプション] ウィンドウ」203 ページ
- 「統計のカスタマイズ」204 ページ

[概要] ウィンドウ枠を閉じるには、**[ビュー]** メニューで **[概要]** のチェックをオフにします。

[概要] ウィンドウ枠は、[Mobile Link プロファイラ] ウィンドウの他の部分から切り離すこともできます。**[オプション]** ウィンドウで、**[概要]** タブをクリックし、**[概要のウィンドウをメインウィンドウに組み込む]** チェックボックスをオフにします。

マーキーツール

マーキーツールとは、**[概要]** ウィンドウ枠に表示される小さなボックスです。**マーキーツール**を使用すると、さまざまなデータを表示したり、データをさまざまな詳細度で表示できます。このボックス内の領域は、[チャート] ウィンドウ枠と [使用率グラフ] ウィンドウ枠に表示されます。**マーキーツール**は次のように使用できます。

- **[概要]** ウィンドウ枠をクリックして、**マーキーツール**を動かします。これにより、[チャート] および [使用率グラフ] に表示されるデータの開始時刻が変わります。
- **[概要]** ウィンドウ枠内でドラッグして**マーキーツール**を描画し直すことで、**マーキーツール**の場所とサイズを変更します。これにより、データの開始時刻と範囲が変わります。マーキーボックスを小さくすると、[チャート] に表示されるデータの間隔が短くなり、より詳細なデータが表示されます。

マーキーツールの色の変更

マーキーツールの色は変更することができます。

前提条件

この作業を実行するための前提条件はありません。

♦ マーキーツールの色の変更

1. **[ツール]** » **[オプション]** をクリックします。
2. **[概要]** タブをクリックします。
3. **[マーキー]** フィールドで新しい色を選択します。
4. **[OK]** をクリックします。

結果

マーキーツールの色が変更されます。

[オプション] ウィンドウ

[オプション] ウィンドウでは、**[チャート]** ウィンドウ枠、**[使用率グラフ]** ウィンドウ枠、**[概要]** ウィンドウ枠のグラフィック表示の色とパターンなど、多くの設定を指定できます。

[オプション] ウィンドウ**[一般]** タブでは、プロファイリングデータを格納するのに使用するプロファイリングデータベースの変更と、プロファイリングデータベースの再作成 (以前のプロファイリングセッションはすべて削除される) を行うことができます。

[オプション] ウィンドウを開くには、Mobile Link プロファイラを開き、**[ツール]** » **[オプション]** をクリックします。

セッションプロパティ

[セッションプロパティ] ウィンドウには、プロファイリングセッションに関する統計が表示されます。現在のプロファイリングセッションのプロパティ値が提供されます。**[セッションプロパティ]** ウィンドウを開くには、**[ファイル]** » **[プロパティ]** をクリックします。

サンプルプロパティ

[サンプルプロパティ] ウィンドウを開くには、**[グラフ使用率]** ウィンドウ枠内で検証する期間の部分をクリックします。**[サンプルプロパティ]** ウィンドウには、時間間隔に関する詳細情報が表示されます。各時間間隔は約 1 秒間です。サンプルが受信された順番を反映できるように、サンプルには Mobile Link プロファイラによって番号が付けられます。

[サンプルプロパティ] ウィンドウには、次の 3 つのタブがあります。

- **[一般]** サンプル取得時に同期で行われていた内容の概要が表示されます。
- **[フェーズ]** サンプル取得時に同期があったフェーズの数を表示します。
- **[イベント]** 同期中に実行されているイベントスクリプトに関する情報を表示します。

グラフの外観をカスタマイズして、プロパティを非表示にすることはできますが、**[サンプルプロパティ]** ウィンドウにはすべてのプロパティが表示されます。フェーズを非表示にした場合、そのフェーズは **[サンプルプロパティ]** ウィンドウの **[フェーズ]** タブで **[非表示]** として示されます。そうでない場合は、色が表示されます。

[使用率グラフ] でドラッグして複数のサンプルを選択した場合は、**[サンプル範囲プロパティ]** ウィンドウに複数のサンプルの情報が表示されます。

[サンプル範囲プロパティ] ウィンドウには、**[サンプルプロパティ]** ウィンドウの場合と同じタブが表示されます。ただし、範囲の平均値と最大値が表示されます。

同期プロパティ

[詳細テーブル] ウィンドウ枠または **[チャート]** ウィンドウ枠内で同期をダブルクリックすると、その同期のプロパティが表示されます。

すべてのテーブルの統計 (その同期のすべてのテーブルの合計) か、個々のテーブルに関する統計を表示できます。ドロップダウンリストには、その同期に関係したテーブルのリストが表示されます。

[同期] ページには警告およびエラー、またはそのいずれかが表示され、**[イベント]** ページにはイベントスクリプトが呼び出された頻度とイベントスクリプトの所要時間が表示されます。同期がブロックされたことを Mobile Link サーバで検出した場合は、**[ブロック]** ページが表示されます。

[同期プロパティ] ウィンドウのいずれかのページに表示される数量の詳細については、**[ヘルプ]** をクリックしてください。

[同期プロパティ] ウィンドウの統計値の説明については、「Mobile Link の同期統計のプロパティ」208 ページを参照してください。

統計のカスタマイズ

ウォッチマネージャを使用すると、指定した基準を満たす同期を視覚的に区別できます。たとえば、大規模な同期、長時間の同期、長時間を要する小規模な同期、または警告を受信した同期を強調表示できます。

ウォッチマネージャを開くには、Mobile Link プロファイラを開き、**[ツール]** » **[ウォッチマネージャ]** をクリックします。

ウォッチマネージャの左ウィンドウ枠には、使用可能なすべてのウォッチのリストが表示されます。右ウィンドウ枠には、アクティブなウォッチのリストが表示されます。ウォッチをアクティブリストに追加したり、アクティブリストから削除するには、左ウィンドウ枠でウォッチを選択し、該当のボタンをクリックします。

事前に定義されたウォッチが 4 つ (**[アクティブ]**、**[ブロック]**、**[完了]**、**[失敗]**) あります。事前に定義されたウォッチを編集して、表示方法を変更できます。また、右ウィンドウ枠からこれらのウォッチを削除すると、非アクティブにできます。

[チャート] には、ウォッチの条件を満たしていない同期は表示されません。すべてのウォッチを無効に (**[現在のウォッチ]** リストから削除) すると、**[チャート]** ウィンドウ枠または **[概要]** ウィンドウ枠に同期が表示されなくなります。

右ウィンドウ枠でのウォッチの順序は重要です。リストの上にあるウォッチから先に処理されます。**[上へ移動]** ボタンと **[下へ移動]** ボタンを使用して、右ウィンドウ枠でのウォッチの順序を編成できます。

事前に定義されたウォッチを使用して、別のウォッチを作成できます。ウォッチの条件を編集するには、古い条件を削除してから新しいウォッチ条件を追加します。

新しい Mobile Link プロファイラが同じ Mobile Link サーバに接続すると、すでに接続されている Mobile Link プロファイラには短時間同期として示されます。Mobile Link プロファイラ同期のバージョン名は for_ML_Monitor_only になります。次の条件を満たしたウォッチを有効にするだけで、この Mobile Link プロファイラ同期を非表示にすることができます。

- **[プロパティ]**　**[バージョン]** に設定
- **[演算子]**　**[等しくない]** に設定
- **[値]**　**for_ML_Monitor_only** に設定

新しいウォッチの作成

定義したウォッチ基準を満たす同期が表示されるように、ウォッチを追加します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. **ウォッチマネージャ**で **[新規]** をクリックします。
2. **[名前]** ボックスにウォッチの名前を入力します。
3. **プロパティ**、**比較演算子**、**値**を選択します。

   プロパティの完全なリストについては、「Mobile Link の同期統計のプロパティ」208 ページを参照してください。
4. **[追加]** をクリックします (条件を保存するために **[追加]** をクリックしてください)。
5. 必要に応じて、別の**プロパティ**、**演算子**、**値**を選択して、**[追加]** をクリックします。
6. **[チャート]** ウィンドウ枠で、ウォッチの**チャートパターン**を選択します
7. **[概要]** ウィンドウ枠で、ウォッチの**概要の色**を選択します
8. **[OK]** をクリックします。

結果

新しいウォッチが作成されます。

プロファイリングデータベースの使用

Sybase Central では、定義済みのビューを使用して、プロファイリングデータベース内のデータを確認し分析することができます。

前提条件

少なくとも 1 つのプロファイリングセッションを実行している必要があります。

プロファイリングデータベースの名前とロケーションを知っている必要があります。この情報を決定するには、**[オプション]** ウィンドウの **[一般]** ページに進みます。

内容と備考

このタスクは、デフォルトのプロファイリングデータベースを使用することが前提となります。

Interactive SQL を使用してプロファイラのビューを操作することもできます。

♦ タスク

1. **SQL Anywhere 16** プラグインを使用して、データベースに接続するには、次のオプションを使用します。
   - **[ユーザ ID]**　[ユーザ ID] に **mlprofiler** と入力します。
   - **[パスワード]**　パスワードに **sql** と入力します。
   - **[アクション]**　**[このコンピュータのデータベースを起動して接続]** を選択します。
   - **[データベースファイル]**　プロファイリングデータベースのパス情報を入力するか、**[参照]** をクリックしてファイルを選択します。デフォルトのデータベースファイルは *mlprofiler.db* です。このファイルは、Documents フォルダ内の *MLProfiler16* と呼ばれるフォルダにあります。
   - **[サーバ名]**　**MLProfilerDB** と入力します。
   - **[開始行]**　データベースページや他のデータベースサーバ情報をキャッシュするために初期メモリを設定するには、次の内容を入力します。

     ```
     dbeng16.exe -c 1g
     ```

2. **[接続]** をクリックします。
3. mlprofiler データベースを展開し、**[ビュー]** をダブルクリックすると、Mobile Link プロファイラのビューの一覧が表示されます。

4. ビューを選択します。次のビューを使用できます。
   - **category_samples** (カテゴリサンプリングデータの基本ビュー)
   - **data_event_statistics**
   - **data_event_times**
   - **data_phase_statistics**
   - **data_phase_times**
   - **event_samples** (イベントサンプリングデータの基本ビュー)
   - **event_statistics**
   - **event_times**
   - **event_total_times**
   - **phase_samples** (フェーズサンプリングデータの基本ビュー)
   - **phase_statistics**
   - **phase_times**
   - **server_cumulative_samples** (サーバ関連の累積サンプリングデータの基本ビュー)
   - **server_snapshot_samples** (サーバ関連の非累積サンプリングデータの基本ビュー)
   - **sync_as_csv** (以前の Mobile Link モニタの *.csv* ファイルフォーマットのようなビュー)
   - **sync_blocked**
   - **sync_statistics**
   - **sync_times**
   - **syncs** (同期の基本ビュー)

   右側にある [SQL] ウィンドウ枠の上部には、選択したビューを説明するコメントが表示されます。

   注意
   サーバ関連のサンプリングデータは、Mobile Link 用 SQL Anywhere モニタでも使用できるメトリックのためのものです。

結果

プロファイリングデータベースからのデータは、**[データ]** ページのビューごとに表示されます。

次の手順

- プロファイリングデータの確認
- クエリで以下のビューを使用します。

例

次のサンプルクエリは、2 番目のセッションのすべての同期にほとんどの時間を費やしたイベントスクリプトを示します。

```
select * from event_total_times where session_id = 2 order by 1 desc
```

次のサンプルクエリは、すべてのセッションにおける同期完了の最大速度を示します。

```
select
   max( "Successful syncs/s" ) as "Max syncs/s",
   session_id
from server_throughput_samples
```

```
group by session_id
order by 1 desc, 2
```

Mobile Link の同期統計のプロパティ

次に、Mobile Link プロファイラで使用できる同期の統計プロパティのリストを示します。これらの統計は、**[新規ウォッチ]** ウィンドウ、**[詳細テーブル]** ウィンドウ枠、**[同期プロパティ]** ウィンドウで表示できます。**[同期プロパティ]** では、プロパティ名にアンダースコアが付いていません。

[新規ウォッチ] ウィンドウの詳細については、「統計のカスタマイズ」204 ページを参照してください。

[詳細テーブル] の詳細については、「[詳細テーブル] ウィンドウ枠」197 ページを参照してください。

[同期プロパティ] ウィンドウの詳細については、「同期プロパティ」203 ページを参照してください。

同期の統計情報

Mobile Link 統計プロパティは、次の同期情報を返します。

プロパティ	説明
active	同期が進行中の場合は true。
apply_upload フェーズ	アップロードしたデータを統合データベースに適用するために必要な時間。
authenticate_user フェーズ	ユーザ認証の実行にかかった時間の合計。authenticate_* イベントの実行時間も含まれます。
begin_sync フェーズ	begin_synchronization イベント時間の合計。
client	Mobile Link クライアントのタイプと完全なクライアントバージョン。たとえば、Dbmlsync 16.0.0.xxxx となります。
completed	同期が正常に完了した場合は true。
conflicted_updates	競合を引き起こした更新ローの数。対処する競合解決スクリプトが正常に呼び出された場合のみ、ローは含まれます。

プロパティ	説明
connect フェーズ	新しいデータベース接続が必要な場合に、データベースワーカスレッドでデータベース接続を確立するために必要な時間。たとえば、エラーの後や、スクリプトのバージョンが変更された場合に新しい接続が必要になります。
connect_for_download_ack フェーズ	ダウンロード確認用に新しいデータベース接続が必要な場合に、データベースワーカスレッドでデータベース接続を確立するために必要な時間。
connection_retries	Mobile Link サーバが統合データベースへの接続をリトライした回数。
download	このプロパティは、ダウンロードコマンドに含まれる同期を示します。
download_ack	なし、または非ブロッキングに指定できます。
download_bytes	Mobile Link サーバ内で、ダウンロードを格納するため、リモートデータベースに送信するため (暗号化または圧縮の前に) に使用されるメモリの容量。
download_deleted_rows	Mobile Link サーバによって (download_delete_cursor スクリプトを使用して) 統合データベースからフェッチされたロー削除の数。
download_errors	ダウンロード中に発生したエラーの数。
download_fetched_rows	Mobile Link サーバによって (download_cursor スクリプトを使用して) 統合データベースからフェッチされたローの数。
download_filtered_rows	クライアントがアップロードしたローと一致したため、Mobile Link クライアントにダウンロードされなかったフェッチ済みローの数。
download_warnings	ダウンロード中に発生した警告の数。
duration	Mobile Link サーバが測定した同期時間の合計。
end_sync フェーズ	end_synchronization イベント時間の合計。
fetch_download フェーズ	統合データベースからダウンロードするローをフェッチしてダウンロードストリームを作成するために必要な時間。

プロパティ	説明
get_db_worker フェーズ	空いているデータベースワーカスレッドを取得するために必要な時間。
get_db_worker_for_download_ack	ダウンロード確認を受信してから、データベースワーカスレッドが空くのを待つ時間。
has_blocked	Mobile Link サーバでブロックが検出された場合は true
ignored_deletes	handle_error または handle_odbc_error が定義されており、1000 が返された場合、または指定のテーブルに対して upload_delete スクリプトが定義されていない場合において、upload_delete スクリプトを呼び出したときにエラーを発生させたアップロード削除ローの数。
ignored_inserts	無視されたアップロード挿入ローの合計数。無視される理由は次のとおりです。1) 通常モードで upload_insert スクリプトがない。2) Mobile Link サーバが対応するスクリプトを呼び出し中にエラーが発生し、handle_error または handle_odbc_error イベントが 1000 を返した。
ignored_updates	競合を引き起こしたが、競合解決スクリプトが正常に呼び出されていなかったり、upload_update スクリプトが定義されていなかったりした、アップロード更新ローの数。
nonblocking_download_ack フェーズ	publication_nonblocking_download_ack 接続と nonblocking_download_ack 接続のイベントに必要な時間。
number	同期番号。
prepare_for_download フェーズ	prepare_for_download イベント時間の合計。
receive_upload フェーズ	アップロードを受信するためのフェーズ時間。
remote_id	リモートデータベースをユニークに識別するリモート ID。
send_download フェーズ	ダウンロードストリームをリモートデータベースに送信するために必要な時間。時間は、ダウンロードストリームのサイズと、転送用のネットワーク帯域幅によって異なります。アップロード専用同期の場合、ダウンロードストリームは単なるアップロード確認です。
server	Mobile Link サーバ名または host:port。
start_time	同期開始の日付／時刻 (ISO-8601 拡張フォーマット)。

プロパティ	説明
sync_deadlocks	同期で検出された統合データベース内のデッドロックの数。
sync_errors	同期で発生したエラーの合計数。
sync_request フェーズ	リモートデータベースと Mobile Link サーバの間のネットワーク接続を確立してから、アップロードストリームの最初のバイトを受信するまでの時間。
sync_tables	同期に関係したクライアントテーブルの数。
sync_warnings	同期で発生した警告の数。
upload	アップロードコマンドに含まれる同期を示します。
upload_bytes	アップロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。同期のサーバメモリへの影響が反映されます。
upload_deadlocks	アップロード中に検出された統合データベース内のデッドロック数。
upload_deleted_rows	統合データベースから正常に削除されたローの数。
upload_errors	アップロード中に発生したエラーの数。
upload_inserted_rows	統合データベースに正常に挿入されたローの数。
upload_updated_rows	統合データベースで正常に更新されたローの数。
upload_warnings	アップロード中に発生した警告の数。
user	Mobile Link ユーザ名。
version	同期バージョンの名前。
wait_for_download_ack	ダウンロードがリモートデータベースに適用され、リモートデータベースからダウンロード確認が送信されるのを待つ時間。

Relay Server

Relay Server は、Web サーバを通じて通信するモバイルデバイスとバックエンドサーバ間で、安全で負荷分散された通信を実現します。サポートされているバックエンドサーバには、Mobile Link、Unwired Server、Afaria、Mobile Office があります。Relay Server には次の機能があります。

- バックエンドサーバと通信するモバイルデバイス用の共通の通信アーキテクチャ。
- バックエンドサーバ用の負荷分散されたフォールトトレラントな環境を可能にするメカニズム。
- 企業の既存のファイアウォール設定やポリシーと簡単に統合可能で、バックエンドサーバとモバイルデバイス間の通信を容易にする方法。

Relay Server の詳細については、Relay Server を参照してください。

Mobile Link ファイルベースのダウンロード

ファイルベースのダウンロードは、SQL Anywhere リモートデータベースにデータをダウンロードする、もう 1 つの方法です。ダウンロード内容はファイルとして配布することが可能であり、同期の変更をオフラインで配布できます。このため、ファイルを一度作成すれば、多数のリモートデータベースにこのファイルを配布できます。

ファイルベースのダウンロードでは、ダウンロードした同期の変更内容をファイルに保存し、ファイルを転送可能なあらゆる方法を使用して SQL Anywhere リモートデータベースに転送できます。次に例を示します。

●衛星マルチキャストでデータをブロードキャストする。

●Sybase Afaria を使用して更新を適用する。

●ファイルを電子メールまたは FTP でユーザに送信する。

ファイルを送信するユーザを選択します。ファイルベースのダウンロードでは、競合の検出と解決を含め、同期の整合性は完全に保護されます。このファイルにサードパーティの暗号化を適用することにより、ファイルの安全性を保証できます。

使用する場合

ファイルベースのダウンロードは、統合データベースで大量のデータが変更されるが、リモートデータベースではデータの更新の頻度が低いか、更新がまったく行われない場合に便利です。たとえば、価格リスト、製品リスト、コードのテーブルなどです。

ファイルベースのダウンロードは、ダウンロードされたデータがリモートデータベースで頻繁に更新される場合、またはアップロード専用の同期を頻繁に実行している場合には適していません。このような状況では、ダウンロードファイルの適用時に実行される整合性のチェックが原因で、リモートサイトはダウンロードファイルを適用できないことがあります。

現時点では、ファイルベースのダウンロードは SQL Anywhere リモートデータベースでのみ使用可能です。

ダウンロード専用のパブリケーション

通常、ファイルベースのダウンロードにはダウンロード専用のパブリケーションを使用する必要があります。通常のパブリケーションは、ファイルベースのダウンロードの実行と同じパブリケーションを使用してアップロードを実行する必要がある場合にかぎり使用してください。

通常のパブリケーションを使用する場合は、リモートデータベースを更新する手段として、ファイルベースのダウンロードのみを使用することはできません。この場合でも、完全な同期またはアップロード専用の同期を定期的に実行する必要があります。完全な同期またはアップロード専用の同期は、ログオフセットを進めたり、ログファイルを保守したりするために必要です。こうしないと、ログファイルのサイズが大きくなり、同期処理に時間がかかるようになります。また、完全な同期ではエラーからのリカバリが必要になることもあります。

参照

- 「ダウンロード専用のパブリケーション」『Mobile Link クライアント管理』

ファイルベースのダウンロードの設定

以下の手順では、ファイルベースのダウンロードを設定するのに必要なタスクの概要について説明します。この手順では、Mobile Link 同期がすでに設定されているものとします。

1. ファイル定義データベースを作成します。

   「ファイル定義データベース」213 ページを参照してください。

2. 統合データベースで、新しいスクリプトバージョンを使用してスクリプトを作成します。

   「統合データベースでの変更」214 ページを参照してください。

3. ダウンロードファイルを作成します。

   「ダウンロードファイルの作成」215 ページを参照してください。

4. ダウンロードファイルを適用します。

   「新しいリモートの同期」215 ページを参照してください。

クイックスタートのためのその他の資料

- 「ファイルベースのダウンロード例」

ファイル定義データベース

ファイルベースのダウンロードを設定するには、「ファイル定義データベース」を作成します。これは、リモートデータベースと同じ同期テーブルと同期パブリケーションを持つ SQL Anywhere データベースです。配置する場所は、どこでもかまいません。このデータベースには、データまたはステータスの情報はありません。バックアップまたは保守を行う必要がなく、必要に応じて削除したり、再作成できます。

ファイル定義データベースには、次のものが含まれている必要があります。

- リモートデータベースと同じパブリケーション、そのパブリケーションで使用されるテーブルとカラム、これらのテーブルとカラムの外部キー関係と制約、これらの外部キー関係に必要なテーブル。
- ダウンロードファイルを適用するリモートデータベースのグループを識別する Mobile Link ユーザ名。リモートデータベースのグループを識別するには、このグループの Mobile Link ユーザ名を同期スクリプトで使用します。

統合データベースでの変更

統合データベースでは、ファイルベースのダウンロード用のスクリプトバージョンを新規に作成し、既存の同期システムに必要なスクリプトを実装します。アップロードスクリプトは必要ありません。このスクリプトバージョンは、ファイルベースのダウンロードのみに使用されます。このスクリプトバージョンの場合、パラメータとして Mobile Link ユーザ名を利用するすべてのスクリプトは、リモートデータベースのグループを示す Mobile Link ユーザ名をパラメータとして利用します。これは、ファイル定義データベースで定義されているユーザ名です。

定義した各スクリプトバージョンには、begin_publication スクリプトを実装します。

タイムスタンプベースのダウンロードの場合は、各スクリプトバージョンに modify_last_download_timestamp スクリプトを実装します。このスクリプトの実装方法は、各ダウンロードファイルで送信するデータ量によって異なります。たとえば、グループのユーザによる前回の正常なダウンロードの時刻の中で最も早いものを使用する方法があります。このスクリプトに渡される ml_username パラメータは、実際にはグループ名です。

注意

Microsoft SQL Server 統合データベースからファイルベースのダウンロードファイルを生成する場合は、Mobile Link サーバで -dsd オプションを使用することを強くおすすめします。-dsd オプションを使用しないと、リモートでファイルベースのダウンロードファイルを適用できないことがあり、次のようなエラーが通知されます。**「パブリケーション <publication> の最終ダウンロード時間は <timestamp> です。ダウンロードファイルの次の最終ダウンロード時間は <timestamp> でした。ダウンロードファイルの次の最終ダウンロード時刻がサブスクリプションの最終ダウンロード時刻の前である場合は、ダウンロードファイルを適用できません。」**

一般的に、リモートデータベースでダウンロードファイルを適用できなくなる頻度は、リモートデータベースで通常の (接続済み) 同期を実行する頻度と統合データベースでの同時アクティビティの量に比例します。

「-dsd mlsrv16 オプション」59 ページを参照してください。

参照

- 「スクリプトバージョン」252 ページ
- 「begin_publication 接続イベント」310 ページ
- 「modify_last_download_timestamp 接続イベント」391 ページ

ダウンロードファイルの作成

ダウンロードファイルには、同期されるデータが格納されています。ダウンロードファイルを作成するには、上記の説明のようにファイル定義データベースと統合データベースを設定します。-bc オプションを使用し、拡張子が *.df* のファイル名を指定して dbmlsync を実行します。次に例を示します。

```
dbmlsync -c "UID=DBA;PWD=sql;SERVER=fbdl_eng;DBF=fdef.db" -v+
  -e "sv=filebased" -bc file1.df
```

ダウンロードファイルの作成時にオプションを指定することもできます。

- **-be オプション**　-be オプションを使用すると、sp_hook_dbmlsync_validate_download_file ストアドプロシージャを使用してリモートデータベースでアクセス可能なダウンロードファイルに文字列を追加できます。

- **-bg オプション**　-bg オプションを使用すると、一度も同期されていないリモートデータベースによって使用可能なダウンロードファイルを作成できます。

参照

- 「-be dbmlsync オプション」『Mobile Link クライアント管理』
- 「sp_hook_dbmlsync_validate_download_file」『Mobile Link クライアント管理』
- 「-bg dbmlsync オプション」『Mobile Link クライアント管理』

新しいリモートの同期

Mobile Link を使用して同期されたことのないリモートデータベースにダウンロードファイルを適用する場合は、リモートデータベースで通常の同期を実行してからダウンロードファイルを適用するか、ダウンロードファイルの作成時に dbmlsync -bg オプションを使用する必要があります。

タイムスタンプベースの同期の場合は、この 2 つの方法のいずれかを行うと、データの初期のスナップショットがダウンロードされます。タイムスタンプベースとスナップショットベースの両方の同期では、この手順によって、統合データベースの begin_publication スクリプトが生成する値に世代番号が設定されます。

通常の同期の実行

ダウンロードファイルを使用しない同期を実行することによって、ダウンロードファイルを受信するためのリモートデータベースを準備します。

-bg オプションの使用

別の方法として、まだ同期されていないリモートデータベースで使用するために、-bg オプションを使用してダウンロードファイルを作成できます。この初期ダウンロードファイルを適用して、ファイルベースの同期に使用されるリモートデータベースを準備します。

- **スナップショットのダウンロード**　スナップショットのダウンロードを実行している場合は、初期ダウンロードファイルに世代番号を設定する必要があります。このファイルにデータの初期スナップショットを含めることは可能ですが、各スナップショットのダウンロードにはすべてのデータが含まれ、前のダウンロードに依存しないため必須ではありません。

スナップショットのダウンロードは、-bg オプションを使用すると簡単です。ダウンロードファイルを作成するときに、dbmlsync コマンドで -bg を指定するだけです。同じスクリプトバージョンを使用して、以降のダウンロードファイルに使用する初期ダウンロードファイルを作成できます。

- **タイムスタンプベースのダウンロード**　タイムスタンプベースのダウンロードを実行している場合は、初期ダウンロードでリモートデータベースの世代番号を設定し、データのスナップショットを含める必要があります。タイムスタンプベースのダウンロードでは、各ダウンロードは前のダウンロードに基づいています。ダウンロードファイルには、それぞれ最終ダウンロードタイムスタンプが格納されています。ファイルの最終ダウンロードタイムスタンプの後に統合データベースで変更されたローは、すべてこのファイルに格納されています。ファイルを適用するには、ファイルの最終ダウンロードタイムスタンプの前に発生したすべての変更をリモートデータベースが受信している必要があります。この確認は、ファイルの最終ダウンロードタイムスタンプが、リモートデータベースの最終ダウンロードタイムスタンプ (リモートデータベースが統合データベースからすべての変更を受信するまでの時刻) 以降であることをチェックして行われます。

  リモートデータベースでは、最初の通常のダウンロードファイルを適用する前に、ファイルの最終ダウンロードタイムスタンプより前に変更され、しかも 1900 年 1 月 1 日より後に変更されたすべてのデータを受信している必要があります。-bg オプションを使用して作成した初期ダウンロードファイルには、このデータが含まれている必要があります。このデータを選択する最も簡単な方法は、通常のファイルベースの同期スクリプトバージョンと同じ download_cursor を使用していても、modify_last_download_timestamp スクリプトは含まない別のスクリプトバージョンを作成することです。no modify_last_download_timestamp スクリプトが定義されていない場合、ファイルベースのダウンロードの最終ダウンロードタイムスタンプは、デフォルトで 1900 年 1 月 1 日に設定されます。

-bg オプションを使用して構築されたダウンロードファイルを同期済みのリモートデータベースに適用すると、この -bg オプションにより、ダウンロードファイルが作成されたときの統合データベースの値を使用してリモートデータベースで世代番号が更新されます。このため、世代番号は無効になります。世代番号は、消失または破損した統合データベースをリカバリする場合にアップロードが実行されるまで、ファイルベースのダウンロードをそれ以上適用しないようにするためのものです。

参照

- 「Mobile Link の世代番号」219 ページ
- 「-bg dbmlsync オプション」『Mobile Link クライアント管理』

検証チェック

dbmlsync は、同期が有効であることを確認するためにいくつかの処理を行ってから、ダウンロードファイルをリモートデータベースに適用します。

- dbmlsync は、ダウンロードファイルの作成に使用されたファイル定義データベースに次のものが含まれていることを確認するため、このダウンロードファイルをチェックします。

  - リモートデータベースと同じパブリケーション

○そのパブリケーションで使用される同じテーブルとカラム

○それらのテーブルとカラムと同じ外部キー関係

●dbmlsync は、リモートデータベースからアップロードされていないデータがパブリケーションに存在するかどうかをチェックします。データが存在する場合は、ダウンロードファイルは適用されません。これは、ダウンロードファイルを適用すると、保留中のアップロードが消失することがあるためです。

●dbmlsync は、最終ダウンロードタイムスタンプ、次の最終ダウンロードタイムスタンプ、ダウンロードファイルの作成時刻をチェックして、次のことを確認します。

○リモートデータベースの新しいデータが、ダウンロードファイルに含まれる古いデータで上書きされないこと。

○ダウンロードファイルを適用すると、統合データベースで発生した変更の一部をリモートデータベースが取得しない場合には、ダウンロードファイルを適用しないこと。この状況は、リモートデータベースが前のファイルベースのダウンロードを適用しなかった場合に発生することがあります。

「自動検証」217 ページを参照してください。

●オプションで、dbmlsync はダウンロードファイルの世代番号と一致することを確認するために、リモートデータベースで世代番号をチェックします。

「Mobile Link の世代番号」219 ページを参照してください。

●オプションで、sp_hook_dbmlsync_validate_download_file ストアドプロシージャを使用して、カスタムの検証論理を作成できます。

詳細については、「カスタム検証」219 ページを参照してください。

自動検証

dbmlsync は、最終ダウンロードタイムスタンプ、次の最終ダウンロードタイムスタンプ、ダウンロードファイルの作成時刻、トランザクションログに特別なチェックを実行してから、ダウンロードファイルを適用します。

最終ダウンロードタイムスタンプと次の最終ダウンロードタイムスタンプ

各ダウンロードファイルには、ファイルの最終ダウンロードタイムスタンプから次の最終ダウンロードタイムスタンプまでの間に統合データベースで発生したダウンロード対象のすべての変更が格納されています。統合データベースでの時刻は、両方の時刻の値に使用されます。デフォルトでは、ファイルの最終ダウンロード時刻は 1900 年 1 月 1 日 12:00 AM で、ファイルの次の最終ダウンロードタイムスタンプはダウンロードファイルが作成された時刻です。これらのデフォルト値を上書きするには、generate_next_last_download_timestamp スクリプト、modify_last_download_timestamp スクリプト、modify_next_last_download_timestamp スクリプトを統合データベースに実装します。

リモートサイトは、ファイルの最終ダウンロードタイムスタンプが、リモートの最終ダウンロードタイムスタンプ以前である場合にのみ、ダウンロードファイルを適用できます。これにより、

リモートデータベースは統合データベースで発生した操作を失うことはありません。通常、このチェックに基づいたファイルベースのダウンロードが失敗した場合、リモートデータベースは 1 つまたは複数のダウンロードファイルを失っていることになります。この状況を修正するには、取得しなかったダウンロードファイルを適用するか、完全な同期またはダウンロード専用の同期を実行します。

さらに、リモートサイトは、次のファイルの最終ダウンロードタイムスタンプが、リモートデータベースの最終ダウンロードタイムスタンプよりも後である場合にのみ、ダウンロードファイルを適用できます。リモートデータベースの最終ダウンロードタイムスタンプは、ダウンロード対象のすべての変更をリモートデータベースが受信するまでの時刻 (統合データベースでの時刻) です。リモートデータベースの最終ダウンロード時刻は、通常またはファイルベースのダウンロードをリモートデータベースが正常に適用するたびに更新されます。このチェックを行うことにより、より新しいデータがすでにダウンロードされている場合はダウンロードファイルが適用されることはありません。一般的には、これが発生するのは、ダウンロードファイルが正常に適用されなかった場合です。たとえば、ダウンロードファイル *F1.df* が作成され、別のファイル *F2.df* が後で作成されたとします。このチェック機能により、*F2.df* の後に *F1.df* が適用されることはありません。これは、*F2.df* の新しいデータが、*F1.df* の古いデータで上書きされてしまうのを防ぐためです。

次の最終ダウンロードタイムスタンプに基づいたファイルベースのダウンロードが失敗した場合、このファイルを削除する以外に必要な作業はありません。新しいファイルを受信すると、同期は成功します。

作成時刻

ダウンロードファイルの作成時刻は、ファイルの作成が開始された時点の統合データベースでの時刻を示しています。ダウンロードファイルを適用できるのは、ファイルの作成時刻が、リモートデータベースの最終アップロード時刻よりも後の場合だけです。リモートの最終アップロード時刻は、リモートの正常な最終アップロードがコミットされた時点の統合データベースでの時刻です。このチェックにより、ダウンロードの作成後にアップロードされた (ダウンロードよりも新しい) データは、ダウンロードファイルの古いデータで上書きされることはありません。

このチェックに基づいてダウンロードファイルが拒否されても、必要な作業はありません。リモートサイトは、次のダウンロードファイルの適用が可能になっている必要があります。

dbmlsync がアップロードを Mobile Link サーバに送信した後に確認を取得しなかったためにアップロードが失敗した場合は、リモートデータベースの最終アップロード時刻が正しくないことがあります。この場合、作成時刻のチェックを実行できません。また、リモートデータベースは通常の同期を完了するまでダウンロードファイルを適用できません。

トランザクションログ

dbmlsync は、リモートデータベースのトランザクションログをスキャンし、アップロードする必要があるすべての変更のリストを構築してから、ダウンロードファイルを適用します。dbmlsync がダウンロードファイルを適用するのは、アップロードが必要な変更のあるローに影響する操作がダウンロードファイルに含まれていない場合だけです。

Mobile Link の世代番号

世代番号とは、リモートデータベースがデータをアップロードしてからダウンロードファイルを適用するようにするためのメカニズムです。これは、統合データベースで問題が発生したためにデータが失われ、そのデータをリモートデータベースからリカバリする必要があるときに、特に役立ちます。

リモートデータベースでは、各サブスクリプションに対して別々の世代番号が自動的に管理されています。統合データベースでは、各サブスクリプションの世代番号は begin_publication スクリプトによって決定されます。リモートデータベースが正常にアップロードを行うたびに、リモートデータベースの世代番号は統合データベースの begin_publication スクリプトによって設定された値で更新されます。

ダウンロードファイルが作成されるたびに、begin_publication スクリプトによって設定された世代番号がダウンロードファイルに格納されます。リモートサイトは、ダウンロードファイルの世代番号がリモートデータベースに格納されている世代番号と同じ場合にのみ、ダウンロードファイルを適用します。

注意
begin_publication スクリプトによってファイルベースのダウンロード用に生成された世代番号が変更された場合、リモートデータベースは正常なアップロードを実行してから、新しいダウンロードファイルを適用する必要があります。

sp_hook_dbmlsync_validate_download_file ストアドプロシージャを使用すると、デフォルトで行われる世代番号のチェックを無効にすることができます。

Mobile Link の世代番号の管理については、次の項を参照してください。

- 「begin_publication 接続イベント」310 ページ
- 「end_publication 接続イベント」350 ページ
- 「sp_hook_dbmlsync_validate_download_file」『Mobile Link クライアント管理』

カスタム検証

カスタムの検証論理を作成すると、ダウンロードファイルをリモートデータベースに適用する必要があるかどうかを判断できます。これには、sp_hook_dbmlsync_validate_download_file ストアドプロシージャを使用します。このストアドプロシージャを使用すると、ダウンロードファイルを拒否し、デフォルトで行われる世代番号のチェックを無効にすることができます。

dbmlsync -be オプションを使用すると、文字列をファイルに埋め込むことができます。-be オプションは、ダウンロードファイルを作成するときに、ファイル定義データベースに対して使用します。この文字列は、#hook_dict テーブルを介して sp_hook_dbmlsync_validate_download_file に渡され、検証論理で使用できます。

参照

- 「sp_hook_dbmlsync_validate_download_file」『Mobile Link クライアント管理』

ファイルベースのダウンロード例

この項には 2 つの例が用意されています。それぞれの例では、統合データベースと 1 つのテーブルのみを使用して、ファイルベースのダウンロードの同期を設定します。1 番目は簡単なスナップショットの例で、2 番目は多少複雑なタイムスタンプベースの例です。

スナップショットの例

この例では、スナップショット同期のファイルベースのダウンロードを実行します。最初に、ファイルベースのダウンロードで必要な 3 つのデータベースを設定し、次に、データをダウンロードする方法を示します。この例は、参考にするだけでもかまいませんし、テキストをコピーアンドペーストしてサンプルを実行することもできます。

サンプル用のデータベースの作成

次のコマンドは、この例で使用する 3 つのデータベース (統合データベース、リモートデータベース、ファイル定義データベース) を作成します。

```
dbinit -dba DBA,sql scons.db
dbinit -dba DBA,sql sremote.db
dbinit -dba DBA,sql sfdef.db
```

次のコマンドは、この 3 つのデータベースを起動し、統合データベースへの接続に使用する Mobile Link のデータソース名を作成します。

```
dbeng16 -n sfdef_eng sfdef.db
dbeng16 -n scons_eng scons.db
dbeng16 -n sremote_eng sremote.db
dbdsn -y -w fbd_demo -c
"SERVER=scons_eng;DBF=scons.db;UID=DBA;PWD=sql;ASTART=off;ASTOP=off"
```

Interactive SQL を開き、*scons.db* に接続して、Mobile Link 設定スクリプトを実行します。次に例を示します。

```
read "C:¥Program Files¥SQL Anywhere 16¥MobiLink¥setup¥syncsa.sql"
```

Mobile Link サーバを起動します。

```
start mlsrv16 -v+ -c "DSN=fbd_demo" -zu+ -ot scons.txt
```

スナップショットの例で使用する統合データベースの設定

この例では、統合データベースには T1 というテーブルが 1 つあります。統合データベースに接続すると、次の SQL を実行してテーブル T1 を作成できます。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);
```

次のコードは、filebased というスクリプトバージョンを作成し、そのスクリプトバージョンのダウンロードスクリプトを作成します。

```
CALL ml_add_table_script( 'filebased',
  'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );
```

次のコードは、normal というスクリプトバージョンを作成し、そのスクリプトバージョンのアップロードスクリプトとダウンロードスクリプトを作成します。

```
CALL ml_add_table_script ( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1 VALUES ({ml r.pk}, {ml r.c1})');

CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );

CALL ml_add_table_script( 'normal', 'T1',
 'upload_delete',
 'DELETE FROM T1 WHERE pk = {ml r.pk}' );

CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1' );

CALL ml_add_table_script( 'normal', 'T1',
  'download_delete_cursor',
  '--{ml_ignore}' );

COMMIT;
```

次のコマンドは、ストアドプロシージャ begin_pub を作成し、begin_pub が、"normal" スクリプトバージョンと "filebased" スクリプトバージョンの両方を対象とした begin_publication スクリプトであることを指定します。

```
CREATE PROCEDURE begin_pub (
      INOUT generation_num integer,
      IN    username       varchar(128),
      IN    pubname        varchar(128) )
BEGIN
   SET generation_num=1;
END;

CALL ml_add_connection_script(
  'filebased',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );

CALL ml_add_connection_script(
  'normal',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );
```

スナップショット例で使用するリモートデータベースの作成

この例では、リモートデータベースにも T1 というテーブルが 1 つあります。リモートデータベースに接続し、次の SQL コマンドを実行して、テーブル T1、パブリケーション P1、ユーザ U1 を作成します。また、この SQL は P1 に対する U1 のサブスクリプションも作成します。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
   TABLE T1
);

CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

次のコードは、sp_hook_dbmlsync_validate_download_file フックを作成して、ユーザ定義の検証論理をリモートデータベースに実装します。

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
   DECLARE udata varchar(256);
SELECT value
   INTO udata
   FROM #hook_dict
   WHERE name = 'user data';
   IF udata <> 'ok' THEN
   UPDATE #hook_dict
   SET value = 'FALSE'
   WHERE name = 'apply file';
   END IF;
END
```

スナップショット例で使用するファイル定義データベースの作成

ファイルベースのダウンロードを使用する Mobile Link システムには、ファイル定義データベースが必要です。このデータベースのスキーマはファイルベースのダウンロードで更新されるリモートデータベースのスキーマと同じですが、データとステータス情報は格納されていません。ファイル定義データベースは、ダウンロードファイルに格納されるデータの構造を定義するためだけに使用します。リモートデータベースの Mobile Link グループのユーザ名で定義された、多数のグループのリモートデータベースに対して、1 つのファイル定義データベースを使用できます。

次のコードは、この例で使用するファイル定義データベースを定義します。このコードはリモートデータベースと同じスキーマを作成し、さらに以下を作成します。

- P1 という名前のパブリケーション。T1 テーブルのすべてのローをパブリッシュします。ファイル定義データベースとリモートデータベースでは、同じパブリケーション名を使用する必要があります。

- G1 という名前の Mobile Link ユーザ。このユーザは、ファイルベースのダウンロードで更新されるすべてのリモートデータベースを表しています。

●パブリケーションに対するサブスクリプション

sfdef.db に接続してから、次のコードを実行してください。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
   TABLE T1
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

初期同期の準備

ダウンロードファイルを適用できるようにするために新しいリモートデータベースを準備するには、通常の同期を実行するか、dbmlsync -bg オプションを使用してダウンロードファイルを作成します。この例は、通常の同期を実行して新しいリモートデータベースを初期化する方法を示しています。

リモートデータベースの初期同期は、以前に作成した normal というスクリプトバージョンで実行できます。

```
dbmlsync -c "UID=DBA;PWD=sql;SERVER=sremote_eng;DBF=sremote.db" -v+ -e "sv=normal"
```

スナップショット例におけるファイルベースのダウンロードの実行

統合データベースに接続し、ファイルベースのダウンロードで同期される次のようなデータをいくつか挿入します。

```
INSERT INTO T1 VALUES( 1, 1 );
INSERT INTO T1 VALUES( 2, 4 );
INSERT INTO T1 VALUES( 3, 9 );
COMMIT;
```

次のコマンドは、ファイル定義データベースのあるコンピュータで実行してください。次の処理が行われます。

●dbmlsync -bc オプションにより、ダウンロードファイルが作成され、*file1.df* という名前が付けられます。

●-be オプションにより、"OK" という文字列がダウンロードファイルに追加され、sp_dbmlsync_validate_download_file フックへのアクセスが可能になります。

```
dbmlsync -c "UID=DBA;PWD=sql;SERVER=sfdef_eng;DBF=sfdef.db" -v+ -e "sv=filebased" -bc file1.df -
be ok -ot fdef.txt
```

ダウンロードファイルを適用するには、-ba オプションと、適用するダウンロードファイルの名前を指定して、リモートデータベースで dbmlsync を実行します。

```
dbmlsync -c "UID=DBA;PWD=sql;SERVER=sremote_eng; DBF=sremote.db" -v+ -ba file1.df -ot
remote.txt
```

これで、リモートデータベースに変更が適用されました。Interactive SQL を開いてリモートデータベースに接続し、次に示す SQL 文を実行して、リモートデータベースにデータがあることを確認します。

```
SELECT * FROM T1
```

スナップショット例のクリーンアップ

次のコマンドは、3 つのデータベースサーバをすべて停止してから、ファイルを消去します。

```
del file1.df
mlstop -h -w
dbstop -y -c "SERVER=sfdef_eng; UID=DBA; PWD=sql"
dbstop -y -c "SERVER=scons_eng; UID=DBA; PWD=sql"
dbstop -y -c "SERVER=sremote_eng; UID=DBA; PWD=sql"
dberase -y sfdef.db
dberase -y scons.db
dberase -y sremote.db
```

タイムスタンプベースの例

この例では、タイムスタンプベースの同期のファイルベースのダウンロードを実行します。3 つのデータベースを設定し、次に、ファイルによってデータをダウンロードする方法を示します。この例は、参考にするだけでもかまいませんし、テキストをコピーアンドペーストしてサンプルを実行することもできます。

サンプル用のデータベースの作成

次のコマンドは、この例で使用する 3 つのデータベース (統合データベース、リモートデータベース、ファイル定義データベース) を作成します。

```
dbinit -dba DBA,sql tcons.db
dbinit -dba DBA,sql tremote.db
dbinit -dba DBA,sql tfdef.db
```

次のコマンドは、この 3 つのデータベースを起動し、統合データベースへの接続に使用する Mobile Link のデータソース名を作成します。

```
dbeng16 -n tfdef_eng tfdef.db
dbeng16 -n tcons_eng tcons.db
dbeng16 -n tremote_eng tremote.db
dbdsn -y -w tfbd_demo -c
"SERVER=tcons_eng;DBF=tcons.db;UID=DBA;PWD=sql;START=off;ASTOP=off"
```

Interactive SQL を開き、*tcons.db* に接続して、Mobile Link 設定スクリプトを実行します。次に例を示します。

```
read "C:¥Program Files¥SQL Anywhere 16¥MobiLink¥setup¥syncsa.sql"
```

Mobile Link サーバを起動します。

```
start mlsrv16 -v+ -c "DSN=tfbd_demo" -zu+ -ot tcons.txt
```

タイムスタンプの例で使用する統合データベースの設定

この例では、統合データベースには T1 というテーブルが 1 つあります。統合データベースに接続すると、次のコードを実行してテーブル T1 を作成できます。

```
CREATE TABLE T1 (
 pk  INTEGER PRIMARY KEY,
 c1  INTEGER,
 last_modified TIMESTAMP DEFAULT TIMESTAMP
);
```

次のコードは、最小限の数のスクリプトで構成される normal というスクリプトバージョンを定義します。このスクリプトバージョンは、ファイルベースのダウンロードを「使用しない」同期に使用されます。

```
CALL ml_add_table_script( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1( pk, c1) VALUES( {ml r.pk}, {ml r.c1} )' );

CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );

CALL ml_add_table_script( 'normal', 'T1',
  'upload_delete',
  'DELETE FROM T1 WHERE pk = {ml r.pk}' );

CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1
    WHERE last_modified >= {ml s.last_table_download}' );
```

次のコードは、すべてのサブスクリプションの世代番号を 1 に設定します。世代番号は、統合データベースが消失または破損し、アップロードが必要となった場合に使用すると便利です。

```
CREATE PROCEDURE begin_pub (
      INOUT generation_num integer,
      IN    username       varchar(128),
      IN    pubname        varchar(128) )
BEGIN
 SET generation_num = 1;
END;

CALL ml_add_connection_script( 'normal',
   'begin_publication',
   '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name},
      {ml s.last_publication_upload},
      {ml s.last_publication_download} ) }' );

COMMIT;
```

次のコードは、filebased というスクリプトバージョンを定義します。このスクリプトバージョンは、ファイルベースのダウンロードの作成に使用されます。

```
CALL ml_add_connection_script( 'filebased',
   'begin_publication',
   '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );

CALL ml_add_table_script( 'filebased', 'T1',
   'download_cursor',
```

```
    'SELECT pk, c1 FROM T1
      WHERE last_modified >= {ml s.last_table_download}' );
```

次のコードは、最後の 5 日間に発生したすべての変更がダウンロードファイルに追加されるように最終ダウンロード時刻を設定します。最後の 5 日間に作成されたどのダウンロードファイルも取得していないリモートデータベースは、通常の同期を実行しないと、これよりも後のファイルベースのダウンロードを適用することはできません。

```
CREATE PROCEDURE ModifyLastDownloadTimestamp(
      INOUT last_download_timestamp TIMESTAMP,
      IN    ml_username             VARCHAR(128) )
BEGIN
   SELECT dateadd( day, -5, CURRENT TIMESTAMP )
   INTO last_download_timestamp;
END;

CALL ml_add_connection_script( 'filebased',
  'modify_last_download_timestamp',
  'CALL ModifyLastDownloadTimestamp(
     {ml s.last_download}, {ml s.username} )' );

COMMIT;
```

タイムスタンプベースの同期で使用するリモートデータベースの作成

この例では、リモートデータベースにも T1 というテーブルが 1 つあります。リモートデータベースに接続した後、次の SQL コマンドを実行して、テーブル T1、パブリケーション P1、ユーザ U1 を作成します。また、このコードは P1 に対する U1 のサブスクリプションも作成します。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
   TABLE T1
);

CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

次のコードは、sp_hook_dbmlsync_validate_download_file ストアドプロシージャを定義します。このストアドプロシージャは、文字列 "ok" が埋め込まれていないダウンロードファイルの適用を防止します。

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
   DECLARE udata varchar(256);

   SELECT value
   INTO udata
   FROM #hook_dict
   WHERE name = 'user data';

   IF udata <> 'ok' THEN
     UPDATE #hook_dict
     SET value = 'FALSE'
```

```
      WHERE name = 'apply file';
    END IF;
END
```

タイムスタンプベースの同期で使用するファイル定義データベースの作成

次のコードは、タイムスタンプベースの同期で使用するファイル定義データベースを定義します。また、このコードは、テーブル、パブリケーション、ユーザ、そのパブリケーションに対するユーザのサブスクリプションを作成します。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

初期同期の準備

ダウンロードファイルを適用できるようにするために新しいリモートデータベースを準備するには、通常の同期を実行するか、dbmlsync -bg オプションを使用してダウンロードファイルを作成します。この例では、-bg の使用方法を示します。

次のコードは、統合データベースの filebased_init というスクリプトバージョンを定義します。このスクリプトバージョンには、1 つの begin_publication スクリプトがあります。

```
CALL ml_add_table_script(
  'filebased_init', 'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );

CALL ml_add_connection_script(
  'filebased_init',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );

COMMIT;
```

次のコマンドラインでは、filebased_init というスクリプトバージョンと -bg オプションを使用して初期ダウンロードファイルを作成、適用します。

```
dbmlsync -c "UID=DBA;PWD=sql;SERVER=tfdef_eng;DBF=tfdef.db"
  -v+ -e "sv=filebased_init" -bc tfile1.df -be ok -bg
  -ot tfdef1.txt

dbmlsync -c "UID=DBA;PWD=sql;SERVER=tremote_eng;DBF=tremote.db"
  -v+ -ba tfile1.df -ot tremote.txt
```

タイムスタンプベースの同期のファイルベースのダウンロードを実行する

統合データベースに接続し、ファイルベースのダウンロードで同期される次のようなデータをいくつか挿入します。

```
INSERT INTO T1(pk, c1) VALUES( 1, 1 );
INSERT INTO T1(pk, c1) VALUES( 2, 4 );
INSERT INTO T1(pk, c1) VALUES( 3, 9 );
commit;
```

次のコマンドラインは、新しいデータを含むダウンロードファイルを作成します。

```
dbmlsync -c
  "UID=DBA;PWD=sql;SERVER=tfdef_eng;DBF=tfdef.db"
  -v+ -e "sv=filebased" -bc tfile2.df -be ok -ot tfdef2.txt
```

次のコマンドラインは、ダウンロードファイルをリモートデータベースに適用します。

```
dbmlsync -c "UID=DBA;PWD=sql;SERVER=tremote_eng;DBF=tremote.db"
  -v+ -ba tfile2.df -ot tfdef3.txt
```

これで、リモートデータベースに変更が適用されました。Interactive SQL を開いてリモートデータベースに接続し、次に示す SQL 文を実行して、リモートデータベースにデータがあることを確認します。

```
SELECT * FROM T1
```

タイムスタンプベースの同期のクリーンアップ

次のコマンドは、3 つのデータベースサーバをすべて停止してから、ファイルを消去します。

```
del tfile1.df
mlstop -h -w
dbstop -y -c "SERVER=tfdef_eng; UID=DBA; PWD=sql"
dbstop -y -c "SERVER=tcons_eng; UID=DBA; PWD=sql"
dbstop -y -c "SERVER=tremote_eng; UID=DBA; PWD=sql"
dberase -y tfdef.db
dberase -y tcons.db
dberase -y tremote.db
```

Mobile Link イベント

この項では、Mobile Link イベントに対するスクリプトを記述する方法について説明します。

同期スクリプトの作成

同期スクリプトを作成して、統合データベースの Mobile Link システムテーブルに格納したりそこで参照することで、同期処理を制御できます。スクリプトは、SQL、Java、または .NET で作成できます。

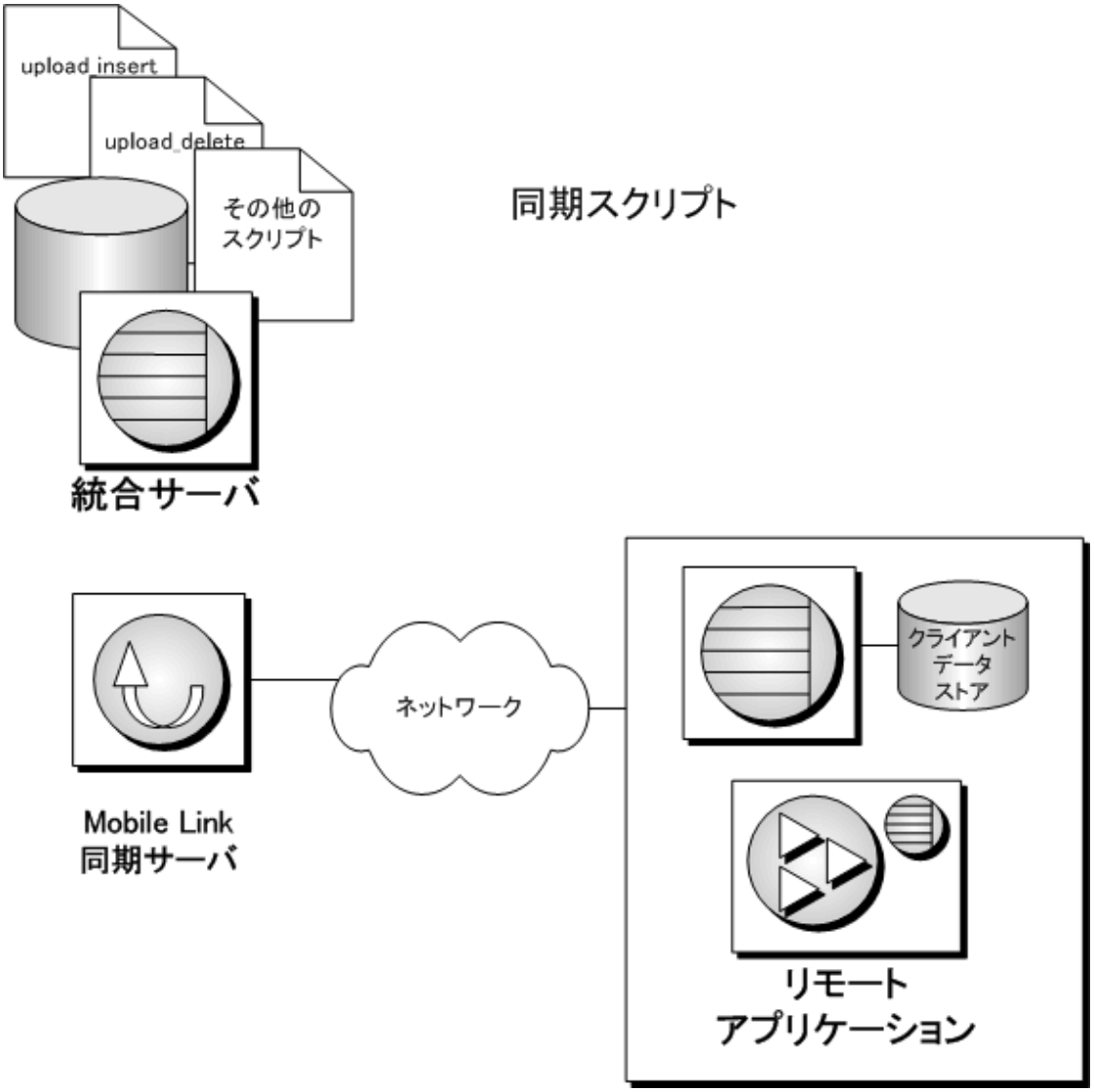
「Mobile Link 同期論理」は、同期スクリプトを使って指定されます。スクリプトでは次の内容を定義します。

- リモートデータベースからアップロードしたデータを、統合データベースに適用する方法
- 統合データベースからダウンロードするデータ
- 同期中の認証方法 (任意)

スクリプトは個別の文またはストアドプロシージャコールです。統合データベースに格納されるか、統合データベースで参照されます。スクリプトを統合データベースに追加するには、Sybase Central またはシステムプロシージャを使用できます。

警告
SQL 同期スクリプト、または SQL 同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。SQL スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、Mobile Link では、障害が発生した場合にデータの整合性を保証できません。

同期中は、Mobile Link サーバがスクリプトを読み込み (まだロードされていない場合)、統合データベースに対してスクリプトを実行します。



同期処理には複数の手順があります。各手順は、ユニークなイベントによって識別されます。これらのイベントのいずれかに対応するスクリプトを作成することによって、同期処理を制御します。スクリプトは、特定のイベントで特定の動作を行う必要がある場合にのみ作成します。Mobile Link サーバは、イベントが発生するとそれに対応するスクリプトを実行します。特定のイベントに対してスクリプトを定義していない場合、Mobile Link サーバは単に次の手順に進みます。

たとえば、begin_upload_rows というイベントを考えてみます。スクリプトを作成して、このイベントに関連付けることができます。Mobile Link サーバは、このスクリプトを、必要となった時点で初めて読み込み、同期のアップロードフェーズ中に実行します。スクリプトを作成しなかった場合、Mobile Link サーバは直ちに次の手順、つまり、アップロードされたローを処理する手順に移ります。

テーブルスクリプトと呼ばれるスクリプトは、イベントだけでなく、リモートデータベースの特定のテーブルにも対応します。Mobile Link サーバは、いくつかのタスクをテーブル単位で実行します (たとえばローのダウンロード)。同じイベントでもアプリケーションテーブルが異なれば、複数のスクリプトに対応できます。または、いくつかのアプリケーションテーブルに対しては多くのスクリプトを定義し、他のアプリケーションテーブルに対してはスクリプトをほとんど定義しないこともできます。

イベントの概要については、「同期処理」『Mobile Link クイックスタート』を参照してください。

作成可能な各スクリプトの説明については、「同期イベント」269 ページを参照してください。

スクリプトは、SQL、Java、または .NET で作成できます。この項の内容はすべての種類のスクリプトに適用されますが、主に SQL で同期スクリプトを作成する方法について説明します。

SQL、Java、.NET の説明と比較については、「サーバ側の同期論理の作成オプション」『Mobile Link クイックスタート』を参照してください。

.NET でのスクリプト作成については、「.NET の同期スクリプト」533 ページを参照してください。

Java でのスクリプト作成については、「Java による同期スクリプトの作成」459 ページを参照してください。

同期スクリプトを実装する方法については、「同期の方法」107 ページを参照してください。

簡単な同期スクリプト

Mobile Link には利用可能なイベントがたくさんありますが、その各イベントに対してスクリプトを作成する必要はありません。簡単な同期モデルの場合、必要なスクリプトはわずかです。

テーブルから各リモートデータベースにすべてのローをダウンロードすると、CustDB サンプルアプリケーションの ULProduct テーブルと同期します。この場合、リモートデータベースでの追加は許可されません。この簡単な形の同期は、2 つのスクリプトで実装できます。この例では、2 つのイベントだけに関連するスクリプトを使用します。

各同期中にダウンロードするローを制御する Mobile Link イベントは、download_cursor イベントと呼ばれます。カーソルスクリプトには、SELECT 文が必要です。Mobile Link サーバは、このクエリを使用してカーソルを定義します。download_cursor スクリプトの場合、カーソルによって、リモートデータベース内の特定の 1 テーブルにダウンロードするローが選択されます。

CustDB サンプルアプリケーションには、このサンプルアプリケーションにある ULProduct テーブルに対応する download_cursor スクリプトが 1 つあり、次のクエリから構成されています。

```
SELECT prod_id, price, prod_name
FROM ULProduct
```

このクエリは結果セットを生成します。クライアントには、この結果セットを構成するローがダウンロードされます。この場合は、テーブルのすべてのローがダウンロードされます。

Mobile Link サーバでは、ULProduct アプリケーションテーブルへローを送信することを認識しています。これは、このスクリプトが download_cursor イベントと ULProduct テーブルの両方に対

応するような方法で統合データベースに格納されているからです。Sybase Central ではこのような対応付けが可能です。

ダウンロードするテーブルごとに download_cursor とともに必要な 2 つ目のイベントは、download_delete_cursor です。このイベントには、スクリプトが定義されている必要があります。この簡単な例では、削除のダウンロードを使用しないため、スクリプトは --{ml_ignore} と定義されています。

この例では、クエリによって同じ ULProduct という名前の統合テーブルからデータが選択されます。この名前が同じである必要はありません。クエリを書き換えることで、統合データベース内の任意の単一または複数のテーブルのデータを ULProduct アプリケーションテーブルにダウンロードするようにできます。

より複雑な同期スクリプトを作成することもできます。たとえば、最近修正されたローだけをダウンロードするスクリプトや、リモートデータベースごとに異なる情報を提供するスクリプトを作成できます。

スクリプトと同期処理

各スクリプトは、同期処理の特定のイベントに対応します。特定の動作を行う必要がある場合にのみ、スクリプトを作成します。不要なイベントは、定義しないでおくことができます。

同期処理を大きく 2 つに分けると、アップロードされた情報の処理と、ダウンロードするためのローの準備があります。ローをリモートテーブルからアップロードする場合は、適切なアップロードスクリプトを定義します。テーブルのローを SQL でダウンロードする場合は、download_cursor スクリプトと download_delete_cursor スクリプトの両方を定義します。

Mobile Link サーバは、各スクリプトを、必要となった時点で初めて読み込み、準備します。以後は、イベントが呼び出されるたびにスクリプトが実行されます。

イベントの順序

Mobile Link イベントの完全な順序については、「Mobile Link イベントの概要」269 ページを参照してください。

アップロード処理の詳細については、「ローをアップロードするスクリプト」260 ページを参照してください。

ダウンロード処理の詳細については、「ローをダウンロードするスクリプト」262 ページを参照してください。

注意

- Mobile Link テクノロジによって、複数のクライアントを一度に同期させることができます。この場合、各クライアントは別々の接続を使用して統合データベースにアクセスします。

- 1 つの接続で複数の同期要求を処理できるので、begin_connection イベントと end_connection イベントは特定の同期に依存しません。これらのスクリプトには、パラメータがありません。これらは、接続レベルのスクリプトの例です。

- イベントの中には、同期されるテーブルの数に関係なく、各同期に対して 1 回だけ呼び出されるものがあります。これらは、接続レベルのスクリプトです。
- 各テーブルが同期されるたびに 1 回ずつ呼び出されるイベントもあります。これらのイベントに対応するスクリプトは、テーブルレベルのスクリプトと呼ばれます。

  各テーブルは専用のテーブルスクリプトを持つことができますが、いくつかのテーブルで共有されるテーブルレベルのスクリプトを作成することもできます。ただし、これは一般的ではありません。
- begin_synchronization など、一部のイベントは接続レベルとテーブルレベルの両方で発生します。これらのイベントに対しては、接続スクリプトとテーブルスクリプトの両方を作成できます。

各スクリプトやそのパラメータなどのリファレンス情報については、「同期イベント」269 ページを参照してください。

スクリプトの種類

同期スクリプトには、次の 2 種類があります。

- **接続レベルスクリプト**　接続専用または同期専用の、どのリモートテーブルにも依存しないアクションを実行します。同期ビジネスロジックを実装する場合は、これらのスクリプトを他のスクリプトと組み合わせて使用できます。
- **テーブルレベルスクリプト**　1 つの同期と特定の 1 つのリモートテーブル専用のアクションを実行します。このスクリプトを他のスクリプトと組み合わせて使用して、競合解決などの同期ビジネスロジックを実装します。

参照

- 「接続スクリプト」233 ページ
- 「テーブルスクリプト」234 ページ

接続スクリプト

接続レベルのスクリプトは、特定のテーブルに関連付けられていない高いレベルのイベントを制御します。これらのイベントは、各同期の処理中に必要な全般的なタスクを実行するときに使用します。

接続スクリプトは、接続と切断に関連するアクションや、アップロード処理やダウンロード処理の開始と終了などの、同期レベルのイベントのアクションを制御します。一部の接続スクリプトには、関連するテーブルスクリプトがあります。これらの接続スクリプトは、テーブルが同期されているかどうかに関係なく、いつでも呼び出されます。

接続レベルのスクリプトは、特定のイベントで特定のアクションを実行する必要がある場合にのみ作成します。少数のイベント用のスクリプトだけを作成すれば済む場合もあります。あらゆるイベントに対する Mobile Link サーバのデフォルトアクションは、何もアクションを実行しない設定になっています。簡単な同期スキームの中には、接続スクリプトが必要でないものもあります。

ml_global スクリプトバージョン

同じスクリプトを何回も定義しないで済むように、接続レベルスクリプトを 1 回定義して、任意のスクリプトバージョンから再使用できます。これを行うには、ml_global と呼ばれるスクリプトバージョンを定義します。

参照

- 「ml_global スクリプトバージョン」253 ページ

テーブルスクリプト

テーブルスクリプトによって、ローのアップロード、競合の解決、ダウンロードするローの選択など、特定のテーブルの同期に関する特定のイベントでのアクションを実行できます。

テーブルの同期スクリプトは、統合データベースのあらゆるテーブル (またはテーブルの組み合わせ) を参照できます。この機能を使用して、1 つまたは複数の統合テーブルに格納されたデータを特定のリモートテーブルに入れたり、1 つのリモートテーブルからアップロードされたデータを統合データベースの複数のテーブルに格納したりできます。

テーブル名は一致しなくてもよい

リモートデータベースでのテーブル名と統合データベースでのテーブル名は、同じである必要はありません。Mobile Link サーバは、ml_table システムテーブルでリモートテーブル名を検索し、テーブルに対応するスクリプトを特定します。スクリプト自体は、選択した統合テーブルを参照します。

スクリプトのパラメータ

同期スクリプトのほとんどは、Mobile Link サーバからパラメータを受け取ることができます。各スクリプトで使用できるパラメータの詳細については、「同期イベント」269 ページを参照してください。

次のいずれかの方法によって、SQL スクリプトでパラメータを指定できます。

- 名前付きスクリプトパラメータ
- 疑問符 (SQL スクリプトでは非推奨)

名前付きスクリプトパラメータ

名前付きパラメータには、推奨されなくなった疑問符と比べて次のような利点があります。

- 使用可能なパラメータの任意のサブセットを任意の順序で指定できます。
- in/out パラメータを除き、スクリプト内で同じ名前付きパラメータを複数回指定できます。
- 名前付きパラメータを使用する場合には、スクリプトでリモート ID を指定できます。スクリプトでリモート ID を指定できるのは、この方法だけです。

●独自の名前付きパラメータを作成できます。「ユーザ定義の名前付きパラメータ」250 ページを参照してください。

1 つのスクリプト内で名前付きパラメータと疑問符を混在させることはできません。

Mobile Link 名前付きパラメータには 4 種類あります。名前付きパラメータを指定するには、次のように種類をプレフィクスとして付ける必要があります。

名前付きパラメータの種類	プレフィクス	例
システムパラメータ	s.	{ml s.remote_id}
ローパラメータ (カラムの名前。カラム名にスペースが含まれる場合は二重引用符または角カッコで囲む)。	r.	{ml r.cust_id} {ml r."Column name"}
古いローパラメータ (プレイメージのカラムの値を指定するために upload_update スクリプトのみで使用。カラム名にスペースが含まれる場合は、二重引用符または角カッコで囲む)。	o.	{ml o.cust_name} {ml o."Column name"}
認証パラメータ。「認証パラメータ」251 ページを参照してください。	a.	{ml a.1}
ユーザ定義のパラメータ。「ユーザ定義の名前付きパラメータ」250 ページを参照してください。	u. または ui.	● **{ml u.varname}** パラメータが更新される場合に使用します。プレフィクスが u. のユーザ定義のパラメータは in/out です。 ● **{ml ui.varname}** パラメータの参照専用の場合に使用します。プレフィクスが ui. のユーザ定義のパラメータは入力専用です。

スクリプトパラメータを名前で参照するには、**{ml** *parameter* **}** のように、パラメータを中カッコで囲み、前に ml を付けます。たとえば、**{ml s.action_code}** のように指定します。中カッコによる表記は ODBC 規則です。

便宜上、Mobile Link スクリプトパラメータと同じ名前のスキーマ名がコードセクション内に含まれていないかぎり、中カッコには大きなコードセクションを含めることができます。たとえば、次の各 upload_insert スクリプトは有効で同じ内容を表しています。

```
INSERT INTO t ( id, c0 ) VALUES( {ml r.id}, {ml r.c0} )
```

および

```
INSERT INTO t ( is, c0 ) VALUES({ml r.id, r.c0})
```

および

```
{ml  INSERT INTO t ( id, c0 ) VALUES( r.id, r.c0 ) }
```

疑問符によって表されるスクリプトパラメータ (SQL では非推奨)

注意
疑問符を使用してパラメータを表す方法は、SQL スクリプトでは推奨されなくなりました。代わりに名前付きパラメータを使用することをおすすめします。 「名前付きスクリプトパラメータ」234 ページと「ユーザ定義の名前付きパラメータ」250 ページを参照してください。

疑問符を使ってパラメータを表すのは ODBC 規則です。Mobile Link SQL スクリプトで疑問符を使用するには、SQL スクリプトで各パラメータに対して 1 つ疑問符を置きます。Mobile Link サーバが、各疑問符をパラメータ値に置き換えます。パラメータ値への置き換えは、スクリプト内でパラメータが定義されている順序に従って行われます。

一部のパラメータは省略可能です。パラメータが省略可能になるのは、後続のパラメータを指定しない場合に限られます。この例では、パラメータ 2 を使用する場合には、パラメータ 1 を使用する必要があります。パラメータは、各イベントで指定された順に置いてください。 「同期イベント」269 ページを参照してください。

スクリプトパラメータのコメント

次の形式のコメントが認識されます。

- 二重ハイフンプレフィクス (--)
- 二重スラッシュプレフィクス (//)
- ブロックコメント (/* */)

初めの 2 つの形式を使用すると、行の最後までのスクリプトテキストが無視されます。3 つ目の形式を使用すると、/* プレフィクスと */ サフィックスの間にあるすべてのスクリプトテキストが無視されます。ブロックコメントはネストできません。

その他のタイプのベンダー固有コメントは認識されないため、名前付きパラメータへの参照のコメントには使用しないでください。

Mobile Link のシステムパラメータとイベント

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
action_code	接続	action_code パラメータは、次のイベントで使用できます。 ● handle_error ● handle_odbc_error ● report_error ● report_odbc_error
authentication_message	接続	authentication_message パラメータは、次のイベントで使用できます。 ● authenticate_parameters ● authenticate_user ● authenticate_user_hashed
authentication_status	接続	authentication_status パラメータは、次のイベントで使用できます。 ● authenticate_parameters ● authenticate_user ● authenticate_user_hashed
bytes	接続とテーブル	bytes パラメータは、次のイベントで使用できます。 ● download_statistics ● upload_statistics
conflicted_deletes	接続とテーブル	conflicted_deletes パラメータは、次のイベントで使用できます。 ● upload_statistics
conflicted_inserts	接続とテーブル	conflicted_inserts パラメータは、次のイベントで使用できます。 ● upload_statistics
conflicted_updates	接続とテーブル	conflicted_updates パラメータは、次のイベントで使用できます。 ● upload_statistics

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
connection_retries	接続	conflicted_retries パラメータは、次のイベントで使用できます。 ● synchronization_statistics
deadlocks	接続とテーブル	deadlocks パラメータは、次のイベントで使用できます。 ● synchronization_statistics (接続イベント) ● upload_statistics (接続イベントとテーブルイベント)
deleted_rows	接続とテーブル	deleted_rows パラメータは、次のイベントで使用できます。 ● download_statistics ● upload_statistics
error_code	接続	error_code パラメータは、次のイベントで使用できます。 ● handle_error ● modify_error_message ● report_error
error_message	接続	error_message パラメータは、次のイベントで使用できます。 ● handle_error ● handle_odbc_error ● modify_error_message ● report_error ● report_odbc_error
errors	接続とテーブル	errors パラメータは、次のイベントで使用できます。 ● download_statistics ● synchronization_statistics ● upload_statistics
event_name	接続とテーブル	event_name パラメータは、次のイベントで使用できます。 ● time_statistics

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
fetched_rows	接続とテーブル	fetched_rows パラメータは、次のイベントで使用できます。 ● download_statistics
file_authentication_code	接続	file_authentication_code パラメータは、次のイベントで使用できます。 ● authenticate_file_transfer ● authenticate_file_upload
filename	接続	filename パラメータは、次のイベントで使用できます。 ● authenticate_file_transfer ● authenticate_file_upload
file_size	接続	file_size パラメータは、次のイベントで使用できます。 ● authenticate_file_upload
filtered_rows	接続とテーブル	filtered_rows パラメータは、次のイベントで使用できます。 ● download_statistics
generation_number	接続	generation_number パラメータは、次のイベントで使用できます。 ● begin_publication ● end_publication
hashed_new_password	接続	hashed_new_password パラメータは、次のイベントで使用できます。 ● authenticate_user_hashed
hashed_password	接続	hashed_password パラメータは、次のイベントで使用できます。 ● authenticate_user_hashed
ignored_deletes	接続とテーブル	ignored_deletes パラメータは、次のイベントで使用できます。 ● upload_statistics

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
ignored_inserts	接続とテーブル	ignored_inserts パラメータは、次のイベントで使用できます。 ● upload_statistics
ignored_updates	接続とテーブル	ignored_updates パラメータは、次のイベントで使用できます。 ● upload_statistics
inserted_rows	接続とテーブル	inserted_rows パラメータは、次のイベントで使用できます。 ● upload_statistics
last_download	接続	last_download パラメータは、次のイベントで使用できます。 ● begin_download ● end_download ● modify_last_download_timestamp ● modify_next_last_download_timestamp ● nonblocking_download_ack ● prepare_for_download
last_publication_download	接続	last_publication_download パラメータは、次のイベントで使用できます。 ● begin_publication ● end_publication ● publication_nonblocking_download_ack
last_publication_upload	接続	last_publication_upload パラメータは、次のイベントで使用できます。 ● begin_publication ● end_publication

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
last_table_download	テーブル	last_table_download パラメータは、次のイベントで使用できます。 ● begin_download ● begin_download_deletes ● begin_download_rows ● download_cursor ● download_delete_cursor ● end_download ● end_download_deletes ● end_download_rows
maximum_time	接続とテーブル	maximum_time パラメータは、次のイベントで使用できます。 ● time_statistics
minimum_time	接続とテーブル	minimum_time パラメータは、次のイベントで使用できます。 ● time_statistics
new_password	接続	new_password パラメータは、次のイベントで使用できます。 ● authenticate_user
next_last_download	接続	next_last_download パラメータは、次のイベントで使用できます。 ● generate_next_last_download_timestamp ● modify_next_last_download_timestamp
number_of_calls	接続とテーブル	number_of_calls パラメータは、次のイベントで使用できます。 ● time_statistics
odbc_state	接続	odbc_state パラメータは、次のイベントで使用できます。 ● handle_odbc_error ● report_odbc_error

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
password	接続	password パラメータは、次のイベントで使用できます。 ● authenticate_user
publication_name	接続	publication_name パラメータは、次のイベントで使用できます。 ● begin_publication ● end_publication ● publication_nonblocking_download_ack

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
remote_id	接続とテーブル	remote_id パラメータは、次のイベントで使用できます。 ● authenticate_parameters (接続イベント) ● authenticate_user (接続イベント) ● authenticate_user_hashed (接続イベント) ● begin_download (接続イベントとテーブルイベント) ● begin_download_deletes (テーブルイベント) ● begin_download_rows (テーブルイベント) ● begin_publication (接続イベント) ● begin_synchronization (接続イベントとテーブルイベント) ● begin_upload (接続イベントとテーブルイベント) ● begin_upload_deletes (テーブルイベント) ● begin_upload_rows (テーブルイベント) ● download_cursor (テーブルイベント) ● download_delete_cursor (テーブルイベント) ● download_statistics (接続イベントとテーブルイベント) ● end_download (接続イベントとテーブルイベント) ● end_download_deletes (テーブルイベント) ● end_download_rows (テーブルイベント) ● end_publication (接続イベント) ● end_synchronization (接続イベントとテーブルイベント) ● end_upload (接続イベントとテーブルイベント) ● end_upload_deletes (テーブルイベント) ● end_upload_rows (テーブルイベント) ● generate_next_last_download_timestamp (接続) ● handle_error (接続イベント) ● handle_odbc_error (接続イベント) ● modify_error_message (接続イベント) ● modify_last_download_timestamp (接続イベント) ● modify_next_last_download_timestamp (接続イベント) ● modify_user (接続イベント) ● nonblocking_download_ack (接続イベント) ● prepare_for_download (接続イベント)

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
		● publication_nonblocking_download_ack (接続イベント) ● report_error (接続イベント) ● report_odbc_error (接続イベント) ● resolve_conflict (テーブルイベント) ● synchronization_statistics (接続イベントとテーブルイベント) ● time_statistics (接続イベントとテーブルイベント) ● upload_delete (テーブルイベント) ● upload_fetch (テーブルイベント) ● upload_fetch_column_conflict (テーブルイベント) ● upload_insert (テーブルイベント) ● upload_new_row_insert (テーブルイベント) ● upload_old_row_insert (テーブルイベント) ● upload_statistics (接続イベントとテーブルイベント) ● upload_update (テーブルイベント)
remote_key	接続	remote_key パラメータは、次のイベントで使用できます。 ● authenticate_file_transfer ● authenticate_file_upload

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
script_version	接続とテーブル	script_version パラメータは、次のイベントで使用できます。 ● authenticate_file_transfer (接続イベント) ● authenticate_file_upload (接続イベント) ● authenticate_parameters (接続イベント) ● authenticate_user (接続イベント) ● authenticate_user_hashed (接続イベント) ● begin_download (接続イベントとテーブルイベント) ● begin_download_deletes (テーブルイベント) ● begin_download_rows (テーブルイベント) ● begin_publication (接続イベント) ● begin_synchronization (接続イベントとテーブルイベント) ● begin_upload (接続イベントとテーブルイベント) ● begin_upload_deletes (テーブルイベント) ● begin_upload_rows (テーブルイベント) ● download_cursor (テーブルイベント) ● download_delete_cursor (テーブルイベント) ● download_statistics (接続イベントとテーブルイベント) ● end_download (接続イベントとテーブルイベント) ● end_download_deletes (テーブルイベント) ● end_download_rows (テーブルイベント) ● end_publication (接続イベント) ● end_synchronization (接続イベントとテーブルイベント) ● end_upload (接続イベントとテーブルイベント) ● end_upload_deletes (テーブルイベント) ● end_upload_rows (テーブルイベント) ● generate_next_last_download_timestamp (接続) ● handle_DownloadData (接続イベント) ● handle_error (接続イベント) ● handle_odbc_error (接続イベント) ● modify_error_message (接続イベント) ● modify_last_download_timestamp (接続イベント) ● modify_next_last_download_timestamp (接続イベント) ● modify_user (接続イベント)

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
		● nonblocking_download_ack (接続イベント) ● prepare_for_download (接続イベント) ● publication_nonblocking_download_ack (接続イベント) ● report_error (接続イベント) ● report_odbc_error (接続イベント) ● resolve_conflict (テーブルイベント) ● synchronization_statistics (接続イベントとテーブルイベント) ● time_statistics (接続イベントとテーブルイベント) ● upload_delete (テーブルイベント) ● upload_fetch (テーブルイベント) ● upload_fetch_column_conflict (テーブルイベント) ● upload_insert (テーブルイベント) ● upload_new_row_insert (テーブルイベント) ● upload_old_row_insert (テーブルイベント) ● upload_statistics (接続イベントとテーブルイベント) ● upload_update (テーブルイベント) **注意** スクリプトバージョンには、Java スクリプトおよび .NET スクリプトから、DBConnectionContext クラスの getVersion メソッドを使用してアクセスできます。DBConnectionContext.getVersion メソッド [Mobile Link サーバ Java]478 ページと DBConnectionContext.GetVersion メソッド [Mobile Link サーバ .NET]565 ページを参照してください。
subdir	接続	subdir パラメータは、次のイベントで使用できます。 ● authenticate_file_transfer ● authenticate_file_upload
subscription_id	接続	subscription_id パラメータは、次のイベントで使用できます。 ● begin_publication ● end_publication ● publication_nonblocking_download_ack

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
synchronization_ok	接続とテーブル	synchronization_ok パラメータは、次のイベントで使用できます。 ● end_synchronization
synchronized_tables	接続	synchronized_tables パラメータは、次のイベントで使用できます。 ● synchronization_statistics
table	接続とテーブル	table パラメータは、次のイベントで使用できます。 ● begin_download (テーブルイベント) ● begin_download_deletes (テーブルイベント) ● begin_download_rows (テーブルイベント) ● begin_synchronization (テーブルイベント) ● begin_upload (テーブルイベント) ● begin_upload_deletes (テーブルイベント) ● begin_upload_rows (テーブルイベント) ● download_statistics (テーブルイベント) ● end_download (テーブルイベント) ● end_download_deletes (テーブルイベント) ● end_download_rows (テーブルイベント) ● end_synchronization (テーブルイベント) ● end_upload (テーブルイベント) ● end_upload_deletes (テーブルイベント) ● end_upload_rows (テーブルイベント) ● handle_error (接続イベント) ● handle_odbc_error (接続イベント) ● report_error (接続イベント) ● report_odbc_error (接続イベント) ● resolve_conflict (テーブルイベント) ● synchronization_statistics (テーブルイベント) ● time_statistics (テーブルイベント) ● upload_statistics (テーブルイベント)
total_time	接続とテーブル	total_time パラメータは、次のイベントで使用できます。 ● time_statistics

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
updated_rows	接続とテーブル	updated_rows パラメータは、次のイベントで使用できます。 ● upload_statistics

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
username	接続とテーブル	username パラメータは、次のイベントで使用できます。 ● authenticate_file_transfer (接続イベント) ● authenticate_file_upload (接続イベント) ● authenticate_parameters (接続イベント) ● authenticate_user (接続イベント) ● authenticate_user_hashed (接続イベント) ● begin_download (接続イベントとテーブルイベント) ● begin_download_deletes (テーブルイベント) ● begin_download_rows (テーブルイベント) ● begin_publication (接続イベント) ● begin_synchronization (接続イベントとテーブルイベント) ● begin_upload (接続イベントとテーブルイベント) ● begin_upload_deletes (テーブルイベント) ● begin_upload_rows (テーブルイベント) ● download_cursor (テーブルイベント) ● download_delete_cursor (テーブルイベント) ● download_statistics (接続イベントとテーブルイベント) ● end_download (接続イベントとテーブルイベント) ● end_download_deletes (テーブルイベント) ● end_download_rows (テーブルイベント) ● end_publication (接続イベント) ● end_synchronization (接続イベントとテーブルイベント) ● end_upload (接続イベントとテーブルイベント) ● end_upload_deletes (テーブルイベント) ● end_upload_rows (テーブルイベント) ● generate_next_last_download_timestamp (接続イベント) ● handle_error (接続イベント) ● handle_odbc_error (接続エラー) ● modify_error_message (接続エラー) ● modify_last_download_timestamp (接続エラー) ● modify_next_last_download_timestamp (接続イベント) ● modify_user (接続イベント) ● nonblocking_download_ack (接続イベント)

システムパラメータ	イベントタイプ	イベントパラメータを使用できるイベント
		● prepare_for_download (接続イベント) ● publication_nonblocking_download_ack (接続イベント) ● report_error (接続イベント) ● report_odbc_error (接続イベント) ● resolve_conflict (テーブルイベント) ● synchronization_statistics (接続イベントとテーブルイベント) ● time_statistics (接続イベントとテーブルイベント) ● upload_delete (テーブルイベント) ● upload_fetch (テーブルイベント) ● upload_fetch_column_conflict (テーブルイベント) ● upload_insert (テーブルイベント) ● upload_new_row_insert (テーブルイベント) ● upload_old_row_insert (テーブルイベント) ● upload_statistics (接続イベントとテーブルイベント) ● upload_update (テーブルイベント)
warnings	接続とテーブル	warnings パラメータは、次のイベントで使用できます。 ● download_statistics ● synchronization_statistics ● upload_statistics

ユーザ定義の名前付きパラメータ

独自のパラメータを定義することもできます。独自のパラメータは、ユーザ定義の変数を使用できない RDBMS に特に便利です。

ユーザ定義のパラメータは、最初に参照されるときに定義され、NULL に設定されます。パラメータが参照専用 (入力専用) の場合は、パラメータに ui とピリオド (ui.) のプレフィクスを付ける必要があります。パラメータが更新される (in/out) 場合は、u とピリオド (u.) のプレフィクスを付ける必要があります。ユーザ定義のパラメータは 1 つの同期が終わるまで値が維持され、別の同期が開始されるときに NULL に設定されます。

ユーザ定義のパラメータは通常、テーブルに格納しないでステータス情報にアクセスするために使用します。テーブルに格納するには、複雑なジョインが必要です。

例

たとえば、var1 という変数を custom_value に設定する MyCustomProc というストアドプロシージャを作成するとします。

```
CREATE PROCEDURE MyCustomProc(
  IN username VARCHAR(128), INOUT var1 VARCHAR(128)
)
begin
 SET var1 = 'custom_value';
end
```

次の begin_synchronization スクリプトでは、ユーザ定義のパラメータ var1 を定義し、値を custom_value に設定しています。

```
CALL ml_add_connection_script (
  'version1',
  'begin_synchronization',
  '{call MyCustomProc( {ml s.username}, {ml u.var1} )}' );
```

次の download_cursor テーブルスクリプトでは、値が custom_value の var1 を参照しています。

```
CALL ml_add_table_script (
  'version1',
  'MyTable',
  'download_cursor',
  'select pk,col1 from MyTable where u_name = {ml s.username} and some_other_column = {ml
ui.var1}' );
```

最初のパラメータを in/out に定義する MyPFDProc という別のストアドプロシージャがあるとします。次の prepare_for_download スクリプトでは、var1 の値を pfd_value に変更しています。

```
CALL ml_add_connection_script (
  'version1',
  'prepare_for_download',
  '{call MyPFDProc( {ml u.var1} )}' );
```

次の begin_download スクリプトでは、値が pfd_value の var1 を参照しています。

```
CALL ml_add_connection_script (
  'version1',
  'begin_download',
  'insert into SomeTable values( {ml s.username}, {ml ui.var1} )' );
```

認証パラメータ

Mobile Link スクリプトでは、名前付きパラメータを使用して認証パラメータを指定できます。名前付きパラメータを使用する場合は、認証パラメータには、{ml a.1} のように先頭に文字 a のプレフィクスを付ける必要があります。パラメータは 1 から始まり、上限が 255 の数値です。各パラメータの最大値は 4000 バイトです。値は Mobile Link クライアントから送信されます。

authenticate_* スクリプトで使用すると、認証パラメータによって認証情報が渡されます。

認証パラメータは他のすべてのイベント (begin_connection と end_connection を除く) で Mobile Link クライアントからの情報を渡すために使用できます。この方法は、通常ならテーブルにローをアップロードする必要があるような処理を行うのに便利です。認証パラメータを使用すると、テーブルのアップロードイベントの前に値を使用できます。

SQL Anywhere リモートでは、dbmlsync -ap オプションを指定して情報を渡します。Ultra Light リモートでは、auth_parms と num_auth_parms を指定して情報を渡します。

参照

- 「スクリプトのパラメータ」234 ページ
- dbmlsync : 「-ap dbmlsync オプション」『Mobile Link クライアント管理』
- Ultra Light:「Authentication Parameters 同期パラメータ」『Ultra Light データベース管理とリファレンス』 および 「Number of Authentication Parameters パラメータ」『Ultra Light データベース管理とリファレンス』

例

Ultra Light リモートデータベースでは、ul_sync_info 構造体の num_auth_parms フィールドと auth_parms フィールドを使用して、パラメータを渡します。num_auth_parms は、パラメータの数で、0 ～ 255 の値になります。auth_parms は、文字列の配列へのポインタです。同期中、認証パラメータはパスワードと同じ方法で難読化されます。num_auth_parms が 0 の場合、auth_parms は NULL に設定します。次に、Ultra Light でパラメータを渡す例を示します。

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
  UL_TEXT( "param2" ), UL_TEXT( "param3" ) };

...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

SQL Anywhere のリモートデータベースでは、dbmlsync -ap オプションを使用して、認証パラメータをカンマで区切られたリストで渡します。たとえば、次のコマンドラインは 3 つのパラメータを渡します。

```
dbmlsync -ap "param1,param2,param3"
```

サーバでは、認証パラメータを送信された順序で参照します。この例では、authenticate_parameters スクリプトは以下のようになります。

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```

スクリプトバージョン

スクリプトは、「スクリプトバージョン」と呼ばれるグループに分けられます。スクリプトバージョンを指定することによって、アップロードの処理やダウンロードの準備に使用する同期スクリプトセットを Mobile Link クライアントで選択できます。

統合データベースにスクリプトバージョンを追加する方法については、「統合データベースへのスクリプトバージョンの追加」254 ページを参照してください。

スクリプトバージョンの使用方法

スクリプトバージョンを使用すると、スクリプトを異なる環境で実行されるスクリプトセットに編成できます。この機能によって柔軟性が生まれ、特に次のような場合に便利です。

- **アプリケーションのカスタマイズ**　さまざまなリモートユーザからの情報を処理するために、異なるスクリプトセットを使用します。たとえば、組織のマネージャが自分が持っているデータベースを同期させるときに使用するスクリプトセットを、他の人が使用するスクリプトセットとは別に作成できます。これと同じ機能を 1 つのスクリプトセットで実現することもできますが、より複雑なスクリプトになってしまいます。

- **アプリケーションのアップグレード**　データベースアプリケーションをアップグレードする場合、新しいスクリプトが必要になることがあります。これは、新しいバージョンのアプリケーションではデータ処理方法が異なる場合があるからです。リモートデータベースが変更される場合はほとんど、新しいスクリプトが必要になります。通常は、すべてのユーザを同時にアップグレードすることはできません。古いスクリプトと新しいスクリプトの両方がサーバ上で共存できるため、使用するアプリケーションのバージョンに関わらず、すべてのユーザが同期できます。

- **複数のアプリケーションの管理**　1 つの Mobile Link サーバが 2 つの完全に異なるアプリケーションを同期しなければならない場合があります。たとえば、販売アプリケーションを使用する従業員がいる一方で、在庫管理用に設計されたアプリケーションが必要な従業員もいます。2 つのアプリケーションで異なるデータセットが必要な場合は、各アプリケーションにつき 1 つのスクリプトバージョンとなるよう、2 つのバージョンの同期スクリプトを作成できます。

- **スクリプトバージョンのプロパティの設定**　.NET または Java 同期論理のクラスから、参照可能なスクリプトバージョンのプロパティを設定できます。 「ml_add_property システムプロシージャ」660 ページを参照してください。

スクリプトバージョン名の割り当て

スクリプトバージョン名は、文字列です。統合データベースにスクリプトを追加するときに、この名前を指定します。たとえば、ml_add_connection_script ストアドプロシージャと ml_add_table_script ストアドプロシージャを使用してスクリプトを追加する場合、スクリプトバージョン名が 1 つ目のパラメータになります。また、Sybase Central を使用してスクリプトを追加する場合は、スクリプトバージョン名を入力するように指示されます。

スクリプトバージョンには、**ml_sis_1** や **ml_qa_1** という名前は使用できません。これらの名前は、Mobile Link によって内部的に使用されています。

> **警告**
> スクリプトバージョンの名前は、**ml_** で始めないでください。ml_ で始まるスクリプトバージョンは内部用に予約されています。

同期のスクリプトバージョンの指定

同期が開始されるときにリモートサイトでスクリプトバージョンが指定されていない場合、同期は失敗します。 「スクリプトバージョンとサブスクリプション」『Mobile Link クライアント管理』を参照してください。

ml_global スクリプトバージョン

他のスクリプトバージョンとは使い方が異なる、**ml_global** と呼ばれるスクリプトバージョンを作成できます。ml_global というスクリプトバージョンを作成する場合、一度定義すると、関連

付けられた接続スクリプトがすべての同期で自動的に使用されます。同期クライアントから ml_global をスクリプトバージョンとして明示的に指定することはありません。

ml_global スクリプトバージョンでスクリプトを定義した後、同期対象として指定したスクリプトバージョンの同じイベントに対してスクリプトを定義した場合には、指定したスクリプトバージョンのスクリプトが使用されます。ml_global スクリプトバージョンのスクリプトが使用されるのは、同期中のプライマリスクリプトバージョンでそのスクリプトが定義されていない場合だけです。

ml_global スクリプトバージョンには、接続レベルスクリプトだけを含めることができます。スクリプトバージョンを 1 つのみ使用している場合、ml_global スクリプトバージョンはオプションとなり、役立たない可能性もあります。

統合データベースへのスクリプトバージョンの追加

スクリプトバージョンは、スクリプトセットを識別します。Sybase Central で作業している場合、統合データベースにスクリプトバージョン名を追加してから、接続スクリプトを追加してください。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

システムプロシージャを使ってスクリプトを追加する場合には、新しいスクリプトバージョン名を指定すると、そのバージョン名がスクリプトによって自動的に追加されます。

Sybase Central では、同期モデルごとに 1 つのスクリプトバージョンだけが使用可能で、デフォルトで同期モデルと同じ名前が付けられます。

同期しないでスキーマの変更を実行する場合は、SQL 構文を使用してスクリプトバージョンを同期サブスクリプションに追加する必要があります。

システムプロシージャを使用して、接続スクリプトまたはテーブルスクリプトを追加するのと同じ操作でスクリプトバージョンを追加できます。 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページを参照してください。

「スクリプトバージョン」252 ページを参照してください。

♦ タスク

1. **[ビュー]** メニューから **[フォルダ]** をクリックします。
2. Sybase Central の左ウィンドウ枠で、Mobile Link プロジェクト名を展開し、使用する統合データベースを展開します。統合データベースをプロジェクトに追加するときに入力した接続情報に基づいて、統合データベースに接続されます。
3. **[バージョン]** フォルダをクリックし、**[ファイル]** » **[新規]** » **[バージョン]** をクリックします。

4. **スクリプトバージョン作成ウィザード**の指示に従います。

結果

スクリプトバージョンが作成されます。

統合データベースからのスクリプトバージョンの削除

次のプロシージャを使用して、スクリプトバージョンと、スクリプトバージョンに関連付けられたスクリプトを統合データベースから削除します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. **[ビュー]** メニューから **[フォルダ]** をクリックします。
2. Sybase Central の左ウィンドウ枠で、Mobile Link プロジェクト名を展開し、使用する統合データベースを展開します。統合データベースをプロジェクトに追加するときに入力した接続情報に基づいて、統合データベースに接続されます。
3. 左ウィンドウ枠の統合データベースの下にある **[バージョン]** をクリックします。右ウィンドウ枠にスクリプトバージョンのリストが表示されます。
4. 右ウィンドウ枠で、削除するスクリプトバージョンを右クリックし、**[削除]** を選択します。
5. **[はい]** をクリックします。

> **注意**
> スクリプトバージョンに関連するすべてのスクリプトも削除されます。

結果

スクリプトバージョンと、スクリプトバージョンに関連付けられたスクリプトが統合データベースから削除されます。

必要なスクリプト

Mobile Link サーバを実行する場合には、特定のスクリプトが必要です。必要なスクリプトは、双方向同期、アップロード専用の同期、ダウンロード専用の同期のどれを行っているかによって決まります。

双方向またはアップロード専用の同期の場合、Mobile Link では次のテーブルスクリプトが必要です。

- upload_delete (SQL を使用して、削除されたローをアップロードする場合)
- upload_insert (SQL を使用して、挿入されたローをアップロードする場合)
- upload_update (SQL を使用して、更新されたローをアップロードする場合)
- または、ダイレクトローハンドリングでアップロードを処理している場合には、handle_UploadData 接続イベント用のスクリプトが必要。

双方向またはダウンロード専用の同期の場合、Mobile Link では同期のすべてのテーブルで download_cursor と download_delete_cursor の両方が必要です。または、ダイレクトローハンドリングでダウンロードを処理している場合には、handle_DownloadData 接続スクリプトを指定する必要があります。このスクリプトを空にして、他のイベントでダウンロードを処理できます。

必要なすべてのスクリプトを指定してください。必要なスクリプトがない場合、同期はアボートされます。無視するデータスクリプトがある場合は、プレフィクス --{ml_ignore} を使用します。 「無視されたスクリプト」259 ページを参照してください。

スクリプトの追加と削除

同期モデル作成ウィザードを使用すると、モデルの配備時にスクリプトが自動的に統合データベースに追加されます。

Sybase Central 以外で同期スクリプトを作成する場合は、統合データベース内の Mobile Link システムテーブルにそのスクリプトを追加してください。SQL スクリプトの場合には、スクリプト全体が Mobile Link システムテーブルに保存されます。Java または .NET スクリプトの場合には、メソッド名がシステムテーブルに登録されます。スクリプトの保存方法とメソッド名の保存方法は、ほぼ同じです。

「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

Sybase Central を使用している場合は、データベースにスクリプトバージョンを追加してから、個々のスクリプトを追加してください。 「統合データベースへのスクリプトバージョンの追加」254 ページを参照してください。

すべてのタイプのスクリプトの追加または削除 (システムプロシージャの場合)

統合データベースの設定時にインストールされるストアドプロシージャを使用して、スクリプトを統合データベースに追加、または統合データベースから削除できます。

スクリプトを追加または削除できるストアドプロシージャについては、以下の項目を参照してください。

- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ

接続スクリプトの追加

Sybase Central を使用して接続スクリプトを追加するには、次の手順を使用します。

前提条件

Sybase Central を使用している場合は、データベースにスクリプトバージョンを追加してから、個々のスクリプトを追加してください。「統合データベースへのスクリプトバージョンの追加」254 ページを参照してください。

♦ タスク

1. **[ビュー]** メニューから **[フォルダ]** をクリックします。
2. Sybase Central の左ウィンドウ枠で、Mobile Link プロジェクト名を展開し、使用する統合データベースを展開します。統合データベースをプロジェクトに追加するときに入力した接続情報に基づいて、統合データベースに接続されます。
3. **[接続スクリプト]** をクリックし、**[新規]** » **[接続スクリプト]** をクリックします。
4. **接続スクリプトの作成ウィザード**の指示に従います。

結果

接続スクリプトが作成されます。

接続スクリプトの削除

Sybase Central を使用して接続スクリプトを削除するには、次の手順を使用します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. **[ビュー]** メニューから **[フォルダ]** をクリックします。
2. Sybase Central の左ウィンドウ枠で、Mobile Link プロジェクト名を展開し、使用する統合データベースを展開します。統合データベースをプロジェクトに追加するときに入力した接続情報に基づいて、統合データベースに接続されます。
3. **[接続スクリプト]** を展開します。
4. 接続スクリプトを右クリックして、**[削除]** をクリックします。
5. **[はい]** をクリックします。

結果

接続スクリプトが削除されます。

テーブルスクリプトの追加

Sybase Central を使用してテーブルスクリプトを追加するには、次の手順を使用します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. **[ビュー]** メニューから **[フォルダ]** をクリックします。
2. Sybase Central の左ウィンドウ枠で、Mobile Link プロジェクト名を展開し、使用する統合データベースを展開します。統合データベースをプロジェクトに追加するときに入力した接続情報に基づいて、統合データベースに接続されます。
3. **[同期テーブル]** を展開します。
4. テーブルを右クリックして、**[新規]** » **[テーブルスクリプト]** をクリックします。
5. **テーブルスクリプト作成ウィザード**の指示に従います。

結果

テーブルスクリプトが作成されます。

テーブルスクリプトの削除

次の手順を使用して、テーブルスクリプトを削除します。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. **[ビュー]** メニューから **[フォルダ]** をクリックします。
2. Sybase Central の左ウィンドウ枠で、Mobile Link プロジェクト名を展開し、使用する統合データベースを展開します。統合データベースをプロジェクトに追加するときに入力した接続情報に基づいて、統合データベースに接続されます。
3. **[同期テーブル]** を展開します。
4. テーブルを展開します。
5. テーブルスクリプトを右クリックして、**[削除]** をクリックします。
6. **[はい]** をクリックします。

結果

テーブルスクリプトが削除されます。

スクリプトの直接挿入

ストアドプロシージャまたは Sybase Central を使用してスクリプトをシステムテーブルに挿入することをおすすめします。ただし、INSERT 文を使用してスクリプトを直接挿入することが必要な場合もあります。たとえば、一部の RDBMS のバージョンでは、長さの制限があって、ストアドプロシージャを使用するのが難しい場合があります。

Mobile Link システムテーブルの詳細については、「Mobile Link サーバのシステムテーブル」4 ページを参照してください。

スクリプトを直接挿入するのに必要な INSERT 文のフォーマットは、ストアドプロシージャ ml_add_connection_script と ml_add_table_script のソースコードにあります。これらのストアドプロシージャのソースコードは、Mobile Link 設定スクリプトにあります。サポートされている各 RDBMS 用に個別の設定スクリプトがあります。設定スクリプトはすべて *%SQLANY16%¥MobiLink¥Setup* にあります。ファイル名は次のとおりです。

統合データベース	設定ファイル
Adaptive Server Enterprise	*syncase.sql*
IBM DB2 LUW	*syncdb2.sql*
Microsoft SQL Server	*syncmss.sql*
MySQL	*syncmys.sql*
Oracle	*syncora.sql*
SQL Anywhere	*syncsa.sql*

無視されたスクリプト

統合データベースに upload_insert、upload_update、upload_delete スクリプトがないテーブルに対するデータの挿入、更新、削除がアップロードストリームに含まれる場合、またはテーブルのダウンロードスクリプト (download_cursor スクリプトと download_delete_cursor スクリプト) がない場合、Mobile Link サーバは見つからないスクリプトについてエラーを通知し、同期をアボートします。

Mobile Link サーバコマンドオプション -zwd を指定すると、この警告メッセージを抑制することができますが、このオプションにより、すべての同期テーブルに関する警告メッセージが抑制されてしまいます。

Mobile Link サーバは、プレフィクス --{ml_ignore} が含まれるすべての接続スクリプトやテーブルスクリプトについて、別の扱いをします。Mobile Link サーバでは、これらのスクリプトが意

図的に無視するスクリプトとして認識されます。より厳密には、upload_insert、upload_update、upload_delete スクリプトのある同期テーブルに対するデータの挿入、更新、削除がアップロードストリームに含まれていて、これらのスクリプトでプレフィクス --{ml_ignore} が指定されている場合、Mobile Link サーバは統合データベースに対してこれらのスクリプトを実行しません。さらに、エラーや警告のメッセージを表示しないで同期を続けます。アップロードされたローは無視されます。

テーブルをダウンロードする場合は、download_cursor スクリプトと download_delete_cursor スクリプトの両方を定義します。ローがダウンロードされないようにするには、これらのスクリプトの一方または両方を必要に応じて --{ml_ignore} として定義します。

ローをアップロードするスクリプト

Mobile Link サーバに、リモートデータベースから受信したアップロードデータを処理する方法を通知するには、アップロードスクリプトを定義します。リモートデータベースで更新、挿入、または削除するローを処理する個別のスクリプトを作成します。簡単な実装によって、対応するアクション (更新、挿入、削除) を統合データベースで実行できます。

Mobile Link サーバは、データを 1 つのトランザクションでアップロードします。アップロードプロセスの詳細については、「アップロード中のイベント」277 ページを参照してください。

Java または .NET 同期ロジックを使用してローをアップロードする方法については、「ローのアップロードとダウンロード」544 ページを参照してください。

注意

- 各リモートテーブルの begin_upload スクリプトと end_upload スクリプトには、更新される個々のローには依存しない論理が保持されます。
- アップロードは、1 ローずつの挿入、更新、削除から構成されます。通常、これらのアクションは、upload_insert、upload_update、upload_delete の各スクリプトを使用して実行されます。
- SQL Anywhere クライアント用のアップロードの準備において、dbmlsync ユーティリティは、正常に行われた最後の同期よりも後に書き込まれたすべてのトランザクションログにアクセスする必要があります。 「トランザクションログファイル」『Mobile Link クライアント管理』を参照してください。

upload_insert スクリプト

Mobile Link サーバは、アップロードの処理中にこのイベントを使用して、リモートデータベースに挿入されたローを処理します。

次の INSERT 文は、upload_insert スクリプトで使用されます。

```
INSERT INTO emp ( emp_id, emp_name )
VALUES ( { ml r.emp_id }, { ml r.emp_name } );
```

注意

- プレースホルダとして名前付きパラメータの代わりに疑問符を使用する場合、upload_new_row_insert イベントと upload_old_row_insert イベントは、remote_id と user_name を追加のパラメータとして受け入れます。これらのパラメータは、テーブルの完全なカラムリストの前に記述されている必要があります。

参照

- 「ローをアップロードするスクリプト」260 ページ
- 「upload_insert テーブルイベント」439 ページ

upload_update スクリプト

Mobile Link サーバは、アップロードの処理中にこのイベントを使用して、リモートデータベースで更新されたローを処理します。次の UPDATE 文は、emp テーブルの upload_update スクリプトとして使用できます。

```
UPDATE emp
SET emp_name = {ml r.emp_name}
WHERE emp_id = {ml o.emp_id};
```

注意

- プレースホルダとして名前付きパラメータの代わりに疑問符を使用する場合は、パラメータの数を次のいずれかにすることができます (SQL スクリプトで疑問符を使用することは推奨されなくなりました)。

  ○非プライマリキーカラムの数 + プライマリキーカラムの数。

  ○2 * (非プライマリキーカラムの数 + プライマリキーカラムの数)。

  カラム順序は、非プライマリキーカラムを先に、次のどちらかのカラムを後にして構成してください。

  ○プライマリキーカラム。

  ○すべてのカラム。

参照

- 「ローをアップロードするスクリプト」260 ページ
- 「upload_update テーブルイベント」455 ページ

upload_delete スクリプト

Mobile Link サーバは、アップロードの処理中にこのイベントを使用して、リモートデータベースから削除されたローを処理します。次の文は、upload_delete 文の使用方法を示しています。

```
DELETE FROM emp
WHERE emp_id = {ml r.emp_id};
```

注意

- プレースホルダとして名前付きパラメータの代わりに疑問符を使用する場合は、パラメータの数を次のいずれかにします (SQL スクリプトで疑問符を使用することは、推奨されなくなりました)。

  ○プライマリキーカラムの数。

  ○すべてのカラムの数。

参照

- 「ローをアップロードするスクリプト」260 ページ
- 「upload_delete テーブルイベント」433 ページ

upload_fetch スクリプト

upload_fetch スクリプトは、統合データベースのテーブルにカーソルを定義する SELECT 文です。このカーソルは、リモートデータベースから更新されたものとして受信したローの古い値と、統合データベースにある現在の値を比較するために使用します。これによって、upload_fetch スクリプトは更新の処理中に競合を識別します。

同期テーブルが、次のように定義されている場合を考えてみます。

```
CREATE TABLE uf_example (
 pk1 integer NOT NULL,
 pk2  integer NOT NULL,
 val  varchar(200),
 PRIMARY KEY( pk1, pk2 ));
```

この場合、このテーブルに対して考えられる upload_fetch スクリプトは次のようになります。

```
SELECT pk1, pk2, val
FROM uf_example
WHERE pk1 = {ml r.pk1} and pk2 = {ml r.pk2}
```

「upload_fetch テーブルイベント」435 ページを参照してください。

Mobile Link サーバが、統合データベース内の競合のチェック対象となる 1 つのローを正確に識別するには、upload_fetch スクリプト内にクエリの WHERE 句が必要となります。

ローをダウンロードするスクリプト

ダウンロードトランザクション時に各テーブルの処理に使用できるスクリプトは、2 つあります。挿入と更新を実行する download_cursor スクリプトと、削除を実行する download_delete_cursor スクリプトです。

これらのスクリプトは、SELECT 文か、結果セットを返すプロシージャの呼び出しのどちらかです。Mobile Link サーバは、スクリプトの結果セットをリモートデータベースにダウンロードします。Mobile Link クライアントは自動的に、download_cursor スクリプトの結果セットに基づいてローを挿入または更新し、download_delete_cursor イベントに基づいてローを削除します。

ストアドプロシージャの使用の詳細については、「ストアドプロシージャコールからの結果セット」137 ページを参照してください。

Mobile Link サーバは、データを 1 つのトランザクションでダウンロードします。 ダウンロードプロセスの詳細については、「ダウンロード中のイベント」280 ページを参照してください。

注意

- アップロードと同様、ダウンロードも接続イベントで開始、終了します。他のイベントは、テーブルレベルのイベントです。
- 各リモートテーブルの begin_download スクリプトと end_download スクリプトには、更新される個々のローには依存しない論理が保持されます。
- ダウンロードでは、挿入と更新が区別されません。download_cursor イベントに対応するスクリプトは、ダウンロードされるローを定義する SELECT 文です。クライアントは、ローが存在するかどうかを調べ、適切な挿入操作または更新操作を実行します。
- タイムスタンプベースのダウンロードの場合は、last_table_download パラメータを指定して、最後の同期以降の変更のみがダウンロードされるようにします。たとえば、download_cursor または download_delete_cursor SQL スクリプトには次の行を挿入できます。

  ```
  WHERE Customer.last_modified >= {ml s.last_table_download}
  ```

  「スクリプトでの最終ダウンロード時刻」109 ページを参照してください。
- 参照整合性違反を避けるために必要であれば、ダウンロード処理の最後にクライアントは自動的にローを削除します。
- SendDownloadAck 設定を ON に変更した場合、ダウンロードトランザクションはコミットされますが、確認スクリプトは確認が受信されるまで実行されません。

  デフォルトでは、SendDownloadAck は OFF に設定されています。

  「SendDownloadAck (sa) 拡張オプション」『Mobile Link クライアント管理』、「Send Download Acknowledgement 同期パラメータ」『Ultra Light データベース管理とリファレンス』、「nonblocking_download_ack 接続イベント」400 ページ、「publication_nonblocking_download_ack 接続イベント」406 ページを参照してください。

> **警告**
> 以前の展開によって作成されたシャドウテーブルを同期しないでください (たとえば、*_mod* または *_del* で終わるテーブルを同期しないでください)。これらのテーブルが必要になるのは、変更または削除されたローを統合データベースで追跡する場合のみです。

「参照整合性と同期」『Mobile Link クイックスタート』を参照してください。

download_cursor スクリプト

統合データベースからリモートデータベースにローをダウンロードするには、download_cursor スクリプトを作成します。同様に、リモートデータベースから削除するローをダウンロードするには、download_delete_cursor スクリプトを作成します。変更をダウンロードするリモートデータ

ベースのテーブルごとに、これらの両方のスクリプトを作成してください。他のスクリプトを使用するとダウンロード処理をカスタマイズできますが、それらは必要ありません。

- ローをダウンロードする各 download_cursor スクリプトには、SELECT 文か、SELECT 文を含むプロシージャの呼び出しが必要です。

- ローをダウンロードしない場合は、スクリプトを --{ml_ignore} として定義します。または、ml_add_missing_dnld_scripts システムプロシージャを使用して、見つからないダウンロードスクリプトを無視するものとして定義します。

- download_cursor スクリプトでは、リモートデータベース内の対応するテーブルにあるカラムに対応するすべてのカラムを選択する必要があります。統合データベース内のカラムは、対応するリモートデータベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

参照

- 「ml_add_missing_dnld_scripts システムプロシージャ」655 ページ

例

次のスクリプトは、従業員情報を格納しているリモートテーブルの download_cursor スクリプトとして機能します。このスクリプトによって、すべての従業員についての情報がダウンロードされます。

```
SELECT emp_id, emp_fname, emp_lname
FROM employee;
```

Mobile Link サーバは、スクリプトへ特定のパラメータを渡します。Mobile Link サーバは、パラメータ値に置き換えてから、統合データベースに対して文を実行します。疑問符の使用は、SQL スクリプトでは推奨されなくなりました。次のスクリプトは、名前付きパラメータの使用方法を示しています。

```
CALL ml_add_table_script(
  'Lab',
  'ULOrder',
  'download_cursor',
  'SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant, o.notes, o.status
      FROM ULOrder o
      WHERE o.last_modified >= {ml s.last_table_download}
          AND o.emp_name = {ml s.username}' )
```

注意

- カーソルスクリプトでは、リモートデータベースで定義されている順序に従ってカラムを選択してください。統合データベースでカラム名やテーブル構造が異なる場合、リモートデータベース (リファレンスデータベースと同等) に対して正しい順序でカラムを選択してください。カラムは、SELECT 文内の順序に基づいてリモートデータベース内のカラムに割り当てられます。

- ローの値は、単一のテーブルまたは複数のテーブル間のジョインから選択できます。

- リモートテーブル名は、統合データベースのテーブル名と同じである必要はありません。スクリプト自体にリモートテーブルの名前を入れる必要はありません。リモートテーブル名は、

Mobile Link システムテーブル ml_table 内のエントリによって示されます。Sybase Central では、リモートテーブルがそのスクリプトとともにリストされます。

参照

- 「download_cursor テーブルイベント」329 ページ
- 「リモートデータベース間でローを分割する」113 ページ
- 「download_delete_cursor スクリプト」265 ページ

download_delete_cursor スクリプト

リモートデータベースからローを削除するには、download_delete_cursor スクリプトを作成します。このスクリプトは、ダウンロードに関係するリモートデータベースの各テーブルに 1 つ作成してください。ローを削除しない場合は、各スクリプトを --{ml_ignore} として定義します。または、ml_add_missing_dnld_scripts システムプロシージャを使用して、見つからないダウンロードスクリプトを無視するものとして定義します。 「ml_add_missing_dnld_scripts システムプロシージャ」655 ページを参照してください。

統合データベースからのローの削除のみを実行し、リモートデータベースからローを消去することはできません。削除されたローのプライマリキーを download_delete_cursor で選択できるようにするため、そのローのプライマリキーを追跡する必要があります。こうするには、以下の 2 つの一般的な方法があります。

- **論理削除**　統合データベースのローを物理的に削除しないでください。その代わり、ローが有効であるかどうかを追跡するステータスカラムを使用します。このようにすると、download_delete_cursor を簡略化できます。ただし、ステータスカラムを認識して使用できるように、download_cursor とその他のアプリケーションを修正しなければならない場合があります。削除の時刻を保持する最終変更カラムが存在し、各リモートの最終ダウンロード時刻の追跡を行っている場合は、すべてのリモートのダウンロード時刻が削除の時刻よりも早ければ、ローを物理的に削除できます。

- **シャドウテーブル**　削除に関する追跡を実行する各テーブルには、テーブルのプライマリキーを保持するカラムとタイムスタンプを保持するカラムの 2 つを含むシャドウテーブルを作成します。ローが削除されたときにプライマリキーとタイムスタンプをシャドウテーブルに挿入するトリガを作成します。このようにすると、download_delete_cursor はこのシャドウテーブルから選択できるようになります。論理削除と同様に、すべてのリモートデータベースによって対応するデータがダウンロードされれば、シャドウテーブルからローを削除できます。

Mobile Link サーバは、統合データベースからプライマリキー値を選択し、それらをリモートデータベースに渡すことによって、リモートデータベースのローを削除します。値がリモートデータベース内のプライマリキーの値と一致する場合、そのローを削除します。

- 削除をダウンロードする各 download_delete_cursor スクリプトには、SELECT 文か、結果セットを返すストアドプロシージャの呼び出しが必要です。Mobile Link サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。

- download_delete_cursor で常にローを選択しない場合は、スクリプトを --{ml_ignore} として定義します。

- SELECT 文では、リモートデータベース内のテーブルのプライマリキーカラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモートデータベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

- 値は、対応するカラムがリモートデータベース内で定義されている順序に従って選択します。この順序は、テーブルの作成に使用される CREATE TABLE 文のカラム順と同じですが、プライマリキーを定義する文中のカラム順とは異なります。

- download_delete_cursor を使用してリモートデータベースの親レコードを削除すると、子レコードも自動的に削除されます。これは、Blackberry デバイスには当てはまりません。

  子レコードの削除の詳細については、「参照整合性と同期」『Mobile Link クイックスタート』を参照してください。

テーブルから全ローを削除

Mobile Link はすべての NULL を含むローの download_delete_cursor を検出すると、リモートデータベース内のデータをすべて削除します。download_delete_cursor 内の NULL の数は、プライマリキーカラムの数またはテーブル内のカラムの総数です。

たとえば、次の download_delete_cursor SQL スクリプトは、2 つのプライマリキーカラムがあるテーブル内のすべてのローを削除します。この例は、SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server のデータベースに適用されます。

```
SELECT NULL, NULL
```

IBM DB2 LUW と Oracle の統合データベースでは、ダミーテーブルを指定して NULL を選択してください。IBM DB2 LUW 9.5 と 9.7 では、次の構文を使用できます。

```
SELECT CAST( NULL AS INTEGER ), CAST( NULL AS INTEGER ) FROM SYSIBM.SYSDUMMY1
```

Oracle 統合データベースでは、次の構文を使用できます。

```
SELECT NULL, NULL FROM DUAL
```

参照

download_delete_cursor スクリプトの使用の詳細については、以下を参照してください。

- 「download_cursor テーブルイベント」329 ページ
- 「download_delete_cursor テーブルイベント」332 ページ
- 「削除」132 ページ
- 「削除同期の一時停止」133 ページ
- 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ SQL リファレンス』
- 「リモートデータベース間でローを分割する」113 ページ
- 「スナップショットを使った同期」112 ページ

例

次の例は、従業員情報を格納しているリモートテーブルに対する download_delete_cursor スクリプトです。Mobile Link サーバは、この SQL 文を使用して削除カーソルを定義します。このスクリプトは、スクリプトの実行時に統合データベースとリモートデータベースの両方に存在するすべての注文の情報を削除します。

```
SELECT order_id
FROM ULorder
```

download_delete_cursor は、パラメータ last_table_download と username を受け入れます。次のスクリプトは、各パラメータを使用して選択範囲を絞る方法を示しています。

```
SELECT order_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
 AND status = 'Approved'
 AND user_name = {ml s.username}
```

また、クライアントアプリケーションにロー自体を削除させるという方法もあります。この方法は、不必要なローをルールによって識別できる場合のみ可能です。たとえば、有効期限を示すタイムスタンプがローに含まれている場合があります。次回の同期時にこれらの削除がアップロードされるのを停止するには、STOP SYNCHRONIZATION DELETE 文を使用してからリモートでローを削除します。他の削除を通常の方法で同期する場合は、この後ですぐに START SYNCHRONIZATION DELETE を実行してください。

注意

- download_delete_cursor スクリプトには、プライマリキーカラムをリモートデータベースで定義されたときと同じ順序で含めてください。
- 親ローのみを削除して効率よくローを削除するには、すべての Mobile Link クライアントに組み込まれている参照整合性検査を使用します。「参照整合性と同期」『Mobile Link クイックスタート』を参照してください。

エラーを処理するスクリプト

Mobile Link サーバがスクリプト内のオペレーションを実行しているとき、そのオペレーションに失敗すると、同期スクリプトのエラーが発生します。SQL スクリプトの場合、DBMS は、エラーの内容を示す SQLCODE とエラーメッセージを Mobile Link サーバに返します。統合データベースの各 DBMS は、独自の SQLCODE とメッセージのセットを持っています。デフォルトでは、Mobile Link サーバは統合データベースのトランザクションをロールバックしてエラーのログを取り、同期をアボートします。

SQL データスクリプトの呼び出し中にエラーが発生すると、handle_error イベントまたは handle_odbc_error イベントが呼び出されます。これらのエラー処理スクリプトが定義されている場合、Mobile Link サーバはエラー処理スクリプトを呼び出し、エラーの性質とコンテキストについての情報を提供するいくつかのパラメータを渡します。その中の action_code というパラメータは、Mobile Link サーバにエラーへの対処方法を指示する出力値です。action_code は、エラーを無視するか、または同期をアボートするよう Mobile Link サーバに指示します。

エラー処理スクリプトは、すべての SQL エラーに対して呼び出されるわけでは「ありません」。エラー処理スクリプトは、データスクリプトによってのみ呼び出されます。データスクリプト以外でエラーが発生した場合、Mobile Link サーバは統合データベースのトランザクションをロールバックしてエラーのログを取り、同期をアボートします。

統合データベースの DBMS で例外処理がサポートされている場合は、エラー処理スクリプトの代わりにその例外処理を使用することを検討してください。特に、データスクリプトの特定のエ

ラーを無視する必要がある場合は、検討が必要です。一般的に、例外処理を使用すると、エラー処理スクリプトよりもパフォーマンスが向上します。

handle_error スクリプトまたは handle_odbc_error スクリプト自体でエラーが発生した場合、Mobile Link サーバは統合データベースのトランザクションをロールバックしてエラーのログを取り、同期をアボートします。

エラー処理アクション

エラー処理スクリプト内で指定できるアクションには次のものがあります。

●エラーを無視するが、監査テーブルにエラーのログを取る

●Mobile Link サーバに対して同期をロールバックするよう指示する

●電子メールで警告メッセージを送信する

1 つの SQL 文で複数のエラーが処理される場合

ODBC では、1 つの SQL 文につき複数のエラーの処理が可能で、RDBMS にはこの機能を利用しているものがあります。たとえば、Microsoft SQL Server では、1 文で 2 つのエラーが発生することがあります。1 番目は実際のエラーで、通常 2 番目は現在の文が終了した理由を説明する情報メッセージです。

1 つの SQL 文で複数のエラーが発生した場合、1 エラーにつき 1 つの handle_error スクリプトが呼び出されます。Mobile Link サーバは、最も厳しいアクションコード (つまり、一番大きな番号) を使用して、どのアクションを取るかを決定します。同じことが handle_error スクリプトにも適用されます。

handle_error スクリプト自体で SQL エラーが発生した場合は、デフォルトのアクションコード (3000) と見なされます。

参照

- 「handle_error 接続イベント」374 ページ
- 「handle_odbc_error 接続イベント」379 ページ
- 「report_error 接続イベント」409 ページ
- 「report_odbc_error 接続イベント」413 ページ

エラーレポート

デフォルトでは、エラーが発生すると統合データベースでロールバックが行われます。このロールバックにより、エラーとその解決方法のログを統合データベース内に作成することは困難です。report_error イベントと report_odbc_error イベントは、同期とは異なるデータベース接続で呼び出されるため、これらのイベントを使用すると、ユーザ定義のスクリプトのエラー関する正しいレコードを作成できます。これらのエラーレポートスクリプトは、エラー処理スクリプトの呼び出しの直後に呼び出され、そのすぐ後にコミットが実行されます。

エラーレポートスクリプトは、ユーザ定義のスクリプトで発生したすべての SQL エラーに対して呼び出されます。ユーザ定義のスクリプトでエラーが発生した場合、Mobile Link サーバは統

合データベースのトランザクションをロールバックしてエラーのログを取り、同期をアボートします。

統合データベースの DBMS で SQL からのアクティビティレポートについてアウトオブバンド (現在のデータベース接続外) メカニズムがサポートされている場合は、Mobile Link で定義されたエラーレポートスクリプトの代わりにそのメカニズムを使用することを検討してください。

次の項を参照してください。

- 「handle_error 接続イベント」374 ページ
- 「handle_odbc_error 接続イベント」379 ページ
- 「report_error 接続イベント」409 ページ
- 「report_odbc_error 接続イベント」413 ページ

例

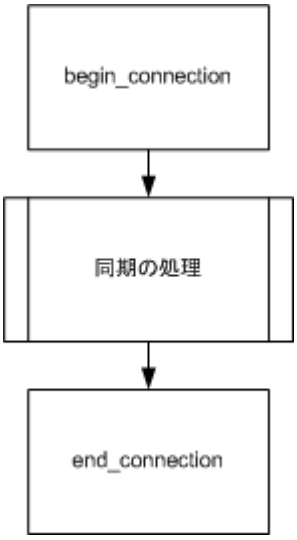
次の report_error スクリプトは 1 つの INSERT 文で構成されています。このスクリプトは、現在の日付と時刻とともにスクリプトのパラメータをテーブルに追加します。スクリプトはこの変更をコミットしません。通常、Mobile Link サーバが自動的に行ってくれるからです。

```
INSERT INTO errors
VALUES(
  CURRENT DATE,
  {ml s.action_code},
  {ml s.error_code},
  {ml s.error_message},
  {ml s.username},
  {ml s.table} );
```

同期イベント

Mobile Link イベントの概要

同期要求が発生し、Mobile Link サーバが、新しい統合データベース接続を作成することを決定すると、begin_connection イベントが呼び出され、同期が始まります。



同期に続いて、統合データベース接続が接続プールに配置され、再度 Mobile Link は同期要求を待ちます。同じバージョンに対する同期要求を再び受信した場合、Mobile Link はその次に受けた同期要求を同じ接続で処理します。end_connection イベントが呼び出された後、接続は最終的に接続プールから削除されます。

各同期には多くのイベントがあります。ほとんどのイベントは、そのイベントを含むトランザクションで分類されます。

トランザクション

各同期内で、次のトランザクションが発生する可能性があります。

- 認証
- 同期の開始
- アップロード
- ダウンロードの準備
- ダウンロード
- 同期の終了
- 非ブロッキングダウンロード確認

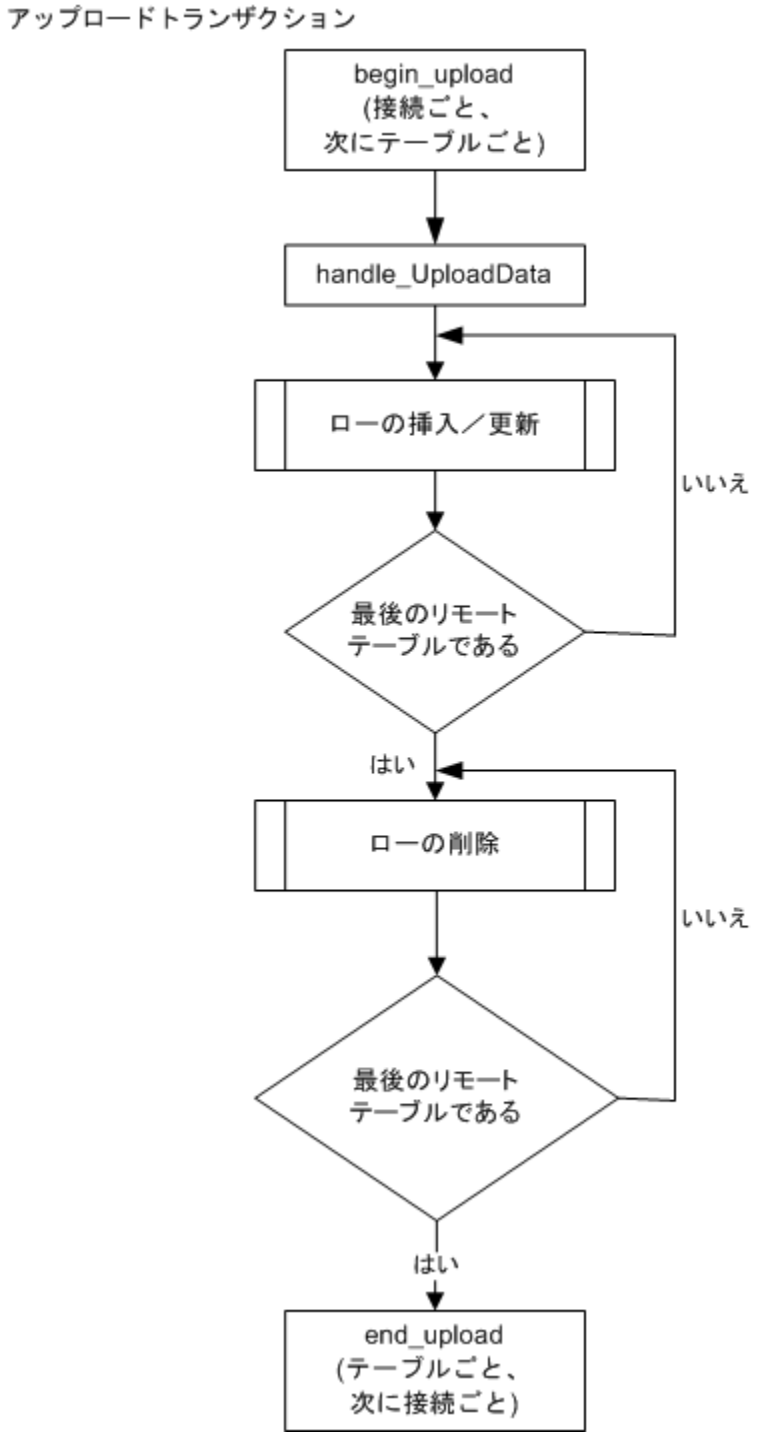
さらに、2 つの接続トランザクションが含まれることがあります。接続の開始トランザクションは、統合データベース接続が確立された直後に発生し、接続の終了トランザクションは、接続が終了したときに発生します。

同期の主要段階は、アップロードトランザクションとダウンロードトランザクションです。アップロードとダウンロードのトランザクションに含まれるイベントについては、以下で概説します。

アップロードトランザクション

アップロードトランザクションは、リモートデータベースからアップロードされた変更を適用します。

begin_upload イベントは、アップロードトランザクションの開始にマークを付けます。アップロードトランザクション処理は 2 段階になっています。まず、すべてのリモートテーブルに対する挿入と更新がアップロードされ、次にすべてのリモートテーブルに対する削除がアップロードされます。



end_upload イベントは、アップロードトランザクションの終了にマークを付けます。

dbmlsync -tu オプションを使用して複数のアップロードトランザクションを指定できます。

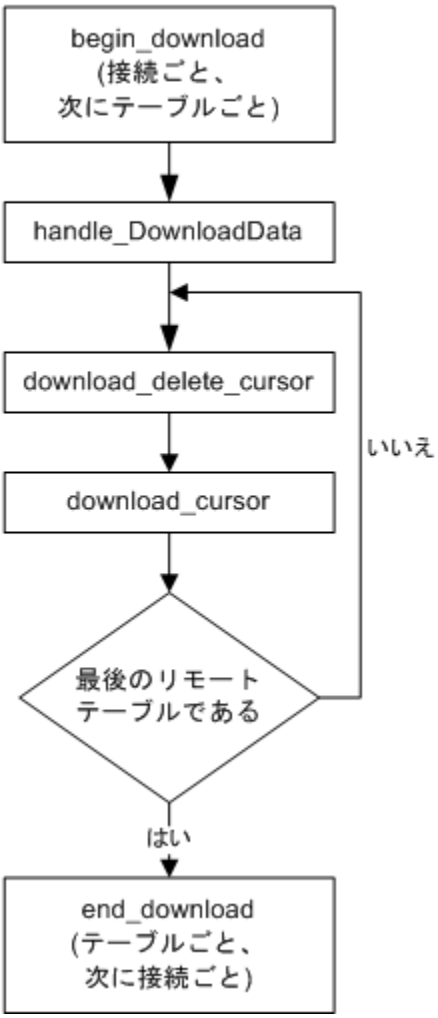
「ローをアップロードするスクリプト」260 ページを参照してください。

ダウンロードトランザクション

ダウンロードトランザクションは、統合データベースからローをフェッチします。ダウンロードトランザクションは、begin_download イベントで始まります。

ダウンロードトランザクション処理は 2 段階になっています。まず最初に各テーブルに対する削除データがダウンロードされ、次に更新／挿入ロー (アップサート) がダウンロードされます。end_download イベントによって、ダウンロードトランザクションが終了します。

ダウンロードトランザクション



「ローをダウンロードするスクリプト」262 ページを参照してください。

非ブロッキングダウンロード確認トランザクション

非ブロッキングダウンロード確認トランザクションは、ダウンロード確認が受信された場合にのみ実行されます。このトランザクションには、2 つの目的があります。このトランザクションでは、publication_nonblocking_download_ack スクリプトと nonblocking_download_ack スクリプトが実行されます。これらのスクリプトは、ダウンロードステータスの追跡を容易にします。次に、このトランザクション中に Mobile Link システムテーブルのダウンロードタイムスタンプが更新されます。

このトランザクションは、ターゲット同期に対する他のイベントと同じデータベース接続で実行されない場合があります。この場合、このトランザクションでは接続レベルの変数が参照されません。

擬似コード内のイベントの概要

次の疑似コードは、イベント (イベントと同名のスクリプト) が呼び出される順序の概要を示します。この Mobile Link イベントモデルの説明は、エラーのない完全な同期 (アップロード専用またはダウンロード専用ではない) を想定しています。

注意

- 通常、指定したイベントのスクリプトを定義していないと、デフォルトアクションは何も実行しません。
- begin_connection イベントと end_connection イベントは、「接続レベルのイベント」です。これらのイベントは特定の同期に依存せず、パラメータがありません。
- 各テーブルが同期されるたびに 1 回ずつ呼び出されるイベントもあります。これらのイベントに対応するスクリプトは、「テーブルレベルのスクリプト」と呼ばれます。

  各テーブルは専用のテーブルスクリプトを持つことができますが、いくつかのテーブルで共有されるテーブルレベルのスクリプトを作成することもできます。
- begin_synchronization など、一部のイベントは接続レベルとテーブルレベルの両方で発生します。これらのイベントに対しては、接続スクリプトとテーブルスクリプトの両方を作成できます。
- 同期処理がどのように複数のトランザクションに分散されるかについては、COMMIT 文が参考になります。

警告
SQL 同期スクリプト、または SQL 同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。SQL スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、Mobile Link では、障害が発生した場合にデータの整合性を保証できません。

Mobile Link の完全なイベントモデル

MobiLink complete event model.

```
Legend:
- // This is a comment.
- <name>
    The pseudocode for <name> is listed separately
    in a later section, under a banner:
        ------------------------
        name
        ------------------------
- VariableName <- value
    Assign the given value to the given variable name.
    Variable names are in mixed case.
- event_name
    If you have defined a script for the given event name,
    it is invoked.
-------------------------------------------------------

CONNECT to consolidated database
begin_connection_autocommit
begin_connection
COMMIT
for each synchronization request with
     the same script version {
  <synchronize>
}
end_connection
COMMIT
DISCONNECT from consolidated database

-------------------------------------------------------
synchronize
-------------------------------------------------------

<authenticate>
<begin_synchronization>
<upload>
<prepare_for_download>
<download>
<end_synchronization>

-------------------------------------------------------
authenticate
-------------------------------------------------------

Status <- 1000
UseDefaultAuthentication <- TRUE
if( authenticate_user script is defined ) {
  UseDefaultAuthentication <- FALSE
  TempStatus <- authenticate_user
  if( TempStatus > Status ) {
    Status <- TempStatus
  }
}
if( authenticate_user_hashed script is defined ) {
  UseDefaultAuthentication <- FALSE
  TempStatus <- authenticate_user_hashed
  if( TempStatus > Status ) {
    Status <- TempStatus
  }
}
  if( authenticate_parameters script is defined )
 {
    TempStatus <- authenticate_parameters
    if( TempStatus > Status ) {
```

```
      Status <- TempStatus
  }
if( UseDefaultAuthentication ) {
  if( the user exists in the ml_user table ) {
    if( ml_user.hashed_password column is not NULL ) {
      if( password matches ml_user.hashed_password ) {
        Status <- 1000
      } else {
        Status <- 4000
      }
    } else {
      Status <- 1000
    }
  } else if( -zu+ was on the command line ) {
    Status <- 1000
  } else {
    Status <- 4000
  }
}
if( Status >= 3000 ) {
  // Abort the synchronization.
} else {
  // UserName defaults to MobiLink user name
  // sent from the remote.
  if( modify_user script is defined ) {
    UserName <- modify_user
    // The new value of UserName is later passed to
    // all scripts that expect the MobiLink user name.
  }
}
COMMIT

-----------------------------------------------------
begin_synchronization
-----------------------------------------------------

begin_synchronization   // Connection event.
for each table being synchronized {
    begin_synchronization    // Call the table level script.
}
for each publication being synchronized {
  begin_publication
}
COMMIT

-----------------------------------------------------
end_synchronization
-----------------------------------------------------

for each publication being synchronized {
  if( begin_publication script was processed ) {
    end_publication
  }
}
for each table being synchronized {
  if( begin_synchronization table script was processed ) {
    end_synchronization // Table event.
  }
}
if( begin_synchronization connection script was processed ) {
  end_synchronization     // Connection event.
}
for each table being synchronized {
synchronization_statistics // Table event.
```

```
}
synchronization_statistics // Connection event.
for each table being synchronized {
  time_statistics // Table event.
}
time_statistics // Connection event.

COMMIT
```

アップロード処理の詳細については、「アップロード中のイベント」277 ページを参照してください。

ダウンロード処理の詳細については、「ダウンロード中のイベント」280 ページを参照してください。

アップロード中のイベント

次の擬似コードは、アップロードイベントとアップロードスクリプトがどのように呼び出されるかを示します。擬似コードは次の規則に従っています。

- **スクリプトは実際のスクリプトとして定義されている**　つまり、スクリプトは同期中に統合データベースに対して実行される実際のスクリプトとして定義されています。
- **スクリプトは無視されるスクリプトとして定義されている**　つまり、スクリプトは "--{ml_ignore}" を使用して無視されるスクリプトとして定義されています。
- **スクリプトは定義済みである**　つまり、スクリプトは実際のスクリプトまたは無視されるスクリプトとして定義されています。
- **スクリプトは未定義である**　つまり、イベントに対して定義されているスクリプトはありません。必要なスクリプトであるが、そのスクリプトを使用する必要がない場合は、そのスクリプトを無視されるスクリプトとして定義する必要があります。

これらのイベントは、完全なイベントモデルのアップロードのロケーションで発生します。「Mobile Link イベントの概要」269 ページを参照してください。

アップロードの概要

```
-----------------------------------------------------
upload
-----------------------------------------------------

begin_upload // Connection event
for each table being synchronized {
   begin_upload // Table event
}
handle_UploadData
for each table being synchronized {
   begin_upload_rows
   for each uploaded INSERT or UPDATE for this table {
   if( INSERT ) {
      <upload_inserted_row>
   }
   if( UPDATE ) {
      <upload_updated_row>
   }
```

```
    }
    end_upload_rows
}
for each table being synchronized IN REVERSE ORDER {
    begin_upload_deletes
    for each uploaded DELETE for this table {
    <upload_deleted_row>
    }
    end_upload_deletes
}
For each table being synchronized {
    if( begin_upload table script was processed ) {
    end_upload // Table event
    }
}
if( begin_upload connection script was processed ) {
    end_upload // Connection event
}
for each table being synchronized {
    upload_statistics  // Table event.
}
upload_statistics  // Connection event.

COMMIT
```

挿入のアップロード

```
-----------------------------------------------------
<upload_inserted_row>
-----------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.

if( upload_insert script is real ) {
    upload_insert
} else if( handle_uploadData script is real or
    upload_insert script is defined as an ignored script ) {
    // Ignore the insert. (Only ignored in SQL, possibly handled by handle_uploadData.)
} else {
    error
}
```

更新のアップロード

```
-----------------------------------------------------
upload_updated_row
-----------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.
// - Both the old (original) and new rows are uploaded for
//   each update.

ConflictsAreExpected <- (
    upload_new_row_insert script is defined or
    upload_old_row_insert script is defined )
Conflicted <- FALSE

if( upload_update script is real ) {
    if( upload_fetch or upload_fetch_column_conflict script is real ) {
        if( ConflictsAreExpected ) {
            FETCH using upload_fetch INTO current_row
            if( current_row <> old row ) {
                Conflicted <- TRUE
            } else {
```

```
                upload_update
            }
        } else {
            error
        }
    } else if( upload_fetch and upload_fetch_column_conflict scripts are not defined ) {
        if( ConflictsAreExpected ) {
        error
        } else {
            // No conflict detection and resolution by the MobiLink server
            // The upload_update script should handle conflict detection and resolution
            upload_update
        }
    } else {
    // the upload_fetch script cannot defined as an ignored script
    error
    }
} else if( handle_uploadData script is defined or upload_update script is defined as an ignored script ) {
    // Ignore the upload update (Only ignored in SQL, possibly handled by handle_uploadData.)
} else {
    error
}
if( Conflicted ) {
    if( upload_old_row_insert script is real ) {
        upload_old_row_insert
    } else if( upload_old_row_insert script is defined as ignored script ) {
        // Ignore the old value
    } else {
        error
    }
    if( upload_new_row_insert script is real ) {
        upload_new_row_insert
    } else if( upload_new_row_insert script is defined as ignored script ) {
        // Ignore the new value
    } else {
        error
    }
    if( no error ) {
        resolve_conflict
    }
}
```

削除のアップロード

```
-----------------------------------------------------
upload_deleted_row
-----------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.

if( upload_delete is real ) {
    upload_delete
} else if( handle_UploadData script is real or
    upload_delete script is defined as an ignored script ) {
    // Ignore this delete. (Only ignored in SQL, possibly handled by handle_uploadData.)
} else {
    error
}
```

ダウンロード中のイベント

次の疑似コードは、ダウンロードイベント (イベントと同名のスクリプト) が呼び出される順序の概要を示します。

これらのイベントは、「Mobile Link イベントの概要」269 ページに示す完全なイベントモデルのダウンロードのロケーションで発生します。

```
-----------------------------------------------------
prepare_for_download
-----------------------------------------------------

generate_next_last_download_timestamp
modify_last_download_timestamp
fetch the next download timestamp from consolidated
prepare_for_download

-----------------------------------------------------
download
-----------------------------------------------------

begin_download // Connection event.
for each table being synchronized {
  begin_download // Table event.
}
  handle_DownloadData
  for each table being synchronized {
    begin_download_deletes
    for each row in download_delete_cursor {
      if( all primary key columns are NULL ) {
        send TRUNCATE to remote
      } else {
        send DELETE to remote
      }
    }
    end_download_deletes
    begin_download_rows
    for each row in download_cursor {
      send INSERT ON EXISTING UPDATE to remote
    }
    end_download_rows
  }
  modify_next_last_download_timestamp
  for each table being synchronized {
    if( begin_download table script was processed ) {
      end_download // Table event
    }
}
if( begin_download connect script was processed ) {
  end_download // Connection event
}
  for each table being synchronized {
    download_statistics   // Table event.
  }
    download_statistics   // Connection event.

COMMIT
```

注意

- ダウンロードストリームでは、挿入と更新が区別されません。download_cursor イベントに対応するスクリプトは、ダウンロードされるローを指定する SELECT 文です。クライアントは、ローが存在するかどうかを調べ、適切な挿入操作または更新操作を実行します。

- ダウンロード処理の最後に、クライアントは自動的に参照整合性に違反するローを削除します。

  「参照整合性と同期」『Mobile Link クイックスタート』を参照してください。

データスクリプト

ローデータを直接処理するスクリプトは、データスクリプトと呼ばれます。その他のすべてのスクリプトは非データスクリプトです。データスクリプトと非データスクリプトの区別が重要になる場合があります。たとえば、カラム値の名前付きパラメータを参照できるのはデータスクリプトのみです。

次のイベントには、データスクリプトが関連付けられています。

- 「download_cursor テーブルイベント」
- 「download_delete_cursor テーブルイベント」
- 「handle_UploadData 接続イベント」
- 「handle_DownloadData 接続イベント」
- 「upload_delete テーブルイベント」
- 「upload_fetch テーブルイベント」
- 「upload_fetch_column_conflict テーブルイベント」
- 「upload_insert テーブルイベント」
- 「upload_new_row_insert テーブルイベント」
- 「upload_old_row_insert テーブルイベント」
- 「upload_update テーブルイベント」

SQL を返す Java と .NET のデータスクリプト (削除)

バージョン 16 から、Mobile Link サーバによって SQL スクリプトとして解釈される文字列を返す Java と .NET のスクリプトロジックの機能は、すべてのデータスクリプトで削除されています。スクリプトによって統合データベースに変更を加える必要がある場合は、Java または .NET からこのことを直接実行する必要があります。

次に、スクリプトの更新の例を示します。最初の例では SQL を使用し、2 つ目の例では SQL を使用していません。

```
public String beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException
{
    doSomeWork( ts, user );
    return( "CALL do_some_sql( {ml s.last_download}, {ml s.username} )" );
}

public void beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException
{
    doSomeWork( ts, user );

    Connection conn = DBConnectionContext.getConnection();
    PreparedStatement stmt = conn.prepareStatement( "CALL do_some_sql( ?,? )" );
    stmt.setTimestamp( 1, ts );
    stmt.setString( 2, user );
    stmt.executeUpdate();
}
```

authenticate_file_transfer 接続イベント

mlfiletransfer ユーティリティまたは MLFileDownload メソッドを使用してファイル転送のカスタム認証を実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.file_authentication_code	INTEGER。必須。これは INOUT パラメータです。認証の全体の成功状況を示します。 この値が 1000 ～ 1999 の場合、ファイル転送は許可されます。この値が 2000 ～ 2999 の場合、ファイル転送は許可されません。	1
s.filename	VARCHAR(128)。必須。認証の対象となる、転送されているファイルの名前を示す INOUT パラメータです。パスを含めたり、省略記号 (ピリオド 3 つ)、スラッシュ (/)、円記号 (¥) を使用したりしないでください。ファイルは、mlsrv16 -ftr または -ftru オプションを使用して指定したルート転送ディレクトリまたは自動的に作成されるサブフォルダに配置してください。これを明示的に指定しない場合、値は、デフォルトでクライアントによって Mobile Link サーバに渡されたファイル名になります。	2
s.username	VARCHAR(128)。Mobile Link ユーザ名。	3

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.subdir	VARCHAR(128)。このオプションの INOUT パラメータは、転送するファイルのサブフォルダのロケーションを設定します。ルートディレクトリを使用する場合は、このオプションを NULL に設定します。このオプションに省略記号 (ピリオド 3 つ)、カンマ、スラッシュ (/)、円記号 (¥) を含めないでください。このパラメータを明示的に設定しない場合、値は、デフォルトで remote_key になります。	適用外
s.remote_key	VARCHAR(128)。ファイル転送用のリモートキーを指定するオプションの IN パラメータです。	適用外
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

備考

Mobile Link サーバはこのイベントを実行してから、mlfiletransfer ユーティリティまたは MLFileDownload メソッドを使用したダウンロードファイル転送を許可します。ユーザが通常の認証を使用して認証した後、このイベントが実行されます。このスクリプトが定義されていない場合、ファイル転送は許可されます。

MLFileDownload メソッドは Ultra Light クライアントだけで使用できます。

参照

- 「スクリプトの追加と削除」256 ページ
- 「-ftr mlsrv16 オプション」62 ページ
- 「-ftru mlsrv16 オプション」63 ページ
- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」『Mobile Link クライアント管理』
- MLFileDownload メソッド [Ultra Light Embedded SQL]『Ultra Light C/C++ プログラミング』
- Ultra Light： 「Mobile Link ファイル転送」『Ultra Light データベース管理とリファレンス』
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

authenticate_file_upload 接続イベント

mlfiletransfer ユーティリティまたは MLFileUpload メソッドを使用してファイル転送のカスタム認証を実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.file_authentication_code	INTEGER。必須。これは INOUT パラメータです。認証の全体の成功状況を示します。 この値が 1000 ～ 1999 の場合、ファイル転送は許可されます。この値が 2000 ～ 2999 の場合、ファイル転送は許可されません。	1

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.filename	VARCHAR(128)。必須。認証の対象となる、転送されているファイルの名前を示す INOUT パラメータです。パスを含めたり、省略記号 (ピリオド 3 つ)、スラッシュ (/)、円記号 (¥) を使用したりしないでください。ファイルは、mlsrv16 -ftr または -ftru オプションを使用して指定したルート転送ディレクトリまたは自動的に作成されるサブフォルダに配置してください。これを明示的に指定しない場合、値は、デフォルトでクライアントによって Mobile Link サーバに渡されたファイル名になります。	2
s.file_size	INTEGER。このオプションの IN パラメータは、アップロード可能なファイルのサイズを制限するために使用できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	3
s.subdir	VARCHAR(128)。このオプションの INOUT パラメータは、転送するファイルのサブフォルダのロケーションを設定します。ルートディレクトリを使用する場合は、このオプションを NULL に設定します。このオプションに省略記号 (ピリオド 3 つ)、カンマ、スラッシュ (/)、円記号 (¥) を含めないでください。このパラメータを明示的に設定しない場合、値は、デフォルトで remote_key になります。	適用外

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_key	VARCHAR(128)。ファイル転送用のリモートキーを指定するオプションの IN パラメータです。	適用外
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

備考

Mobile Link サーバはこのイベントを実行してから、mlfiletransfer ユーティリティまたは MLFileUpload メソッドを使用したダウンロードファイル転送を許可します。ユーザが通常の認証を使用して認証した後、このイベントが実行されます。このスクリプトが定義されていない場合、ファイル転送は許可されます。

MLFileUpload メソッドは Ultra Light クライアントだけで使用できます。

参照

- 「スクリプトの追加と削除」256 ページ
- 「-ftr mlsrv16 オプション」62 ページ
- 「-ftru mlsrv16 オプション」63 ページ
- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」『Mobile Link クライアント管理』
- MLFileUpload メソッド [Ultra Light Embedded SQL]『Ultra Light C/C++ プログラミング』
- Ultra Light： 「Mobile Link ファイル転送」『Ultra Light データベース管理とリファレンス』
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

authenticate_parameters 接続イベント

ユーザ ID とパスワード以外の認証に使用できる、リモートデータベースからの値を受信します。この値を使用して、各同期を任意にカスタマイズすることもできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラ

メータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.authentication_status	INTEGER。これは INOUT パラメータです。	1
s.authentication_message	VARCHAR(1024)。これは INOUT パラメータです。認証メッセージを提供します。	適用外
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
a.*N* (1 つ以上)	VARCHAR(4000)。たとえば、a.1 a.2 のような名前付きパラメータを指定できます。	3 以上

パラメータの説明

- **authentication_status** authentication_status パラメータは必須です。認証の全体の成功状況を示します。次のいずれかの値に設定されます。

戻り値	authentication_status	説明
V <= 1999	1000	認証に成功しました。
1000 <= V <= 2999	2000	認証に成功しましたが、パスワードの有効期限がもうすぐ切れます。

戻り値	authentication_status	説明
2000 <= V <= 3999	3000	認証に失敗しました。パスワードの有効期限が切れています。
4000 <= V <= 4999	4000	認証に失敗しました。
5000 <= V <= 5999	5000	リモート ID がすでに使用されているため、認証できません。後で同期を再試行してください。
6000 <= V	4000	戻り値が 5999 より大きい場合、その値は 4000 (認証に失敗) として解釈されます。

- **authentication_message**　このオプションのパラメータは、認証メッセージを提供します。

  この名前のパラメータは、ユーザ認証スクリプトが初回に使用する前に NULL に初期化されます。スクリプトがこの名前のパラメータを使用する場合、返されるメッセージは次のユーザ認証スクリプトに渡されます。最終メッセージは、リモートデータベースの文字セットに変換されます。

  ユーザ認証スクリプトの実行時にエラーが発生しなかった場合は、ユーザ認証ステータスに関係なく、アップロードストリームの処理前に Mobile Link サーバによってこのメッセージがクライアントに送信されます。

  このメッセージは、ユーザ認証が失敗した場合でもクライアントに送信されます。

- **remote_ID**　Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。

  「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』を参照してください。

- **script_version**　このオプションのパラメータは、Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定します。このパラメータの指定に疑問符を使用することはできません。

- **username**　このパラメータは、Mobile Link ユーザ名です。VARCHAR(128)。

- **remote a.N**　リモートクライアントから送信された N 番目の認証パラメータ。

備考

リモートパラメータの数が authenticate_parameters スクリプトに必要な数と一致しないと、エラーの原因となります。また、パラメータがクライアントから送信されたときにこのイベントのスクリプトがない場合にもエラーが発生します。

SQL Anywhere のクライアントからも Ultra Light のクライアントからも、文字列またはパラメータを文字列の形式で送信できます。これにより、ユーザ ID とパスワード以外の認証も可能にな

ります。また、パラメータの値に基づいて同期をカスタマイズでき、それも前同期フェーズで認証中にカスタマイズできます。これらのパラメータは、同期スクリプトから参照される場合もあります。

Mobile Link 同期サーバは、各同期の開始時にこのイベントを実行します。このイベントは、authenticate_user イベントと同じトランザクションで実行されます。

このイベントを使用して、組み込み Mobile Link 認証メカニズムを、カスタムメカニズムに置き換えられます。使用している DBMS の認証メカニズムを使用したり、Mobile Link 組み込みメカニズムには存在しない機能を実装したりできます。

authenticate_user スクリプトまたは authenticate_user_hashed スクリプトが呼び出されてエラーを返すと、このイベントは呼び出されません。

authenticate_parameters イベント用の SQL スクリプトは、ストアドプロシージャとして実装してください。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「認証パラメータ」251 ページ
- 「Mobile Link ユーザ」『Mobile Link クライアント管理』
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「カスタムユーザ認証」『Mobile Link クライアント管理』
- 「authenticate_user 接続イベント」291 ページ
- 「authenticate_user_hashed 接続イベント」297 ページ
- 「begin_synchronization 接続イベント」314 ページ
- dbmlsync： 「-ap dbmlsync オプション」『Mobile Link クライアント管理』
- Ultra Light: 「Authentication Parameters 同期パラメータ」『Ultra Light データベース管理とリファレンス』 および 「Number of Authentication Parameters パラメータ」『Ultra Light データベース管理とリファレンス』

例

Ultra Light リモートデータベースでは、ul_sync_info 構造体の num_auth_parms フィールドと auth_parms フィールドを使用して、パラメータを渡します。num_auth_parms は、パラメータの数で、0 ～ 255 の値になります。auth_parms は、文字列の配列へのポインタです。文字列がプレーンテキストとして表示されるのを防ぐため、文字列はパスワードと同じように難読化されて送信されます。num_auth_parms が 0 の場合、auth_parms は NULL に設定します。次に、Ultra Light でパラメータを渡す例を示します。

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
  UL_TEXT( "param2" ), UL_TEXT( "param3" ) };

...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

SQL Anywhere のリモートデータベースでは、dbmlsync -ap オプションを使用して、パラメータをカンマで区切られたリストで渡します。たとえば、次のコマンドラインは 3 つのパラメータを渡します。

```
dbmlsync -ap "param1,param2,param3"
```

この例では、authenticate_parameters スクリプトは以下のようになります。

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```

authenticate_user 接続イベント

カスタムユーザ認証を実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.authentication_status	INTEGER。これは INOUT パラメータです。	1
s.authentication_message	VARCHAR(1024)。これは INOUT パラメータです。認証メッセージを提供します。	適用外
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.password	VARCHAR(128)。認証に使用するパスワード。ユーザがパスワードを入力しない場合、この値は NULL です。	3
s.new_password	VARCHAR(128)。パスワードをリセットするために使用している場合は、新しいパスワード。ユーザがパスワードを変更しない場合、この値は NULL です。	4
s.new_remote_id	VARCHAR(128)。統合データベースでの新しいリモート ID の場合には、Mobile Link リモート ID。新しいリモート ID でない場合は、値は Null です。	
s.new_username	VARCHAR(128)。統合データベースでの新しいユーザ名の場合には、Mobile Link ユーザ名。新しいユーザ名でない場合は、値は Null です。	
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

Mobile Link 組み込みユーザ認証メカニズムを使用します。

備考

Mobile Link 同期サーバは、各同期の開始時にこのイベントを実行します。このイベントは、begin_synchronization トランザクションの前にトランザクション内で実行されます。

このイベントを使用して、組み込み Mobile Link 認証メカニズムを、カスタムメカニズムに置き換えられます。使用している DBMS の認証メカニズムを使用したり、Mobile Link 組み込みメカニズムには存在しない機能 (パスワードの有効期限やパスワードの最小長など) を実装したりできます。

authenticate_user イベントで使用するパラメータは、次のとおりです。

- **authentication_status**　authentication_status パラメータは必須です。認証の全体の成功状況を示します。次のいずれかの値に設定されます。

戻り値	authentication_status	説明
V <= 1999	1000	認証に成功しました。
2000 <= V <= 2999	2000	認証に成功しました。パスワードの有効期限がもうすぐ切れます。
3000 <= V <= 3999	3000	認証に失敗しました。パスワード失効。
4000 <= V <= 4999	4000	認証に失敗しました。
5000 <= V <= 5999	5000	リモート ID がすでに使用されているため、認証できません。後で同期を再試行してください。
6000 <= V	4000	戻り値が 5999 より大きい場合、その値は 4000 として解釈されます。

値は、クライアントで認証動作のカスタマイズに使用できるように、クライアントに送信されます。

次の項を参照してください。

○「Authentication Value 同期パラメータ」『Ultra Light データベース管理とリファレンス』
○「sp_hook_dbmlsync_upload_end」『Mobile Link クライアント管理』

- **authentication_message**　このオプションのパラメータは、認証メッセージを提供します。

この名前のパラメータは、ユーザ認証スクリプトが初回に使用する前に NULL に初期化されます。スクリプトがこの名前のパラメータを使用する場合、返されるメッセージは次のユーザ認証スクリプトに渡されます。最終メッセージは、リモートデータベースの文字セットに変換されます。

ユーザ認証スクリプトの実行時にエラーが発生しなかった場合は、ユーザ認証ステータスに関係なく、アップロードストリームの処理前に Mobile Link サーバによってこのメッセージがクライアントに送信されます。

このメッセージは、ユーザ認証が失敗した場合でもクライアントに送信されます。

- **username**　Mobile Link ユーザ名を示す省略可能なパラメータです。

「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』を参照してください。

- **remote_id**　Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。
- **password**　認証に使用するパスワードを示す省略可能なパラメータです。ユーザがパスワードを入力しない場合、この値は NULL です。
- **new_password**　新しいパスワードを示す省略可能なパラメータです。ユーザがパスワードを変更しない場合、この値は NULL です。
- **new_remote_id**　新しいリモート ID を示すオプションのパラメータです。新しいリモート ID でない場合は、Null です。
- **new_username**　新しいユーザ名を示すオプションのパラメータです。新しいユーザ名でない場合は、値は Null です。
- **script_version**　このオプションのパラメータは、Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定します。このパラメータの指定に疑問符を使用することはできません。

authenticate_user イベント用の SQL スクリプトは、ストアドプロシージャとして実装してください。

2 つの認証スクリプトを両方とも定義し、両方のスクリプトが異なる authentication_status コードを返す場合は、大きい方の値が使用されます。

authenticate_user スクリプトは、すべての認証スクリプトとともに、トランザクション内で実行されます。このトランザクションは、常にコミットを実行します。

LDAP サーバ、IMAP サーバ、POP3 サーバを使用する認証を簡素化するために、authenticate_user イベントに使用できる事前に定義されたスクリプトがあります。

「外部サーバに対する認証」『Mobile Link クライアント管理』を参照してください。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「Mobile Link ユーザ」『Mobile Link クライアント管理』
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「カスタムユーザ認証」『Mobile Link クライアント管理』
- 「外部サーバに対する認証」『Mobile Link クライアント管理』
- 「authenticate_user_hashed 接続イベント」297 ページ
- 「authenticate_parameters 接続イベント」287 ページ
- 「modify_user 接続イベント」397 ページ
- 「begin_synchronization 接続イベント」314 ページ

SQL の例

一般的な authenticate_user スクリプトは、ストアドプロシージャの呼び出しです。呼び出しの中のパラメータ順は、上記の順序と同じでなければなりません。次の例では、ml_add_connection_script を使用して、my_auth というストアドプロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
'ver1', 'authenticate_user', 'call my_auth ( {ml s.authentication_status}, {ml s.username} )'
)
```

たとえば、次の SQL Anywhere ストアドプロシージャは認証にユーザ名のみ使用し、パスワードのチェックは行いません。このプロシージャは、指定されたユーザ名が ULEmployee テーブルにある従業員 ID の 1 つであるかどうかだけを確認します。

```
CREATE PROCEDURE my_auth( inout @auth_status int, in @user_name varchar(128) )
BEGIN
  IF EXISTS
  ( SELECT * FROM ulemployee
    WHERE emp_id = @user_name )
  THEN
    MESSAGE 'OK' type info to client;
    SET @auth_status = 1000;
  ELSE
    MESSAGE 'Not OK' type info to client;
    SET @auth_status = 4000;
  END IF
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、authenticateUser という Java メソッドを authenticate_user イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_java_connection_script(
  'ver1', 'authenticate_user',
  'ExamplePackage.ExampleClass.authenticateUser'
)
```

次に示すのは、サンプルの Java メソッド authenticateUser です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する Java メソッドを呼び出します。

```
public void authenticateUser(
  ianywhere.ml.script.InOutInteger authStatus,
  String user,
  String pwd,
  String newPwd )
  throws java.sql.SQLException {
  // A real authenticate_user handler would
  // handle more authentication code states.
  _curUser = user;
  if( checkPwd( user, pwd ) ) {
    // Authentication successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( changePwd( user, pwd, newPwd ) ) {
        // Authentication OK and password change OK.
        // Use custom code.
        authStatus.setValue( 1001 );
      } else {
        // Authentication OK but password
        // change failed. Use custom code.
        java.lang.System.err.println( "user: "
         + user + " pwd change failed!" );
        authStatus.setValue( 1002 );
      }
    } else {
```

```
      authStatus.setValue( 1000 );
    }
  } else {
    // Authentication failed.
    authStatus.setValue( 4000 );
  }
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、AuthUser という .NET メソッドを authenticate_user 接続イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_connection_script(
  'ver1', 'authenticate_user',
  'TestScripts.Test.AuthUser'
)
```

次に示すのは、サンプルの .NET メソッド AuthUser です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する .NET メソッドを呼び出します。

```
namespace TestScripts {
public class Test {
   string _curUser = null;
public void AuthUser(
  ref int authStatus,
  string user,
  string pwd,
  string newPwd ) {
  // A real authenticate_user handler would
  // handle more authentication code states.
  _curUser = user;
  if( CheckPwd( user, pwd ) ) {
    // Authentication successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( ChangePwd( user, pwd, newPwd ) ) {
        // Authentication OK and password change OK.
        // Use custom code.
        authStatus = 1001;
      } else {
        // Authentication OK but password
        // change failed. Use custom code.
        System.Console.WriteLine( "user: "
         + user + " pwd change failed!" );
        authStatus = 1002;
      }
    } else {
      authStatus = 1000 ;
    }
  } else {
    // Authentication failed.
    authStatus = 4000;
  }
  }}}
```

.NET において C# で作成された authenticate_user スクリプトの詳しい例については、「.NET 同期のサンプル」546 ページを参照してください。

authenticate_user_hashed 接続イベント

カスタムユーザ認証メカニズムを実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.authentication_status	INTEGER。これは INOUT パラメータです。	1
s.authentication_message	VARCHAR(1024)。これは INOUT パラメータです。認証メッセージを提供します。	適用外
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.hashed_password	BINARY(32)。ユーザがパスワードを入力しない場合、この値は NULL です。	3
s.hashed_new_password	BINARY(32)。ユーザのパスワードを変更するためにこのイベントを使用していない場合、この値は NULL です。	4

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

Mobile Link 組み込みユーザ認証メカニズムを使用します。

備考

このイベントは authenticate_user と同じですが、パスワードの部分のみ異なります。パスワードは、ml_user.hashed_password カラムに格納されたものと同じように、ハッシュされた形式となります。パスワードをハッシュされた形式で渡すことにより、セキュリティが向上します。

一方向ハッシュが使用されます。一方向ハッシュはパスワードを使用し、それを各パスワードで (基本的に) ユニークなバイトシーケンスに変換します。一方向ハッシュでは、統合データベースに実際のパスワードを保存せずにパスワードの認証を行えます。

Mobile Link のバージョンごとにハッシュの質が徐々に向上しており、このスクリプトはユーザの認証シーケンス中に複数回呼び出すことができます。

authenticate_user と authenticate_user_hashed を両方とも定義し、両方のスクリプトが異なる authentication_status コードを返す場合は、大きい方の値が使用されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「Mobile Link ユーザ」『Mobile Link クライアント管理』
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「カスタムユーザ認証」『Mobile Link クライアント管理』
- 「authenticate_user 接続イベント」291 ページ
- 「authenticate_parameters 接続イベント」287 ページ

SQL の例

一般的な authenticate_user_hashed スクリプトは、ストアドプロシージャの呼び出しです。呼び出しの中のパラメータ順は、上記の順序と同じでなければなりません。次の例では、ml_add_connection_script を呼び出して、my_auth というストアドプロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
  'ver1', 'authenticate_user_hashed',
  'call my_auth (
    {ml s.authentication_status},
```

```
        {ml s.username},
        {ml s.hashed_password})'
)
```

次の SQL Anywhere のストアドプロシージャは、認証にユーザ名とパスワードの両方を使用します。このプロシージャは、指定されたユーザ名が ULEmployee テーブルにある従業員 ID の 1 つであるかどうかだけを確認します。プロシージャは、Employee テーブルには hashed_pwd という名前の binary(20) のカラムがあることを前提としています。

```
CREATE PROCEDURE my_auth(
  inout @authentication_status integer,
  in @user_name varchar(128),
  in @hpwd binary(32) )
BEGIN
  IF EXISTS
  ( SELECT * FROM ulemployee
    WHERE emp_id = @user_name
      and hashed_pwd = @hpwd )
  THEN
    message 'OK' type info to client;
    RETURN 1000;
  ELSE
    message 'Not OK' type info to client;
    RETURN 4000;
  END IF
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、authUserHashed という Java メソッドを authenticate_user_hashed イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1', 'authenticate_user_hashed',
   'ExamplePackage.ExampleClass.authUserHashed')
```

次に示すのは、サンプルの Java メソッド authUserHashed です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する Java メソッドを呼び出します。

```
public void authUserHashed(
  ianywhere.ml.script.InOutInteger authStatus,
  String user,
  byte pwd[],
  byte newPwd[] )
  throws java.sql.SQLException {
  // A real authenticate_user_hashed handler
  // would handle more auth code states.
  _curUser = user;
  if( checkPwdHashed( user, pwd ) ) {
    // Authorization successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( changePwdHashed( user, pwd, newPwd ) ) {
        // Authorization OK and password change OK.
        // Use custom code.
        authStatus.setValue( 1001 );
      } else {
        // Auth OK but password change failed.
        // Use custom code
        java.lang.System.err.println( "user: " + user
```

```
        + " pwd change failed!" );
      authStatus.setValue( 1002 );
    }
    } else {
      authStatus.setValue( 1000 );
    }
  } else {
    // Authorization failed.
    authStatus.setValue( 4000 );
  }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、AuthUserHashed という .NET メソッドを authenticate_user_hashed 接続イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'authenticate_user_hashed',
  'TestScripts.Test.AuthUserHashed'
)
```

次に示すのは、サンプルの .NET メソッド AuthUserHashed です。

```
namespace TestScripts {
public class Test {
   string _curUser = null;
public void AuthUserHashed(
  ref int authStatus,
  string user,
  byte[] pwdHash,
  byte[] newPwdHash ) {
  // A real authenticate_user_hashed handler
  // would handle more auth code states.
  _curUser = user;
  if( CheckPwdHashed( user, pwdHash ) ) {
    // Authorization successful.
    if( newPwdHash != null ) {
      // Password is being changed.
      if( ChangePwdHashed( user, pwdHash, newPwdHash ) ) {
      // Authorization OK and password change OK.
      // Use custom code.
      authStatus = 1001;
    } else {
      // Auth OK but password change failed.
      // Use custom code
      System.Console.WriteLine( "user: " + user
        + " pwd change failed!" );
      authStatus = 1002;
    }
      } else {
        authStatus = 1000;
      }
    } else {
      // Authorization failed.
      authStatus = 4000;
    }
    }}}
```

begin_connection 接続イベント

Mobile Link サーバが統合データベースサーバに接続するときに呼び出されます。

パラメータ

なし

デフォルトのアクション

なし。

備考

Mobile Link 同期は、同期要求を受け取ると、要求に応じて統合データベース接続を開きます。Mobile Link クライアントが Mobile Link サーバに接続すると、Mobile Link サーバはその同期のすべてのデータベースアクティビティに対して、統合データベースサーバへの接続を一時的に 1 つ割り付けます。Mobile Link サーバが接続プールからの接続を使用している場合、このイベントは呼び出されない可能性があります。

> **注意**
> このスクリプトは、通常は Java または .NET では使用されません。これは、データベース変数の代わりにこのクラスインスタンスのメンバー変数を使用し、メンバーをコンストラクタで準備するためです。

参照

- 「スクリプトの追加と削除」256 ページ
- 「end_connection 接続イベント」340 ページ
- 「-cn mlsrv16 オプション」55 ページ
- 「-w mlsrv16 オプション」89 ページ

SQL の例

次の SQL スクリプトは、SQL Anywhere 統合データベースで動作します。これで 2 つの変数が作成されます。1 つは last_download タイムスタンプ、もう 1 つは従業員 ID の変数です。

```
CALL ml_add_connection_script(
   'custdb',
   'begin_connection',
   'create variable @LastDownload timestamp;
   create variable @EmployeeID integer;')
```

begin_connection_autocommit 接続イベント

Mobile Link サーバが統合データベースサーバに接続するときに呼び出され、オートコミットがオンの場合にスクリプトの実行を一時的に許可します。

パラメータ

なし。

デフォルトのアクション

オートコミットはオフです。

備考

Mobile Link サーバが統合データベースに接続するときにエラーが発生した場合、アップロードとダウンロードをロールバックできるように、オートコミットをオフにして、データ整合性を保持します。

しかし、Adaptive Server Enterprise の統合データベースを使用している場合、オートコミットがオンでなければ、テンポラリテーブルを作成するなどの DDL 機能は実行できません。Adaptive Server Enterprise の統合データベースを使用している場合、begin_connection_autocommit イベントで DDL コマンドを実行します。イベントが終了すると、オートコミットがオフになります。

begin_connection_autocommit スクリプトは、繰り返し可能なように作成します。エラーまたはデッドロックが発生した場合、Mobile Link サーバがスクリプトをリトライする必要があるからです (スクリプトのロールバックはできません)。

このイベントは、イベントに対するスクリプトが定義されている場合のみ実行されます。

参照

● 「スクリプトの追加と削除」256 ページ

begin_download 接続イベント

Mobile Link サーバがダウンロードの準備を開始する直前に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_download	TIMESTAMP。同期テーブルの最初のダウンロード時刻。	1

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、ダウンロード情報を処理する最初の手順としてこのイベントを実行します。ダウンロード情報は 1 つのトランザクションで処理されます。このイベントは、このトランザクションで最初に実行されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_download 接続イベント」342 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

次の例では、ml_add_connection_script を呼び出して、SetDownloadParameters というストアドプロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script (
  'Lab',
  'begin_download',
  'CALL SetDownloadParameters( {ml s.last_table_download}, {ml s.username} )' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginDownloadConnection という Java メソッドを begin_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'example_ver',
  'begin_download',
  'ExamplePackage.ExampleClass.beginDownloadConnection' )
```

次に示すのは、サンプルの Java メソッド beginDownloadConnection です。このメソッドは、以前に設定された JDBC 接続を使用して削除テーブルを準備する Java メソッド (prepDeleteTables) を呼び出します。

```
public void beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException {
  prepDeleteTables ( _syncConn, ts, user );
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、BeginDownload という .NET メソッドを begin_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'begin_download',
  'TestScripts.Test.BeginDownload'
)
```

次に示すのは、サンプルの .NET メソッド BeginDownload です。このメソッドは、以前に設定されたデータベース接続を使用して削除テーブルを準備する .NET メソッド (prepDeleteTables) を呼び出します。

```
public void BeginDownload(
  DateTime timestamp,
  string user ) {
  prepDeleteTables ( _syncConn, ts, user );
  }
```

begin_download テーブルイベント

挿入、更新、削除のダウンロードを準備する直前に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1

つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、特定のテーブルのダウンロード情報を準備する最初の手順として、このイベントを実行します。ダウンロード情報は、専用トランザクションで準備されます。このイベントの実行が、このダウンロードトランザクションで最初のテーブル固有のアクションとなります。

リモートデータベースのテーブルごとに、begin_download スクリプトを 1 つ指定できます。このスクリプトは、テーブルが同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_download テーブルイベント」344 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

次の Mobile Link システムプロシージャ procedure ml_add_table_script の呼び出しは BeginTableDownload プロシージャを呼び出します。これは SQL Anywhere 16 統合データベース用の構文です。

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'begin_download',
  'CALL BeginTableDownLoad(
    {ml s.username},
    {ml s.table} )' );
```

次の SQL 文は BeginTableDownload プロシージャを作成します。テーブルのダウンロード試行を記録します。

```
CREATE PROCEDURE BeginTableDownload(
  MLUser varchar(128),
  TableName varchar(128) )
BEGIN
 INSERT INTO DownloadAttempts ( MLUser, TableName, LastDownload );
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginDownloadTable という Java メソッドを begin_download テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'begin_download',
 'ExamplePackage.ExampleClass.beginDownloadTable'
)
```

次に示すのは、サンプルの Java メソッド beginDownloadTable です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
 String _curUser = null;
public void beginDownloadTable(
 String user,
 String table ) {
 java.lang.System.out.println("Beginning to process download for: " + table);
 }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、BeginTableDownload という .NET メソッドを begin_download テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_download',
  'TestScripts.Test.BeginTableDownload'
)
```

次に示すのは、サンプルの .NET メソッド BeginTableDownload です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void BeginTableDownload(
  string user,
  string table ) {
  System.Console.WriteLine("Beginning to process download for: " + table);
  }}}
```

begin_download_deletes テーブルイベント

リモートデータベース内の指定したテーブルから削除するローのリストをフェッチする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このイベントは、リモートデータベースの指定したテーブルから削除するローのリストをフェッチする直前に実行されます。

注意
ダウンロードテーブルごとに、begin_download_deletes、download_delete_cursor、end_download_deletes イベントが順番に呼び出されます。download_delete_cursor イベントでテーブルのすべてのダウンロード削除ロジックを実装し、そのイベントを、リモートテーブルから削除するすべてのローが格納された結果セットを返す単一のストアドプロシージャとして実装することを検討してください。スクリプトの呼び出し回数を減らすと、ダウンロードパフォーマンスが向上する可能性があります。

リモートデータベースのテーブルごとに、begin_download_deletes スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_download_rows テーブルイベント」309 ページ
- 「end_download_rows テーブルイベント」348 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

begin_download_rows テーブルイベント

リモートデータベース内の指定したテーブルで挿入または更新するローのリストをフェッチする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このイベントは、リモートデータベース内の指定したテーブルで挿入または更新されるローをフェッチする直前に、実行されます。

注意
ダウンロードテーブルごとに、begin_download_deletes、download_delete_cursor、end_download_deletes イベントが順番に呼び出されます。download_delete_cursor イベントでテーブルのすべてのダウンロード削除ロジックを実装し、そのイベントを、リモートテーブルから削除するすべてのローが格納された結果セットを返す単一のストアドプロシージャとして実装することを検討してください。スクリプトの呼び出し回数を減らすと、ダウンロードパフォーマンスが向上する可能性があります。

リモートデータベースのテーブルごとに、begin_download_rows スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_download_deletes テーブルイベント」307 ページ
- 「end_download_deletes テーブルイベント」346 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

begin_publication 接続イベント

同期しているパブリケーションに関する有用な情報を提供します。このスクリプトを使って、ファイルベースのダウンロードの世代番号を管理できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.generation_number	INTEGER。これは INOUT パラメータです。使用している配備環境でファイルベースのダウンロードを使用しない場合、このパラメータは無視できます。デフォルトは 1 です。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.publication_name	VARCHAR(128)。パブリケーションの名前。	3
s.last_publication_upload	TIMESTAMP。このパブリケーションが最後に正常にアップロードされた時刻。	4
s.last_publication_download	TIMESTAMP。パブリケーションの最後のダウンロード時刻。	5
s.subscription_id	VARCHAR(128)。リモートサブスクリプション ID。	6
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

デフォルトの世代番号は 1 です。このイベントにスクリプトが定義されていない場合、リモートデータベースに送信される世代番号は必ず 1 になります。

備考

このイベントを使って、現在同期されているパブリケーションに基づいて、同期論理を設計できます。このイベントは、begin_synchronization イベントと同じトランザクションで呼び出され、begin_synchronization イベントの後で呼び出されます。このイベントは、パブリケーションが同期するたびに 1 回呼び出されます。

このイベントは、使用されるパブリケーションに基づいてダウンロードされるものに影響を与えることができます。たとえば、優先度パブリケーション (PriorityPub) と全テーブル用のパブリケーション (AllTablesPub) の双方の一部であるテーブルを考えます。begin_publication イベントのスクリプトは、Java クラスまたは SQL 変数やパッケージにパブリケーション名を保存できます。ダウンロードスクリプトは、パブリケーションが PriorityPub と同期するか AllTablesPub と同期するかにより、異なった動作ができます。

Ultra Light リモートデータベースが UL_SYNC_ALL を使用して同期されている場合、このイベントは 'unknown' というパブリケーション名で 1 回呼び出されます。

世代番号

generation_number パラメータは、特にファイルベースのダウンロード用です。世代番号の出力値は、begin_publication スクリプトから end_publication スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロードファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

ファイルベースのダウンロードでは、世代番号の変更によって、ダウンロードの前にアップロードを強制的に行います。世代番号が変わらない間は、リモートデータベースでは、アップロードの必要なく、多数のファイルベースのダウンロードを処理できます。世代番号は、ダウンロードファイルに保存されます。アップロードを持つ同期中に、パブリケーションに対するサブスクリプションごとに 1 つの世代番号が出力されます。この番号はアップロード確認でリモートデータベースへ送信され、SYSSYNC.generation_number に保存されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_publication 接続イベント」350 ページ
- 「Mobile Link ファイルベースのダウンロード」212 ページ
- 「Mobile Link の世代番号」219 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

同期されるパブリケーションごとに情報を記録する必要がある場合があります。次の例では、ml_add_connection_script を呼び出して、RecordPubSync というストアドプロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
 'version1',
 'begin_publication',
 '{CALL RecordPubSync(
    {ml s.generation_number},
    {ml s.username},
```

```
{ml s.publication_name},
{ml s.last_publication_upload},
{ml s.last_publication_download},
{ml s.subscription_id} )}' );
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginPublication という Java メソッドを begin_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_publication',
 'ExamplePackage.ExampleClass.beginPublication' )
```

次に示すのは、サンプルの Java メソッド beginPublication です。このメソッドは後で使用する各パブリケーションの名前を保存します。

```
package ExamplePackage;
public class ExampleClass
{
   java.util.ArrayList<String> _publicationNames;
   int              _numPublications = 0;

   public void beginPublication( ianywhere.ml.script.InOutInteger    generation_number,
            String              user,
            String              pub_name )
   {
   _numPublications++;
   _publicationNames.add( pub_name );
   }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、BeginPub という .NET メソッドを begin_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'begin_publication',
 'TestScripts.Test.BeginPub'
)
```

次に示すのは、サンプルの .NET メソッド BeginPub です。このメソッドは後で使用する各パブリケーションの名前を保存します。

```
using System.Collections.Generic;
namespace TestScripts
{
   class Test
   {
   List<string>    _publicationNames = new List<string>();
   int       _numPublications  = 0;

   public void BeginPub( ref int    generation_number,
            string    user,
            string    pub_name )
```

```
        {
            _numPublications++;
            _publicationNames.Add( pub_name );
        }
        }
    }
```

begin_synchronization 接続イベント

同期処理の準備のために文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.new_remote_id	VARCHAR(128)。統合データベースでの新しいリモート ID の場合には、Mobile Link リモート ID。新しいリモート ID でない場合は、値は Null です。	
s.new_username	VARCHAR(128)。統合データベースでの新しいユーザ名の場合には、Mobile Link ユーザ名。新しいユーザ名でない場合は、値は Null です。	

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、Mobile Link クライアントから同期の開始に必要なすべてを受信してからこのイベントを実行します。

begin_synchronization スクリプトは統計値の管理に便利です。これは、エラーや競合が発生しても end_synchronization スクリプトが起動されるので、アップロードトランザクションがロールバックされている間は、統計値のようにデータが維持されるためです。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_synchronization 接続イベント」353 ページ
- 「begin_synchronization テーブルイベント」316 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

username の値を後続のスクリプトで何度も参照する場合は、その値をテンポラリテーブルまたは変数に格納できます。

```
CALL ml_add_connection_script (
 'version1',
 'begin_synchronization',
 'set @EmployeeID = {ml s.username}' );
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginSynchronizationConnection という Java メソッドを begin_synchronization 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_synchronization',
 'ExamplePackage.ExampleClass.beginSynchronizationConnection'
)
```

次に示すのは、サンプルの Java メソッド beginSynchronizationConnection です。このメソッドは後で使用する同期ユーザの名前を保存します。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void beginSynchronizationConnection(
  String user ) {
  _curUser = user;
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、BeginSync という .NET メソッドを begin_synchronization 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script( 'ver1',
  'begin_synchronization',
  'TestScripts.Test.BeginSync'
)
```

次に示すのは、サンプルの .NET メソッド BeginSync です。このメソッドは後で使用する同期ユーザの名前を保存します。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void BeginSync(
  string user ) {
  _curUser = user;
  }}}
```

begin_synchronization テーブルイベント

同期の開始時に特定のテーブルに関連する文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.new_remote_id	VARCHAR(128)。統合データベースでの新しいリモート ID の場合には、Mobile Link リモート ID。新しいリモート ID でない場合は、値は Null です。	
s.new_username	VARCHAR(128)。統合データベースでの新しいユーザ名の場合には、Mobile Link ユーザ名。新しいユーザ名でない場合は、値は Null です。	
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、Mobile Link クライアントから同期の開始に必要なすべてを受信してからこのイベントを実行します。

リモートデータベースのテーブルごとに、begin_synchronization スクリプトを 1 つ指定できます。このイベントは、テーブルが同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_synchronization テーブルイベント」355 ページ
- 「begin_synchronization 接続イベント」314 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

begin_synchronization テーブルイベントを使って、特定のテーブルの同期を設定します。次の SQL スクリプトは、同期中にローを保存するためのテンポラリテーブルを作成するスクリプトを登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'begin_synchronization',
 'CREATE TABLE #sales_order (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date      date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep       integer NOT NULL,
    PRIMARY KEY (id),
)' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginSynchronizationTable という Java メソッドを begin_synchronization テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'begin_synchronization',
 'ExamplePackage.ExampleClass.beginSynchronizationTable' )
```

次に示すのは、サンプルの Java メソッド beginSynchronizationTable です。このメソッドは、このインスタンスに含まれるテーブル名のリストに現在のテーブル名を追加します。

```
package ExamplePackage;
import java.util.ArrayList;
import java.sql.Timestamp;
class ExampleClass
{
    ArrayList<String>    _tableList;
    String        _curTable;
    public void beginSynchronizationTable( String user,
                    String table )
    {
    _curTable = table;
    _tableList.add( table );
    }

    public void endTableDownload( Timestamp ts,
                String user,
                String table )
    {
    _curTable = null;
    }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、BeginTableSync という .NET メソッドを begin_synchronization テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script (
 'ver1',
 'table1',
 'begin_synchronization',
 'TestScripts.Test.BeginTableSync' )
```

次に示すのは、サンプルの .NET メソッド BeginTableSync です。このメソッドは、このインスタンスに含まれるテーブル名のリストに現在のテーブル名を追加します。

```
using System.Collections.Generic;
using System;
namespace TestScripts
{
   class Test
   {
   List<string>    _tableList = new List<string>();
   string       _curTable  = "";

   public void BeginSynchronizationTable( string    user,
                     string    table )
   {
      _curTable = table;
      _tableList.Add( table );
   }
   public void EndTableDownload( DateTime    timestamp,
                  string    user,
                  string    table )
   {
      _curTable = null;
   }
   }
}
```

begin_upload 接続イベント

Mobile Link サーバがアップロードされた挿入、更新、削除の処理を開始する直前に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、アップロードしたローを処理する最初の手順としてこのイベントを実行します。アップロードされたローは 1 つのトランザクションで処理されます。このイベントは、このトランザクションで最初に実行されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_upload 接続イベント」358 ページ
- 「begin_upload テーブルイベント」321 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

begin_upload 接続イベントを使って、ローをアップロードする前に行う必要がある手順を実行できます。次の SQL スクリプトは、sales_order テーブルの矛盾処理のために新旧のローの値を保存するテンポラリテーブルを作成します。この例は SQL Anywhere 統合データベースで動作します。

```
CALL ml_add_connection_script(
 'version1',
 'begin_upload',
 'CREATE TABLE #sales_order_conflicts (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
```

```
order_date      date NOT NULL,
fin_code_id char(2) NULL,
region      char(7) NULL,
sales_rep       integer NOT NULL,
new_value       char(1) NOT NULL,
PRIMARY KEY (id) )' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginUploadConnection という Java メソッドを begin_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_upload',
 'ExamplePackage.ExampleClass.beginUploadConnection ' )
```

次に示すのは、サンプルの Java メソッド beginUploadConnection です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void beginUploadConnection( String user ) {
  java.lang.System.out.println(
    "Starting upload for user: " + user );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、BeginUpload という .NET メソッドを begin_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'begin_upload',
  'TestScripts.Test.BeginUpload'
)
```

次の C# サンプルは、後のイベントで使用するために現在のユーザ名を保存します。

```
namespace TestScripts {
public class Test {
    string _curUser = null;
public void BeginUpload( string curUser ) {
  _curUser = curUser;
  }}}
```

begin_upload テーブルイベント

Mobile Link サーバがアップロードされた挿入、更新、削除の処理を開始する直前に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、アップロードしたローを処理する最初の手順としてこのイベントを実行します。アップロードされたローは別のトランザクションで処理されます。このイベントの実行が、このトランザクションで最初のテーブル固有のアクションとなります。

リモートデータベースのテーブルごとに、begin_upload スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_upload テーブルイベント」361 ページ
- 「begin_upload 接続イベント」319 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

リモートからローをアップロードする場合は、変更内容を中間テーブルに入れて手動で処理できます。このイベントでは、新しいローを受信するための準備として、グローバルなテンポラリテーブルから削除できます。

```
CALL ml_add_table_script(
 'version1',
 'Leads',
 'begin_upload',
 'DELETE FROM T_Leads' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginUploadTable という Java メソッドを begin_upload テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload',
  'ExamplePackage.ExampleClass.beginUploadTable'
 )
```

次に示すのは、サンプルの Java メソッド beginUploadTable です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void beginUploadTable(
  String user,
  String table ) {
  java.lang.System.out.println("Beginning to process upload for: " + table);
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、BeginTableUpload という .NET メソッドを begin_upload テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload',
  'TestScripts.Test.BeginTableUpload'
)
```

次に示すのは、サンプルの .NET メソッド BeginTableUpload です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void BeginTableUpload(
  string user,
  string table ) {
  System.Console.WriteLine("Beginning to process upload for: " + table);
  }}}
```

begin_upload_deletes テーブルイベント

リモートデータベース内の指定のテーブルから削除されたローをアップロードする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このイベントは、指定したリモートテーブルからローを削除した結果生じる変更を適用する直前に、発生します。

リモートデータベースのテーブルごとに、begin_upload_deletes スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_upload_deletes テーブルイベント」363 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

begin_upload_deletes テーブルイベントは、特定のテーブルの挿入と更新をアップロードした後で、そのテーブルの削除をアップロードする前に行う必要のある手順を実行するために使用します。次の SQL スクリプトは、アップロード中に一時的に削除を保存するためのテンポラリテーブルを作成します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'begin_upload_deletes',
 'CREATE TABLE #sales_order_deletes (
   id          integer NOT NULL default autoincrement,
   cust_id     integer NOT NULL,
   order_date  date NOT NULL,
   fin_code_id char(2) NULL,
   region      char(7) NULL,
   sales_rep   integer NOT NULL,
   PRIMARY KEY (id) )' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginUploadDeletes という Java メソッドを begin_upload_deletes テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'begin_upload_deletes',
 'ExamplePackage.ExampleClass.beginUploadDeletes' )
```

次に示すのは、サンプルの Java メソッド beginUploadDeletes です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
   String _curUser = null;
public void beginUploadDeletes(
 String user,
 String table )
 throws java.sql.SQLException {
 java.lang.System.out.println(
 "Starting upload deletes for table: " + table );
 }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、BeginUploadDeletes という .NET メソッドを begin_upload_deletes テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'begin_upload_deletes',
 'TestScripts.Test.BeginUploadDeletes'
)
```

次に示すのは、サンプルの .NET メソッド BeginUploadDeletes です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
 string _curUser = null;
public void BeginUploadDeletes(
 string user,
 string table ) {
 System.Console.WriteLine(
  "Starting upload deletes for table: " + table );
 }}}
```

begin_upload_rows テーブルイベント

リモートデータベース内の指定したテーブルから挿入と更新をアップロードする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このイベントは、2 番目のパラメータで指定したリモートテーブルに対する挿入と削除から生じる変更を適用する直前に、発生します。

リモートデータベースのテーブルごとに、begin_upload_rows スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_upload_rows テーブルイベント」366 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

begin_upload_rows テーブルイベントを使って、特定のテーブルの挿入と更新をアップロードする前に行う必要がある手順を実行します。次のスクリプトは、統合データベースで Inventory テーブルへの挿入と更新を準備するストアドプロシージャを呼び出します。

```
CALL ml_add_table_script(
 'MyCorp 1.0',
 'Inventory',
 'begin_upload_rows',
 'CALL PrepareForUpserts()' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、beginUploadRows という Java メソッドを begin_upload_rows テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload_rows',
  'ExamplePackage.ExampleClass.beginUploadRows' )
```

次に示すのは、サンプルの Java メソッド beginUploadRows です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void beginUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  java.lang.System.out.println(
   "Starting upload rows for table: " +
   table + " and user: " + user );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、BeginUploadRows という .NET メソッドを begin_upload_rows テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload_rows',
  'TestScripts.Test.BeginUploadRows'
)
```

次に示すのは、サンプルの .NET メソッド BeginUploadRows です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test
  string _curUser = null;
public void BeginUploadRows(
  string user,
  string table ) {
  System.Console.WriteLine(
   "Starting upload rows for table: " +
   table + " and user: " + user );
  }}}
```

download_cursor テーブルイベント

ダウンロードして、リモートデータベースの特定のテーブルで挿入または更新するローを選択するためのカーソルを定義するデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバはスクリプトを使用して読み込み専用のカーソルを開き、リモートデータベースにダウンロードするローのリストをフェッチします。

リモートデータベースのテーブルごとに、download_cursor スクリプトを 1 つ指定できます。

Ultra Light クライアントに対する同期のダウンロード処理のパフォーマンスを最適化するには、プライマリキー値の範囲がデバイスで指定されている現在のローの外側にある場合に、ダウンロードカーソル内のローをプライマリキー順に並べてください。たとえば、大きいリファレンステーブルをダウンロードする場合は、このような最適化による利点が得られます。

各 download_cursor スクリプトには、SELECT 文か、結果セットを返すプロシージャの呼び出しが必要です。Mobile Link サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。

スクリプトでは、対応するリモートデータベース内のテーブルのカラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモートデータベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

カラムは、対応するカラムがリモートデータベース内で定義されている順序に従って選択します。

不必要なローがダウンロードされることを防ぐために、タイムスタンプベースのダウンロードを使用することを検討してください。タイムスタンプベースのダウンロードを使用する場合は、download_cursor スクリプトの WHERE 句に次のような行を追加してください。

```
AND last_modified >= {ml s.last_table_download}
```

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

ダウンロードパフォーマンスに影響を与える大量の更新を行っているので、download_cursor スクリプトで READPAST テーブルヒントの使用を検討している場合は、代わりにダウンロードのスナップショットアイソレーションの使用を検討してください。READPAST テーブルヒントは、download_cursor スクリプトで使用すると問題を引き起こす可能性があります。タイムスタ

ンプベースのダウンロードを使用している場合は、READPAST ヒントによってローが失われ、1 つのローが一度もリモートデータベースにダウンロードされなくなる可能性があります。次に例を示します。

- 1 つのローが統合データベースに追加され、コミットされます。そのローの last_modified カラムは、昨日の日時になっています。
- 同じローが更新されますが、コミットはされません。
- last_download の値が先週の日時になっているリモートデータベースと同期されます。
- download_cursor スクリプトは READPAST を使用してローを選択しようとして、そのローをスキップします。
- ローを更新したトランザクションはロールバックされます。リモートに対する次の最終ロード時間は今日に進められます。

この時点から、このローは更新されないかぎりダウンロードされません。回避方法として、generate_next_last_download_timestamp または modify_next_last_download_timestamp スクリプトを実装して、最終ダウンロード時刻を最初に開いたトランザクションの開始時間に設定する方法が考えられます。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ローをダウンロードするスクリプト」262 ページ
- 「download_cursor スクリプト」263 ページ
- 「リモートデータベース間でローを分割する」113 ページ
- 「download_delete_cursor テーブルイベント」332 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ
- 「FROM 句」『SQL Anywhere サーバ SQL リファレンス』の「Mobile Link 同期で READPAST を使用」

SQL の例

次の例は Oracle インストール環境の場合を示していますが、文はサポートされているすべてのデータベースに対して有効です。この例は、前回データをダウンロードした後に変更されたローのうち、emp_name カラム内のユーザ名と一致するローをすべてダウンロードします。

```
CALL ml_add_table_script(
 'Lab',
 'ULOrder',
 'download_cursor',
 'SELECT order_id,
   cust_id,
   prod_id,
   emp_id,
   disc,
   quant,
   notes,
   status
```

```
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
 AND emp_name = {ml s.username}' )
```

download_delete_cursor テーブルイベント

リモートデータベースで削除するローを選択するためのカーソルを定義するデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは読み込み専用のカーソルを開き、リモートデータベースにダウンロードしてリモートデータベースから削除するローのリストをフェッチします。このスクリプトには、リ

モートデータベース内のテーブルから削除されるローのプライマリキー値を返す SELECT 文を含めてください。

リモートデータベースのテーブルごとに、download_delete_cursor スクリプトを 1 つ指定できます。

テーブル内の 1 つ以上のローのプライマリキーカラムで download_delete_cursor が NULL の場合、Mobile Link サーバは、リモートデータベースにテーブル内のローをすべて削除するように命令します。 「テーブルから全ローを削除」266 ページを参照してください。

統合データベースから削除されたローは、download_delete_cursor イベントにより定義された結果セットには表示されないため、リモートデータベースから自動的に削除されません。リモートデータベースから削除されるローを識別するには、ローを非アクティブとして識別するカラムを統合データベーステーブルに追加する方法があります。

削除対象として不必要なローがダウンロードされることを防ぐために、タイムスタンプベースのダウンロードを使用することを検討してください。download_delete_cursor スクリプトの WHERE 句に次のような行を追加してください。

```
AND last_modified >= {ml s.last_table_download}
```

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

download_delete_cursor で READPAST テーブルヒントを使用すると、問題が発生する可能性があります。詳細については、download_cursor イベントを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_cursor テーブルイベント」329 ページ
- 「ローをダウンロードするスクリプト」262 ページ
- 「リモートデータベース間でローを分割する」113 ページ
- 「download_delete_cursor スクリプト」265 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ
- 「FROM 句」『SQL Anywhere サーバ SQL リファレンス』の「Mobile Link 同期で READPAST を使用」

SQL の例

この例は Contact の例から抜粋したもので、*Samples¥MobiLink¥Contact¥build_consol.sql* にあります。この例は、このユーザが前回データをダウンロードした後に変更があった顧客 (Customer.last_modified >= {ml s.last_table_download}) と、次のいずれかに該当する顧客をリモートデータベースから削除します。

- 同期中のユーザに属していない顧客 (SalesRep.username != {ml s.username})

- 統合データベース内で非アクティブのマークが付いている顧客 (Customer.active = 0)

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'download_delete_cursor',
 'SELECT cust_id FROM Customer key join SalesRep
  WHERE Customer.last_modified >= {ml s.last_table_download} AND
   ( SalesRep.username != {ml s.username} OR Customer.active = 0 )')
```

download_statistics 接続イベント

ダウンロード操作の同期統計へのアクセスを提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。SYNCHRONIZATION USER 定義に指定した Mobile Link ユーザ名。	1
s.warnings	INTEGER。発行された警告の数。	2
s.errors	INTEGER。処理済みのエラーを含め、発生したエラーの数。	3
s.fetched_rows	INTEGER。download_cursor スクリプトによってフェッチされたローの数。	4
s.deleted_rows	INTEGER。download_delete_cursor スクリプトによってフェッチされたローの数。	5

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.filtered_rows	INTEGER。fetched_rows パラメータから実際にリモートに送信されたローの数。これには、アップロードされた値のダウンロードフィルタが反映されます。	6
s.bytes	INTEGER。ダウンロードとしてリモートデータベースに送信されたバイト数。	7
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

download_statistics イベントを使用すると、任意のユーザについてダウンロード統計を収集できます。ダウンロードトランザクション終了時のコミット直前に、download_statistics 接続スクリプトが呼び出されます。

注意
コマンドラインによっては、一部の警告のログが取られないことがあります。このスクリプトに渡される警告数は、警告が無効になっていない場合にログに取られる警告数であり、ログに取られる警告数より多くなることがあります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、同期の統計を download_audit というテーブルに挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'download_statistics',
 'INSERT INTO download_audit(
   user_name,
   warnings,
   errors,
   fetched_rows,
   deleted_rows,
   filtered_rows,
   bytes )
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.fetched_rows},
   {ml s.deleted_rows},
   {ml s.filtered_rows},
   {ml s.bytes})')
```

監査テーブルに重要な統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、downloadStatisticsConnection という Java メソッドを download_statistics イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'download_statistics',
 'ExamplePackage.ExampleClass.downloadStatisticsConnection' )
```

次に示すのは、サンプルの Java メソッド downloadStatisticsConnection です。このメソッドは、フェッチしたローの数を Mobile Link メッセージログに出力します (フェッチしたローの数を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void downloadStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int filteredRows,
  int bytes ) {
  java.lang.System.out.println(
   "download connection stats fetchedRows: "
   + fetchedRows );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、DownloadStats という .NET メソッドを download_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'download_statistics',
 'TestScripts.Test.DownloadStats'
)
```

次に示すのは、サンプルの .NET メソッド DownloadStats です。このメソッドは、フェッチしたローの数を Mobile Link メッセージログに出力します (フェッチしたローの数を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
 string _curUser = null;
public void DownloadStats(
 string user,
 int warnings,
 int errors,
 int deletedRows,
 int fetchedRows,
 int downloadRows,
 int filteredRows,
 int bytes ) {
 System.Console.WriteLine(
  "download connection stats fetchedRows: "
  + fetchedRows );
 }}}
```

download_statistics テーブルイベント

ダウンロード操作のテーブル別同期統計へのアクセスを提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.username	VARCHAR(128)。SYNCHRONIZATION USER 定義に指定した Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.warnings	INTEGER。発行された警告の数。	3
s.errors	INTEGER。処理済みのエラーを含め、発生したエラーの数。	4
s.fetched_rows	INTEGER。download_cursor スクリプトによってフェッチされたローの数。	5
s.deleted_rows	INTEGER。download_delete_cursor スクリプトによってフェッチされたローの数。	6
s.filtered_rows	INTEGER。fetched_rows からフィルタされたローの合計数。これには、アップロードされた値のダウンロードフィルタが反映されます。	7
s.bytes	INTEGER。ダウンロードとしてリモートデータベースに送信されたバイト数。	8
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし

備考

download_statistics イベントを使用すると、任意のユーザとテーブルについて、そのテーブルに適用されるダウンロードの統計を収集できます。ダウンロードトランザクション終了時のコミット直前に、download_statistics テーブルスクリプトが呼び出されます。

注意
コマンドラインによっては、一部の警告のログが取られないことがあります。このスクリプトに渡される警告数は、警告が無効になっていない場合にログに取られる警告数であり、ログに取られる警告数より多くなることがあります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_statistics 接続イベント」334 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、同期の統計を download_audit というテーブルに挿入します。監査テーブルに重要な統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'download_statistics',
 'INSERT INTO download_audit (
   user_name,
   table, warnings,
   errors,
   fetched_rows,
   deleted_rows,
   filtered_rows,
   bytes)
  VALUES (
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.fetched_rows},
   {ml s.deleted_rows},
   {ml s.filtered_rows},
   {ml s.bytes})')
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、downloadStatisticsTable という Java メソッドを download_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド downloadStatisticsTable です。このメソッドは、このテーブルの統計を Mobile Link メッセージログに出力します (テーブルの統計を Mobile Link

メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void downloadStatisticsTable(
  String user,
  String table,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int filteredRows,
  int bytes ) {
  java.lang.System.out.println( "download table stats "
   + "table: " + table + "bytes: " + bytes );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、DownloadTableStats という .NET メソッドを download_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'download_statistics',
 'TestScripts.Test.DownloadTableStats'
)
```

次に示すのは、サンプルの .NET メソッド DownloadTableStats です。このメソッドは、このテーブルの統計を Mobile Link メッセージログに出力します (テーブルの統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void DownloadTableStats(
  string user,
  string table,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int filteredRows,
  int bytes ) {
  System.Console.WriteLine( "download table stats "
   + "table: " + table + "bytes: " + bytes );
  }}}
```

end_connection 接続イベント

停止準備中、または接続プールから接続が削除されるとき、Mobile Link サーバが統合データベースサーバとの接続を閉じる直前に、任意の文を処理します。

パラメータ

なし。

デフォルトのアクション

なし。

備考

Mobile Link サーバと統合データベースサーバ間の接続を閉じる直前に、end_connection スクリプトを使用して、選択したアクションを実行できます。

このスクリプトは通常、begin_connection スクリプトによって起動されたすべてのアクションを完了し、取得されていたリソースをすべて解放するために使用されます。

参照

- 「begin_connection 接続イベント」301 ページ
- 「スクリプトの追加と削除」256 ページ

SQL の例

次の SQL スクリプトは、begin_connection スクリプトが作成したテンポラリテーブルを削除します。これは SQL Anywhere 統合データベース用の構文です。厳密に言うと、このテーブルは明示的に削除する必要はありません。接続が切断されるときに SQL Anywhere が自動的に削除します。テンポラリテーブルを明示的に削除する必要があるかどうかは、統合データベースのタイプによります。

```
CALL ml_add_connection_script(
 'version 1.0',
 'end_connection',
 'DROP TABLE #sync_info' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endConnection という Java メソッドを end_connection イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_connection',
  'ExamplePackage.ExampleClass.endConnection' )
```

次に示すのは、サンプルの Java メソッド endConnection です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void endConnection() {
  java.lang.System.out.println( "Ending connection." );
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、EndConnection という .NET メソッドを end_connection 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'end_connection',
 'TestScripts.Test.EndConnection'
)
```

次に示すのは、サンプルの .NET メソッド EndConnection です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
 string _curUser = null;
public void EndConnection() {
 System.Console.WriteLine( "Ending connection." );
 }}}
```

end_download 接続イベント

Mobile Link サーバがダウンロードデータの準備を完了した直後に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_download	TIMESTAMP。同期テーブルの最初のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

すべてのダウンロードローが統合データベースからフェッチされてから、Mobile Link サーバはこのスクリプトを実行します。このスクリプトの実行は、ダウンロードの最後の非統計アクションです。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_download 接続イベント」302 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

次の例は、end_download 接続スクリプトの、考えられる用途の 1 つを示します。このスクリプトは、ダウンロードの生成に使用されるテンポラリテーブルからローを削除します。

```
CALL ml_add_connection_script(
 'ver1',
 'end_download',
 'DELETE FROM TempDownloadTable where user = {ml s.username}')
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endDownloadConnection という Java メソッドを end_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadConnection' )
```

次に示すのは、サンプルの Java メソッド endDownloadConnection です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
import java.sql.*;
public class ExampleClass {
    String _curUser = null;
public void endDownloadConnection(
  Timestamp ts,
  String user )
{
  java.lang.System.out.println( "Ending download for user: " + user );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、EndDownload という .NET メソッドを end_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'end_download',
 'TestScripts.Test.EndDownload' )
```

次に示すのは、サンプルの .NET メソッド EndDownload です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void EndDownload(
  DateTime timestamp,
  string user ) {
  System.Console.WriteLine( "Ending download for user: " + user );
  }
```

end_download テーブルイベント

Mobile Link サーバがダウンロードデータの準備を完了した直後に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

すべてのダウンロードローが統合データベースからフェッチされてから、Mobile Link サーバはこのスクリプトを実行します。このスクリプトの実行が、このダウンロードトランザクションで最後のテーブル固有の非統計アクションとなります。

リモートデータベースのテーブルごとに、end_download スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_download テーブルイベント」304 ページ
- 「end_download 接続イベント」342 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

end_download テーブルイベントを使って、特定のテーブルをダウンロードした後で行う必要がある手順を実行します。次の SQL Anywhere の SQL スクリプトは、sales_summary テーブルのダウンロードローを保持するために prepare_for_download スクリプトが作成したテンポラリテーブルを削除します。

```
CALL ml_add_table_script(
 'MyCorp 1.0',
 'sales_summary',
 'end_download',
 'DROP TABLE #sales_summary_download' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endDownloadTable という Java メソッドを end_download テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script (
  'ver1',
  'table1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadTable' )
```

次に示すのは、サンプルの Java メソッド endDownloadTable です。このメソッドは現在のテーブルメンバー変数をリセットします。

```
public void endDownloadTable(
  Timestamp ts,
  String user,
  String table ) {
  _curTable = null;
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、EndTableDownload という .NET メソッドを end_download テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_download',
  'TestScripts.Test.EndTableDownload'
)
```

次に示すのは、サンプルの .NET メソッド EndTableDownload です。このメソッドは現在のテーブルメンバー変数をリセットします。

```
public void EndTableDownload
  DateTime timestamp,
  string user,
  string table ) {
  _curTable = null;
  }}}
```

end_download_deletes テーブルイベント

リモートデータベース内の指定されたテーブルから削除するローのリストを準備した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトは、リモートデータベース内の指定されたテーブルから削除されるローのリストを準備した直後に実行されます。

注意
ダウンロードテーブルごとに、begin_download_deletes、download_delete_cursor、end_download_deletes イベントが順番に呼び出されます。download_delete_cursor イベントでテーブルのすべてのダウンロード削除ロジックを実装し、そのイベントを、リモートテーブルから削除するすべてのローが格納された結果セットを返す単一のストアドプロシージャとして実装することを検討してください。スクリプトの呼び出し回数を減らすと、ダウンロードパフォーマンスが向上する可能性があります。

リモートデータベースのテーブルごとに、end_download_deletes スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_download_deletes テーブルイベント」307 ページ
- 「end_download 接続イベント」342 ページ
- 「begin_download_rows テーブルイベント」309 ページ
- 「end_download_rows テーブルイベント」348 ページ
- 「download_delete_cursor テーブルイベント」332 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

end_download_rows テーブルイベント

リモートデータベース内の指定されたテーブルで挿入または更新するローのリストを準備した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトは、リモートデータベース内の指定されたテーブルで挿入または更新されるローのストリームを準備した直後に実行されます。

> **注意**
> ダウンロードテーブルごとに、begin_download_deletes、download_delete_cursor、end_download_deletes イベントが順番に呼び出されます。download_delete_cursor イベントでテーブルのすべてのダウンロード削除ロジックを実装し、そのイベントを、リモートテーブルから削除するすべてのローが格納された結果セットを返す単一のストアドプロシージャとして実装することを検討してください。スクリプトの呼び出し回数を減らすと、ダウンロードパフォーマンスが向上する可能性があります。

リモートデータベースのテーブルごとに、end_download_rows スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_download_rows テーブルイベント」309 ページ
- 「end_download 接続イベント」342 ページ
- 「end_download_deletes テーブルイベント」346 ページ
- 「begin_download_deletes テーブルイベント」307 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

end_publication 接続イベント

同期しているパブリケーションに関する有用な情報を提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.generation_number	INTEGER。使用している配備環境でファイルベースのダウンロードを使用しない場合、このパラメータは無視できます。デフォルト値は 1 です。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.publication_name	VARCHAR(128)。パブリケーションの名前。	3
s.last_publication_upload	TIMESTAMP。このパブリケーションが最後に正常にアップロードされた時刻。	4

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_publication_download	TIMESTAMP。パブリケーションの最後のダウンロード時刻。	5
s.subscription_id	VARCHAR(128)。リモートサブスクリプション ID。	6
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このイベントを使って、現在同期されているパブリケーションに基づいて、同期論理を設計できます。このイベントは、end_synchronization イベントと同じトランザクションで呼び出され、end_synchronization イベントの前に呼び出されます。このイベントは、パブリケーションが同期するたびに 1 回呼び出されます。

現在の同期がアップロードを正常に適用すると、last_upload パラメータにはこの最終アップロードが適用された時刻が含まれます。last_publication_download は、最終ダウンロード時刻としてダウンロードスクリプトに渡された値と同じです。

Ultra Light リモートデータベースが UL_SYNC_ALL を使用して同期されている場合、このイベントは 'unknown' という名前で 1 回呼び出されます。

世代番号

generation_number パラメータは、特にファイルベースのダウンロード用です。ファイルベースのダウンロードでは、ファイルがリモートで適用されるときに、世代番号の変更によって、ダウンロードの前にアップロードを強制的に行います。世代番号は、ダウンロードファイルに保存されます。

世代番号の出力値は、begin_publication スクリプトから end_publication スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロードファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_publication 接続イベント」310 ページ
- 「Mobile Link ファイルベースのダウンロード」212 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

同期されるパブリケーションごとに情報を記録する必要がある場合があります。次の例では、ml_add_connection_script を呼び出して、RecordPubEndSync というストアドプロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
  'version1',
  'end_publication',
  'CALL RecordPubEndSync(
   {ml s.generation_number},
   {ml s.username},
   {ml s.publication_name},
   {ml s.last_publication_upload},
   {ml s.last_publication_download} )' );
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endPublication という Java メソッドを end_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_publication',
  'ExamplePackage.ExampleClass.endPublication' )
```

次に示すのは、サンプルの Java メソッド endPublication です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
import java.sql.*;
public class ExampleClass {
  String _curUser = null;
public void endPublication(
  int generation_number,
  String user,
  String pub_name,
  Timestamp last_publication_upload,
  Timestamp last_publication_download ) {
  java.lang.System.out.println(
   "Finished synchronizing publication " + pub_name );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、EndPub という .NET メソッドを end_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_publication',
  'TestScripts.Test.EndPub'
)
```

次に示すのは、サンプルの .NET メソッド EndPub です。このメソッドは、Mobile Link メッセージログにメッセージを出力します (メッセージを Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public void EndPub(
  int generation_number,
  string user,
  string pub_name,
  DateTime last_publication_upload,
  DateTime last_publication_download ) {
  System.Console.Write(
   "Finished synchronizing publication " + pub_name );
  }
```

end_synchronization 接続イベント

同期処理の最後に文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.synchronization_ok	INTEGER。この値は、同期が成功すると 1 に、失敗すると 0 になります。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

同期の完了後、Mobile Link サーバはこのスクリプトを実行します。

このスクリプトは、ダウンロードトランザクションの後に、別のトランザクションで実行されます。ダウンロード確認が要求されていない場合、リモートデータベースは end_synchronization スクリプトが開始または完了する前に同期を終了し、接続を切断することがあります。

end_synchronization スクリプトは統計値の管理に便利です。これは、begin_synchronization スクリプトを呼び出した場合、前のトランザクションでエラーが発生しても end_synchronization スクリプトが呼び出されるので、アップロードトランザクションがロールバックされている間は、統計値が維持されるためです。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_synchronization 接続イベント」314 ページ
- 「begin_synchronization テーブルイベント」316 ページ
- 「end_synchronization テーブルイベント」355 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の SQL スクリプトは、同期試行の終了時刻と同期の成功または失敗を記録するシステムプロシージャを呼び出します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_connection_script(
 'ver1',
 'end_synchronization',
 'CALL RecordEndOfSyncAttempt(
   {ml s.username},
   {ml s.synchronization_ok} )' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endSynchronizationConnection という Java メソッドを end_synchronization イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'end_synchronization',
 'ExamplePackage.ExampleClass.endSynchronizationConnection'
)
```

次に示すのは、サンプルの Java メソッド endSynchronizationConnection です。このメソッドは JDBC 接続を使用して更新を実行します。これは SQL Anywhere 統合データベース用の構文です。

```
public void endSynchronizationConnection(
 String user )
 throws java.sql.SQLException {
 execUpdate( _syncConn,
  "UPDATE sync_count set count = count + 1 where user_id = '"
  + user + "' " );
 }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、EndSync という .NET メソッドを end_synchronization 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'end_synchronization',
 'TestScripts.Test.EndSync'
)
```

次に示すのは、サンプルの .NET メソッド EndSync です。このメソッドは、テーブル sync_count を更新します。これは SQL Anywhere 統合データベース用の構文です。

```
namespace TestScripts {
public class Test {
 string _curUser = null;
public void EndSync(
 string user ) {
 return(
  "UPDATE sync_count set count = count + 1 where user_id = '"
  + user + "' ");
 }}}
```

end_synchronization テーブルイベント

同期処理の最後に文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.synchronization_ok	INTEGER。この値は、同期が成功すると 1 に、失敗すると 0 になります。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

アプリケーションが同期を終了し、Mobile Link サーバから切断しようとしているとき、Mobile Link サーバは、同じ名前の接続レベルのスクリプトの前に、このスクリプトを実行します。

リモートデータベースのテーブルごとに、end_synchronization スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_synchronization テーブルイベント」316 ページ
- 「end_synchronization 接続イベント」353 ページ
- 「end_synchronization テーブルイベント」355 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の SQL Anywhere の SQL スクリプトは、begin_synchronization スクリプトが作成したテンポラリテーブルを削除します。

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'end_synchronization',
 'DROP TABLE #sales_order' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endSynchronizationTable という Java メソッドを end_synchronization テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_synchronization',
  'ExamplePackage.ExampleClass.endSynchronizationTable' )
```

次に示すのは、サンプルの Java メソッド endSynchronizationTable です。

```
package ExamplePackage;
import ianywhere.ml.script.*;
import java.sql.*;
public class ExampleClass {
   private DBConnectionContext _cc = null;
   public ExampleClass( DBConnectionContext cc ) {
            _cc = cc;
   }
   public void endSynchronizationTable() throws SQLException {
            try( Connection conn = _cc.getConnection() ) {
               try( PreparedStatement stmt = conn.prepareStatement( "DROP TABLE #sales_order" ) ) {
                        stmt.executeUpdate();
               }
            }
   }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、EndTableSync という .NET メソッドを end_synchronization テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
```

```
 'table1',
 'end_synchronization',
 'TestScripts.Test.EndTableSync'
)
```

次に示すのは、サンプルの .NET メソッド EndSynchronizationTable です。

```
using iAnywhere.MobiLink.Script;
namespace TestScripts {
   public class ExampleClass {
            DBConnectionContext _cc = null;
            ExampleClass( DBConnectionContext cc ) {
               _cc = cc;
            }
            public void EndSynchronizationTable() {
               DBConnection conn = _cc.GetConnection();
               try {
                        DBCommand cmd = conn.CreateCommand();
                        try {
                           cmd.CommandText = "DROP TABLE #sales_order";
                           cmd.Prepare();
                           cmd.ExecuteNonQuery();
                        } finally {
                           cmd.Close();
                        }
               } finally {
                        conn.Close();
               }
            }
   }
}
```

end_upload 接続イベント

Mobile Link サーバがアップロードされた挿入、更新、削除の処理を完了した直後に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、アップロードした情報を処理する最後の手順としてこのスクリプトを実行します。アップロード情報は 1 つのトランザクションで処理されます。このスクリプトは、このトランザクションで統計スクリプトの前に実行される最後のアクションです。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_upload 接続イベント」319 ページ
- 「end_upload テーブルイベント」361 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の SQL Anywhere の SQL スクリプトは EndUpload ストアドプロシージャを呼び出します。

```
CALL ml_add_connection_script(
 'ver1',
 'end_upload',
 'CALL EndUpload({ml s.username});' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endUploadConnection という Java メソッドを end_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_upload',
  'ExamplePackage.ExampleClass.endUploadConnection' )
```

次に示すのは、サンプルの Java メソッド endUploadConnection です。このメソッドは、データベースの操作を実行するメソッドを呼び出します。

```
public void endUploadConnection( String user ) {
  // Clean up new and old tables.
  Iterator    two_iter = _tables_with_ops.iterator();
  while( two_iter.hasNext() ) {
    TableInfo cur_table = (TableInfo)two_iter.next();
    dumpTableOps( _sync_conn, cur_table );
  }
  _tables_with_ops.clear();
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、EndUpload という .NET メソッドを end_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_upload',
  'TestScripts.Test.EndUpload'
)
```

次に示すのは、サンプルの .NET メソッド EndUpload です。

```
using iAnywhere.MobiLink.Script;
namespace TestScripts {
   public class ExampleClass {
            DBConnectionContext _cc = null;
            ExampleClass( DBConnectionContext cc ) {
               _cc = cc;
            }
            public void EndUpload( string userName ) {
               DBConnection conn = _cc.GetConnection();
               try {
                        DBCommand cmd = conn.CreateCommand();
                        try {
                           cmd.CommandText = "CALL EndUpload( ? )";
                           cmd.Prepare();
                           DBParameter parm = new DBParameter();
                           parm.DbType = SQLType.SQL_CHAR;
                           parm.Value = userName;
                           cmd.Parameters.Add( parm );
                           cmd.ExecuteNonQuery();
                        } finally {
                           cmd.Close();
                        }
               } finally {
                        conn.Close();
               }
            }
   }
}
```

end_upload テーブルイベント

Mobile Link サーバがアップロードされた挿入、更新、削除の処理を終了した直後に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

パラメータ	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、アップロードした情報を処理する最後の手順としてこのスクリプトを実行します。アップロード情報は別のトランザクションで処理されます。このスクリプトの実行が、このトランザクションで最後のテーブル固有のアクションになります。

リモートデータベースのテーブルごとに、end_upload スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_upload テーブルイベント」321 ページ
- 「end_upload 接続イベント」358 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の Mobile Link システムプロシージャコールは end_upload イベントを、ULCustomerIDPool_maintain というストアドプロシージャに割り当てます。

```
CALL ml_add_table_script(
  'custdb',
  'ULCustomerIDPool',
  'end_upload',
  '{ CALL ULCustomerIDPool_maintain( {ml s.username} ) }' )
```

次の SQL 文は ULCustomerIDPool_maintain ストアドプロシージャを作成します。このプロシージャは、新しいプライマリキーをプライマリキープールに挿入して、アップロードしたばかりのローで使用されているキーを置き換えます。プライマリキープールは、後で同じ同期でリモートデータベースにダウンロードされます。

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
   DECLARE pool_count INTEGER;

   -- Determine how many ids to add to the pool
   SELECT COUNT(*) INTO pool_count
      FROM ULCustomerIDPool WHERE pool_emp_id = syncuser_id;

   -- Top up the pool with new ids
   WHILE pool_count < 20 LOOP
      INSERT INTO ULCustomerIDPool ( pool_emp_id ) VALUES ( syncuser_id );
      SET pool_count = pool_count + 1;
   END LOOP;
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endUploadTable という Java メソッドを end_upload テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_upload',
  'ExamplePackage.ExampleClass.endUploadTable' )
```

次に示すのは、サンプルの Java メソッド endUploadTable です。このメソッドは、渡されたテーブル名と関連する名前のテーブルに対する削除を生成します。これは SQL Anywhere 統合データベース用の構文です。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
```

```
public void endUploadTable(
  String user,
  String table ) {
  return( "DELETE from '" + table + "_temp'" );
}}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、EndUpload という .NET メソッドを end_upload テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_upload',
  'TestScripts.Test.EndUpload'
)
```

次の .NET サンプルは、テンポラリテーブルに挿入されたローを、スクリプトに渡されたテーブルに移動します。

```
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
   public class Test
   {
   DBConnection    _curConn = null;

   public Test( DBConnectionContext cc )
   {
      _curConn = cc.GetConnection();
   }
   public void EndUpload( string user, string table )
   {
      DBCommand stmt = _curConn.CreateCommand();
      // Move the uploaded rows to the destination table.
      stmt.CommandText = "INSERT INTO "
         + table
         + " SELECT * FROM dnet_ul_temp";
      stmt.ExecuteNonQuery();
      stmt.Close();
   }
   }
}
```

end_upload_deletes テーブルイベント

リモートデータベース内の指定されたテーブルからアップロードされた削除を適用した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトは、指定したリモートテーブルでローを削除した結果生じる変更を適用した直後に実行されます。

リモートデータベースのテーブルごとに、end_upload_deletes スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「begin_upload_deletes テーブルイベント」324 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

このイベントを使用すると、中間テーブル上でアップロード中に削除されたローを処理できます。ベーステーブルのローを中間テーブルのローと比較して、削除されたローの処理を決定できます。

次の Mobile Link システムプロシージャコールは、EndUploadDeletesLeads ストアドプロシージャを end_upload_deletes イベントに割り当てます。

```
CALL ml_add_table_script(
 'version1',
 'Leads',
 'end_upload_deletes',
 'call EndUploadDeletesLeads()');
```

次の SQL 文は EndUploadDeletes ストアドプロシージャを作成します。

```
CREATE PROCEDURE EndUploadDeletesLeads ( )
Begin
 FOR names AS curs CURSOR FOR
 SELECT LeadID
   FROM Leads
   WHERE LeadID NOT IN (SELECT LeadID FROM T_Leads)
 DO
  CALL decide_what_to_do( LeadID )
 END FOR;
end
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endUploadDeletes という Java メソッドを end_upload_deletes テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_upload_deletes',
  'ExamplePackage.ExampleClass.endUploadDeletes' )
```

次に示すのは、サンプルの Java メソッド endUploadDeletes です。このメソッドは、データベースを操作する Java メソッドを呼び出します。

```
public void endUploadDeletes(
 String user,
 String table )
 throws java.sql.SQLException {
 processUploadedDeletes( _syncConn, table );
 }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、EndUploadDeletes という .NET メソッドを end_upload_deletes テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
```

```
    'end_upload_deletes',
    'TestScripts.Test.EndUploadDeletes'
  )
```

次に示すのは、サンプルの .NET メソッド EndUploadDeletes です。このメソッドは、データベースを操作する .NET メソッドを呼び出します。

```
namespace TestScripts {
public class Test {
   string _curUser = null;
public void EndUploadDeletes(
  string user,
  string table ) {
  processUploadedDeletes( _syncConn, table );
  }}}
```

end_upload_rows テーブルイベント

リモートデータベース内の指定されたテーブルからアップロードされた挿入と更新を適用した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトは、指定したリモートテーブルに対する修正の結果生じる変更を適用した直後に実行されます。

リモートデータベースのテーブルごとに、end_upload_rows スクリプトを 1 つ指定できます。

参照

- 「スクリプトの追加と削除」256 ページ
- 「begin_upload_rows テーブルイベント」326 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の Mobile Link システムプロシージャの呼び出しは、スクリプトバージョン ver1 を同期するときに、EndUploadRows という SQL メソッドを end_upload_rows テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_table_script(
 'version1',
 'table1',
 'end_upload_rows',
 'CALL EndUploadRows(
    { ml s.username },
    { ml s.table } )' )
```

次に示すのは、サンプルの SQL メソッド EndUploadRows です。このメソッドは、データベースを操作する SQL メソッドを呼び出します。

```
CREATE PROCEDURE EndUploadRows (
 IN username VARCHAR(128)
 IN tablename VARCHAR{128} )
BEGIN
   CALL decide_what_to_do(tablename);
END;
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、endUploadRows という Java メソッドを end_upload_rows テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_upload_rows',
  'ExamplePackage.ExampleClass.endUploadRows' )
```

次に示すのは、サンプルの Java メソッド endUploadRows です。このメソッドは、データベースを操作する Java メソッドを呼び出します。

```
public void endUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  processUploadedRows( _syncConn, table );
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、EndUploadRows という .NET メソッドを end_upload_rows テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_upload_rows',
  'TestScripts.Test.EndUploadRows'
)
```

次に示すのは、サンプルの .NET メソッド EndUploadRows です。このメソッドは、データベースを操作する .NET メソッドを呼び出します。

```
public void EndUploadRows(
  string user,
  string table ) {
  processUploadedRows( _syncConn, table );
  }}}
```

generate_next_last_download_timestamp イベント

このスクリプトは、ユーザ定義のアルゴリズムを呼び出して、next_last_download_timestamp を生成する場合に使用します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1

つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.next_last_download	TIMESTAMP。これは INOUT パラメータです。Mobile Link サーバは、このパラメータを last_download_timestamp で初期化します。last_download_timestamp は、現在の同期でダウンロードストリームを生成するために使用されたタイムスタンプです。	1
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

備考

このスクリプトは、prepare_for_download スクリプトの呼び出しの直前に prepare_for_download トランザクションで呼び出されます。

このイベントは、特にスナップショットアイソレーションレベルをサポートする統合データベース (SQL Anywhere、Oracle、Microsoft SQL Server、IBM DB2 LUW 9.7 など) では注意して使用してください。Mobile Link サーバは、Oracle を使用したダウンロードでは常にスナップショットアイソレーションレベルを使用します。デフォルトでは、SQL Anywhere と Microsoft SQL Server を使用したダウンロードでもスナップショットアイソレーションレベルを使用します (スナップショットアイソレーションレベルがデータベースで有効になっている場合)。

信頼性の高いタイムスタンプベースの同期を行うには、next_last_download の出力が次のうち早い方の時刻である必要があります。

1. 現在のタイムスタンプ
2. ダウンロードの作成に使用されたテーブルまたはビューを更新 (挿入、更新、削除など) する、最も早く開いたトランザクションの開始タイムスタンプ

このスクリプトは、--{ml_ignore} 句を使用して、無視されるスクリプトとして指定することもできます。このスクリプトが無視されるスクリプトとして定義されると、Mobile Link サーバではこのスクリプトを呼び出さず、次の最終ダウンロードタイムスタンプを生成するために Mobile Link の内部ロジックを使用しません。代わりに、Mobile Link サーバは、現在の同期でクライアントから送信された最終ダウンロードタイムスタンプをクライアントに送り返します。この方法は、すべての同期テーブルについて、常に統合データベースからすべてのローをダウンロードする同期に使用できます。ただし、時間ベースの同期では、適切なビジネスロジックを使用してこのスクリプトを実際のスクリプトとして定義し、次の最終ダウンロードタイムスタンプを生成する必要があります。また、このイベントにはスクリプトを定義しないでください。Mobile Link サーバは内部ロジックを使用して次の最終ダウンロードタイムスタンプを生成します。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ
- 「ダウンロードタイムスタンプの生成および使用方法」110 ページ
- 「modify_next_last_download_timestamp 接続イベント」394 ページ
- 「modify_last_download_timestamp 接続イベント」391 ページ

SQL の例

generate_next_last_download_timestamp スクリプトを Mobile Link サーバで使用して、UTC ベースのダウンロードを生成できます。Oracle 用の UTC ベースのダウンロードを設定する手順を次に示します。

1. 次のように定義された **my_table** という同期テーブルがあるとします。

```
CREATE TABLE my_table (  pk            INT PRIMARY KEY NOT NULL,
                         c1            VARCHAR(100) ,
                         last_modified   TIMESTAMP DEFAULT SYS_EXTRACT_UTC( SYSTIMESTAMP )
)
```

2. **GenerateNextDownloadTimestamp** というストアドプロシージャを作成して、Oracle データベースで最も早く開いたトランザクションの開始時刻を UTC で取得します。

```
CREATE PROCEDURE GenerateNextDownloadTimestamp ( p_ts IN OUT TIMESTAMP ) AS
BEGIN
    SELECT SYS_EXTRACT_UTC( NVL( MIN( TO_TIMESTAMP( START_TIME, 'mm/dd/rr
hh24:mi:ss' ) ),
                                 SYSTIMESTAMP ) )
        INTO p_ts FROM GV$TRANSACTION;
END;
```

3. ml_add_connect_script を呼び出してスクリプトをインストールします。

```
call ml_add_connection_script(
  'my_script_version',
```

```
'generate_next_last_download_timestamp',
'{ call GenerateNextDownloadTimestamp( {ml s.next_last_download} ) }' )
```

> **注意**
> Mobile Link サーバのログオン ID には、GV_$TRANSACTION に対する SELECT 権限が必要です。

handle_DownloadData 接続イベント

ダイレクトローハンドリングにおいて、ダウンロードするローセットの作成に使用される非 SQL データスクリプトです。

パラメータ

なし。

デフォルトのアクション

なし。

備考

handle_DownloadData イベントを使用すると、ダイレクトローハンドリングを使用して Mobile Link クライアントにダウンロードする操作を決定できます。

ダイレクトローハンドリングは、Mobile Link でサポートされている統合データベース以外のデータソースと同期するために使用されます。 「ダイレクトローハンドリング」623 ページを参照してください。

ダイレクトダウンロードを作成するには、Java または .NET 用 Mobile Link サーバ API の DownloadData クラスと DownloadTableData クラスを使用できます。

Java の場合、DBConnectionContext の getDownloadData メソッドは、現在の同期に対する DownloadData インスタンスを返します。DownloadData はすべてのダウンロード操作をカプセル化して、リモートクライアントに送信します。DownloadData の getDownloadTables メソッドと getDownloadTableByName メソッドを使用して DownloadTableData インスタンスを取得できます。DownloadTableData は、特定のテーブルに対するダウンロード操作をカプセル化します。getUpsertPreparedStatement メソッドを使用して、挿入と更新操作の準備文を取得できます。DownloadTableData の getDeletePreparedStatement メソッドを使用して、削除操作の準備文を取得できます。

.NET の場合、DBConnectionContext の GetDownloadData メソッドは、現在の同期に対する DownloadData インスタンスを返します。DownloadData はすべてのダウンロード操作をカプセル化して、リモートクライアントに送信します。DownloadData の GetDownloadTables メソッドと GetDownloadTableByName メソッドを使用して DownloadTableData インスタンスを取得できます。DownloadTableData は、特定のテーブルに対するダウンロード操作をカプセル化します。GetUpsertCommand メソッドを使用して、挿入と更新操作のコマンドを取得できます。DownloadTableData の getDeleteCommand メソッドを使用して、削除操作のコマンドを取得できます。

Java の場合については、DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページを参照してください。

.NET の場合については、DBConnectionContext インタフェース [Mobile Link サーバ .NET]561 ページを参照してください。

handle_DownloadData または他の同期イベントでダウンロードを作成できます。Mobile Link にはこの柔軟性があるので、データがアップロードされたときまたは特定のイベントが発生したときのダウンロード操作を設定できます。handle_DownloadData 以外のイベントでダイレクトダウンロードを作成する場合は、含まれているメソッドが何も処理をしない handle_DownloadData スクリプトを作成してください。アップロード専用同期の場合を除き、Mobile Link サーバでは、少なくともダウンロードのダイレクトローハンドリングを有効にするように handle_DownloadData スクリプトが 1 つ定義されている必要があります。

ダイレクトダウンロードを handle_DownloadData 以外のイベントで作成する場合、そのイベントは begin_synchronization イベントより前または end_download イベントより後に実装できません。

注意
このイベントは SQL として実装できません。

参照

- 「データスクリプト」281 ページ
- 「ダイレクトローハンドリング」623 ページ
- 「ダイレクトダウンロード」633 ページ
- DownloadData インタフェース [Mobile Link サーバ Java]478 ページ
- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ
- DownloadData インタフェース [Mobile Link サーバ .NET]582 ページ
- DownloadTableData インタフェース [Mobile Link サーバ .NET]584 ページ
- 「handle_UploadData 接続イベント」383 ページ
- 「必要なスクリプト」255 ページ
- 「スクリプトの追加と削除」256 ページ

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handle_DownloadData 接続イベントに対して handleDownload という Java メソッドを登録します。Mobile Link 統合データベースに対してこのシステムプロシージャを実行します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_DownloadData',
 'MyPackage.MobiLinkOrders.handleDownload' )
```

「ml_add_java_connection_script システムプロシージャ」651 ページを参照してください。

次の例は、handleDownload メソッドを使用してダウンロードを作成する方法を示します。

次のコードは、クラスレベルの DBConnectionContext インスタンスを MobiLinkOrders クラスのコンストラクタ内で設定します。

```
import ianywhere.ml.script.*;
import java.io.*;
```

```
import java.sql.*;
import java.lang.System;

public class  MobiLinkOrders{

  DBConnectionContext _cc;

  public MobiLinkOrders( DBConnectionContext cc ) {
    _cc = cc;
  }}
```

HandleDownload メソッドでは、DBConnectionContext の getDownloadData メソッドを使用して、現在の同期に対する DownloadData インスタンスを返します。DownloadData の getDownloadTableByName メソッドは、remoteOrders テーブルの DownloadTableData インスタンスを返します。DownloadTableData の getUpsertPreparedStatement メソッドは java.sql.PreparedStatement を返します。ダウンロードに操作を追加するには、すべてのカラム値を設定して、executeUpdate メソッドを呼び出します。

次に示すのは、MobiLinkOrders クラスの handleDownload メソッドです。ここでは、remoteOrders テーブル用のダウンロードに 2 つのローを追加しています。

```
// Method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
  // Get DownloadData instance for current synchronization.
  DownloadData downloadData = _cc.getDownloadData();

  // Get a DownloadTableData instance for the remoteOrders table.
  DownloadTableData td = downloadData.getDownloadTableByName("remoteOrders");

  // Get a java.sql.PreparedStatement for upsert (update/insert) operations.
  PreparedStatement upsertPS = td.getUpsertPreparedStatement();

  // Set values for one row.
  upsertPS.setInt( 1, 2300 );
  upsertPS.setInt( 2, 100 );

  // Add the values to the download.
  int updateResult = upsertPS.executeUpdate();

  // Set values for another row.
  upsertPS.setInt( 1, 2301 );
  upsertPS.setInt( 2, 50 );
  updateResult = upsertPS.executeUpdate();
  // ...
  upsertPS.close();

}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、HandleDownload という .NET メソッドを handle_DownloadData 接続イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_connection_script(
  'ver1', 'handle_DownloadData',
  'TestScripts.MobiLinkOrders.HandleDownload'
)
```

次に示すのは、サンプルの .NET メソッド HandleDownload です。

```
using System;
using System.Data;
using System.IO;
using iAnywhere.MobiLink.Script;
using iAnywhere.MobiLink;

namespace MyScripts
{
   /// <summary>
   /// Tests that scripts are called correctly for most sync events.
   /// </summary>
   public class MobiLinkOrders
   {
 private DBConnectionContext _cc;

 public MobiLinkOrders( DBConnectionContext cc )
 {
    _cc = cc;
 }

 ~MobiLinkOrders()
 {
 }

 public void handleDownload()
 {
    // Get DownloadData instance for current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get a DownloadTableData instance for the remoteOrders table.
    DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");

    // Get an IDbCommand for upsert (update/insert) operations.
    IDbCommand upsert_stmt = td.GetUpsertCommand();

    IDataParameterCollection parameters = upsert_stmt.Parameters;

    // Set values for one row.
    parameters[ 0 ] = 2300;
    parameters[ 1 ] = 100;

    // Add the values to the download.
    int update_result = upsert_stmt.ExecuteNonQuery();

    // Set values for another row.
    parameters[ 0 ] = 2301;
    parameters[ 1 ] = 50;
    update_result = upsert_stmt.ExecuteNonQuery();

    // ...
 }

   }
}
```

handle_error 接続イベント

Mobile Link サーバでデータスクリプトの呼び出し中に SQL エラーが発生したときに実行されます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.action_code	INTEGER。これは INOUT パラメータです。この値は、Mobile Link サーバにエラーへの対処方法を指示するために設定します。	1
s.error_code	INTEGER。ネイティブの RDBMS エラーコード。	2
s.error_message	TEXT。ネイティブの RDBMS エラーメッセージ。	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。スクリプトにエラーがあるテーブル。スクリプトがテーブルスクリプトでない場合、テーブル名は NULL です。	5
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

Mobile Link サーバは、デフォルトのアクションを選択します。スクリプト内でアクションを修正し、Mobile Link に次の処理を指示する値を返すことができます。action_code パラメータには、次のいずれかの値を指定します。

- **1000**　現在のローをスキップして、処理を続行します。
- **3000**　現在のトランザクションをロールバックし、現在の同期をキャンセルします。これはデフォルトアクションコードで、handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合に使用されます。
- **4000**　現在のトランザクションのロールバック、同期のキャンセル、Mobile Link サーバの停止を行います。

備考

Mobile Link サーバは、現在のアクションコードで送信します。最初は 1 回の SQL 操作によって発生したエラーのセットごとに 3000 が設定されます。通常、エラー数は SQL 操作ごとに 1 つのみですが、複数の場合もあります。ローをバッチでアップロードする場合、この handle_error スクリプトは、バッチに含まれるエラーごとに 1 回呼び出されます。同じ同期中、最初のエラーに渡されるアクションコードは 3000 です。以降の呼び出しには、直前の呼び出しから返されたアクションコードが渡されます。Mobile Link は、複数の呼び出しから返される値のうち最も大きい番号が付いた値を使用します。

ローをバッチでアップロードする方法の詳細については、「-s mlsrv16 オプション」75 ページを参照してください。

スクリプト内でアクションコードを修正し、Mobile Link に次の処理を指示する値を返すことができます。Mobile Link サーバが次に何を行うかは、アクションコードに示されます。Mobile Link サーバは、エラーの重大度に応じてアクションコードにデフォルト値を設定してから、このスクリプトを呼び出します。この値は、スクリプトで変更できます。スクリプトは、必ずアクションコードを返すか設定するようにします。

エラーの内容は、error_code とメッセージで識別できます。

Mobile Link がアップロードトランザクション中に挿入、更新、または削除スクリプトを処理している間、またはダウンロードローをフェッチしている間に ODBC エラーが発生した場合、Mobile Link サーバはこのスクリプトを実行します。別のときに ODBC エラーが発生した場合、Mobile Link サーバは report_error または report_odbc_error スクリプトを呼び出して、同期をアボートします。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、クライアントアプリケーションでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、リモートデータベースのテーブル名が統合データベースのテーブルにどのようにマッピングされているかによって異なります。

handle_error イベント用の SQL スクリプトは、ストアドプロシージャとして実装してください。

次のいずれかの方法で、handle_error スクリプトから値を返すことができます。

●次のように、プロシージャの OUTPUT パラメータに action_code パラメータを渡す。

```
CALL my_handle_error( {ml s.action_code}, {ml s.error_code}, {ml s.error_message}, {ml
s.username}, {ml s.table} )
```

●次のように、プロシージャまたはファクションの戻り値を介して action_code を設定する。

```
{ml s.action_code} = CALL my_handle_error( {ml s.error_code}, {ml s.error_message}, {ml
s.username}, {ml s.table} )
```

ほとんどの RDBMS では、RETURN 文を使用してプロシージャまたはプロシージャからの戻り値を設定します。

CustDB サンプルアプリケーションには、さまざまなデータベース管理環境に対するエラーハンドラが含まれています。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「report_error 接続イベント」409 ページ
- 「report_odbc_error 接続イベント」413 ページ
- 「handle_odbc_error 接続イベント」379 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「データスクリプト」281 ページ

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これによって、アプリケーションは冗長挿入を無視できます。

次の Mobile Link システムプロシージャコールは、ULHandleError ストアドプロシージャを handle_error イベントに割り当てます。

```
CALL ml_add_connection_script(
   'ver1',
   'handle_error',
   'CALL ULHandleError(
     {ml s.action_code},
     {ml s.error_code},
     {ml s.error_message},
     {ml s.username},
     {ml s.table} )' )
```

次の SQL 文は ULHandleError ストアドプロシージャを作成します。

```
CREATE PROCEDURE ULHandleError(
  INOUT action integer,
  IN error_code integer,
  IN error_message varchar(1000),
  IN user_name varchar(128),
  IN table_name varchar(128) )
BEGIN
 -- -196 is SQLE_INDEX_NOT_UNIQUE
 -- -194 is SQLE_INVALID_FOREIGN_KEY
 IF error_code = -196 or error_code = -194 then
    -- ignore the error and keep going
    SET action = 1000;
```

```
  ELSE
     -- abort the synchronization
     SET action = 3000;
  END IF;
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handleError という Java メソッドを handle_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'handle_error',
   'ExamplePackage.ExampleClass.handleError' )
```

次に示すのは、サンプルの Java メソッド handleError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラーコードも判別します。

```
package ExamplePackage;
public class ExampleClass
{
   public void handleError( ianywhere.ml.script.InOutInteger    actionCode,
            int              errorCode,
            String               errorMessage,
            String               user,
            String               table )
   {
   // -196 is SQLE_INDEX_NOT_UNIQUE
   // -194 is SQLE_INVALID_FOREIGN_KEY
   if( errorCode == -196 || errorCode == -194 ) {
      // ignore the error and keep going
      actionCode.setValue( 1000 );
   } else {
      // abort the synchronization
      actionCode.setValue( 3000 );
   }
   }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、HandleError という .NET メソッドを handle_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'handle_error',
 'TestScripts.Test.HandleError' )
```

次に示すのは、サンプルの .NET メソッド HandleError です。

```
namespace TestScripts
{
   public class Test
   {
   public void HandleError( ref int     actionCode,
            int       errorCode,
            string       errorMessage,
```

```
            string      user,
            string      table )
    {
        // -196 is SQLE_INDEX_NOT_UNIQUE
        // -194 is SQLE_INVALID_FOREIGN_KEY
        if( errorCode == -196 || errorCode == -194 ) {
        // ignore the error and keep going
        actionCode = 1000;
        } else {
        // abort the synchronization
        actionCode = 3000;
        }
    }
    }
}
```

handle_odbc_error 接続イベント

Mobile Link サーバでデータスクリプトの呼び出し中に ODBC エラーが発生したときに実行されます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.action_code	INTEGER。これは INOUT パラメータです。この値は、Mobile Link サーバにエラーへの対処方法を指示するために設定します。	1
s.odbc_state	VARCHAR(5)。ODBC SQLSTATE。	2
s.error_message	TEXT。ODBC エラーメッセージ。	3

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。テーブル名。	5
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

Mobile Link サーバは、デフォルトのアクションを選択します。スクリプト内でアクションを修正し、Mobile Link に次の処理を指示する値を返すことができます。action_code パラメータには、次のいずれかの値を指定します。

- **1000**　現在のローをスキップして、処理を続行します。
- **3000**　現在のトランザクションをロールバックし、現在の同期をキャンセルします。これはデフォルトアクションコードで、handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合に使用されます。
- **4000**　現在のトランザクションのロールバック、同期のキャンセル、Mobile Link サーバの停止を行います。

備考

Mobile Link がアップロードトランザクション中に挿入、更新、または削除スクリプトを処理している間、またはダウンロードローをフェッチしている間に発生し、ODBC ドライバマネージャによってフラグが設定されたエラーを検出した場合、Mobile Link サーバはこのスクリプトを実行します。別のときに ODBC エラーが発生した場合、Mobile Link サーバは report_error または report_odbc_error スクリプトを呼び出して、同期をアボートします。

エラーの内容は、エラーコードで識別できます。

Mobile Link サーバが次に何を行うかは、アクションコードに示されます。Mobile Link サーバは、エラーの重大度に応じてアクションコードにデフォルト値を設定してから、このスクリプトを呼

び出します。この値は、スクリプトで変更できます。スクリプトは、必ずアクションコードを返すか設定するようにします。

handle_odbc_error スクリプトは、handle_error スクリプトと report_error スクリプトの後、report_odbc_error スクリプトの前に呼び出されます。

エラー処理スクリプトがどちらか 1 つだけ定義されている場合、そのスクリプトからの戻り値によってエラー処理が決定されます。エラー処理スクリプトが両方とも定義されている場合、Mobile Link サーバは数値の一番大きいアクションコードを使用します。handle_error と handle_ODBC_error の両方が定義されると、Mobile Link はすべての呼び出しから返されるアクションコードのうち、最も大きい番号を持つものを使用します。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「handle_error 接続イベント」374 ページ
- 「report_error 接続イベント」409 ページ
- 「report_odbc_error 接続イベント」413 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これによって、アプリケーションは ODBC 整合性制約違反を無視できます。

次の Mobile Link システムプロシージャコールは、HandleODBCError ストアドプロシージャを handle_odbc_error イベントに割り当てます。

```
CALL ml_add_connection_script(
 'ver1',
 'handle_odbc_error',
 'CALL HandleODBCError(
   {ml s.action_code},
   {ml s.ODBC_state},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )' )
```

次の SQL 文は HandleODBCError ストアドプロシージャを作成します。

```
CREATE PROCEDURE HandleODBCError(
   INOUT action integer,
   IN odbc_state varchar(5),
   IN error_message varchar(1000),
   IN user_name varchar(128),
   IN table_name varchar(128) )
  BEGIN
   IF odbc_state = '23000' then
      -- Ignore the error and keep going.
      SET action = 1000;
   ELSE
      -- Abort the synchronization.
      SET action = 3000;
   END IF;
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handleODBCError という Java メソッドを handle_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_odbc_error',
 'ExamplePackage.ExampleClass.handleODBCError'
)
```

次に示すのは、サンプルの Java メソッド handleODBCError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラーコードも判別します。

```
package ExamplePackage;
public class ExampleClass
{
   public void handleODBCError( ianywhere.ml.script.InOutInteger    actionCode,
            String                  odbcState,
            String                  errorMessage,
            String                  user,
            String                  table )
   {
   if( odbcState == "23000" ) {
      // Ignore the error and keep going.
      actionCode.setValue( 1000 );
   } else {
      // Abort the synchronization.
      actionCode.setValue( 3000 );
   }
   }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、HandleODBCError という .NET メソッドを handle_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'handle_odbc_error',
 'TestScripts.Test.HandleODBCError' )
```

次に示すのは、サンプルの .NET メソッド HandleODBCError です。

```
namespace TestScripts
{
   public class Test
   {
   public void HandleODBCError( ref int    actionCode,
            string    odbcState,
            string    errorMessage,
            string    user,
            string    table )
   {
      if( odbcState == "23000" ) {
      // Ignore the error and keep going.
      actionCode = 1000;
      } else {
```

```
            // Abort the synchronization.
            actionCode = 3000;
            }
        }
        }
    }
```

handle_UploadData 接続イベント

ダイレクトローハンドリングにおいて、アップロードされたローの処理に使用される非 SQL データスクリプトです。

パラメータ

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
UploadData	.NET または Java クラスは、Mobile Link クライアントによってアップロードされたテーブル操作をカプセル化します。このクラスは、Java と .NET 用 Mobile Link サーバ API で定義されています。	1
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

handle_UploadData イベントを使用すると、Mobile Link のダイレクトローハンドリングでアップロードを処理できます。このイベントは、同期のアップロードトランザクションごとに発生します。ただし、トランザクションレベルアップロードを使用している場合は、トランザクションごとに発生します。

「ダイレクトローハンドリング」623 ページを参照してください。

このイベントは 1 つの UploadData パラメータを取ります。Java または .NET メソッドは UploadData の getUploadedTables または getUploadedTableByName メソッドを使用して、UploadedTableData インスタンスを取得できます。UploadedTableData を使用して、現在の同期で Mobile Link クライアントによってアップロードされた挿入、更新、削除操作にアクセスできます。

UploadData と UploadedTableData クラスの詳細については、「ダイレクトアップロード」627 ページを参照してください。

カラム名は、デフォルトでは最初の同期で常に Mobile Link サーバに送信され、再送信を回避するために Mobile Link サーバによってキャッシュされます。必要に応じて、ml_add_column システムプロシージャを使用してカラム名を設定できます (廃止)。それ以外の方法でカラムを参照するには、リモートデータベースに定義されているインデックスを使用します。

更新のアップロードされた更新前イメージカラムを取得するには、SetOldRowValues メソッドと SetNewRowValues メソッドを使用します。「ダイレクトアップロードでの競合」629 ページを参照してください。

注意
このイベントは SQL として実装できません。

参照

- 「データスクリプト」281 ページ
- 「ダイレクトローハンドリング」623 ページ
- 「ダイレクトアップロード」627 ページ
- UploadData インタフェース [Mobile Link サーバ Java]524 ページ
- UploadedTableData インタフェース [Mobile Link サーバ Java]526 ページ
- UploadData インタフェース [Mobile Link サーバ .NET]612 ページ
- UploadedTableData インタフェース [Mobile Link サーバ .NET]614 ページ
- 「handle_DownloadData 接続イベント」371 ページ
- 「必要なスクリプト」255 ページ
- 「スクリプトの追加と削除」256 ページ

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して handleUpload という Java メソッドを登録します。Mobile Link 統合データベースに対してこのシステムプロシージャを実行します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_UploadData',
 'MyPackage.MyClass.handleUpload' )
```

ml_add_java_connection_script の詳細については、「ml_add_java_connection_script システムプロシージャ」651 ページを参照してください。

次の Java メソッドは remoteOrders テーブルのアップロードを処理します。UploadData.getUploadedTableByName メソッドは remoteOrders テーブルの UploadedTableData インスタンスを返します。UploadedTableData の getInserts メソッドは、新しいローを表す java.sql.ResultSet インスタンスを返します。

```
package MyPackage;
import ianywhere.ml.script.*;
import java.sql.*;
import java.io.*;
// ...
public class MyClass {
   String _curUser = null;
public void handleUpload( UploadData ut )
  throws SQLException, IOException {
```

```
    // Get an UploadedTableData instance representing the
    // remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");
    // Get inserts uploaded by the MobiLink client.
    java.sql.ResultSet results = remoteOrdersTable.getInserts();
    while( results.next() ) {
      // Get the primary key.
      int pk = results.getInt("pk");

      // Get the uploaded num_ordered value.
      int numOrdered = results.getInt("num_ordered");

      // The current insert row is now ready to be uploaded to wherever
      // you want it to go (a file, a web service, and so on).

    }

    results.close();

  }}
```

次の例は、Mobile Link リモートデータベースによってアップロードされた挿入、更新、削除操作を出力します。UploadData の getUploadedTables メソッドは、リモートによってアップロードされたすべてのテーブルを表す UploadedTableData インスタンスを返します。配列内でのテーブルの順序は、リモートによってアップロードされた順序と同じです。UploadedTableData の getInserts、getUpdates、getDeletes メソッドは標準 JDBC 結果セットを返します。println メソッドまたは出力データを使用して、テキストファイルまたは別の場所に出力できます。

```
import ianywhere.ml.script.*;
import java.sql.*;
import java.io.*;
// ...

public void handleUpload( UploadData ud )
  throws SQLException, IOException {
  UploadedTableData tables[] = ud.getUploadedTables();
  for( int i = 0; i < tables.length; i++ ) {
    UploadedTableData currentTable = tables[i];
    println( "table " + java.lang.Integer.toString( i ) +
     " name: " + currentTable.getName() );
    // Print out insert result set.
    println( "Inserts" );
    printRSInfo( currentTable.getInserts() );
    // print out update result set
    println( "Updates" );
    printUpdateRSInfo( currentTable.getUpdates() );
    // Print out delete result set.
    println( "Deletes" );
    printRSInfo( currentTable.getDeletes() );
  }
}
```

printRSInfo メソッドは、挿入、更新、または削除の結果セットを出力し、1 つの java.sql.ResultSet オブジェクトを受け入れます。カラムラベルを含む、詳細なカラム情報は、ResultSet の getMetaData メソッドによって返される ResultSetMetaData オブジェクトによって提供されます。printRow メソッドは結果セットの各ローを出力します。

```
public void printRSInfo( ResultSet results )
  throws SQLException, IOException {
```

```
  // Obtain the result set metadata.
  ResultSetMetaData metaData = results.getMetaData();
  String columnHeading = "";

  // Print out column headings.
  for( int c = 1; c <= metaData.getColumnCount(); c++ ) {
    columnHeading += metaData.getColumnLabel(c);
    if( c < metaData.getColumnCount() ) {
      columnHeading += ", ";
    }
  }

  println( columnHeading );
  while( results.next() ) {

    // Print out each row.
    printRow( results, metaData.getColumnCount() );
  }

  // Close the java.sql.ResultSet.
  results.close();
}
```

次に示す printRow メソッドは ResultSet の getString メソッドを使用して、各カラム値を取得します。

```
public void printRow( ResultSet results, int colCount )
  throws SQLException, IOException {
  String row = "( ";

  for( int c = 1; c <= colCount; c++ ) {
    // Get a column value.
    String currentColumn = results.getString( c );

    // Check for null values.
    if( currentColumn == null ) {
      currentColumn = "<NULL>";
    }

    // Add the column value to the row string.
    row += cur_col;
    if( c < colCount ) {
      row += ", ";
    }
  }

  row += " )";

  // Print out the row.
  println( row );
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して HandleUpload という .NET メソッドを登録します。Mobile Link 統合データベースに対してこのシステムプロシージャを実行します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'handle_UploadData',
 'TestScripts.Test.HandleUpload' )
```

次の .NET メソッドは remoteOrders テーブルのアップロードを処理します。この例では、SetOldRowValues メソッドと SetNewRowValues メソッドを使用して、各更新の更新前イメージと更新後イメージの両方にアクセスします。

```
using System;
using System.Data;
using System.IO;
using iAnywhere.MobiLink.Script;
using iAnywhere.MobiLink;

namespace MyScripts
{
   public class MyUpload
   {
 public MyUpload( DBConnectionContext cc )
 {
 }

 ~MyUpload()
 {
 }

 public void handleUpload( UploadData ut )
 {
    int i;
    UploadedTableData[] tables = ut.GetUploadedTables();

    for( i=0; i<tables.Length; i+=1 ) {
  UploadedTableData cur_table = tables[i];
  Console.Write( "table " + i + " name: " + cur_table.GetName() );

  // Print out insert result set.
  Console.Write( "Inserts" );
  printRSInfo( cur_table.GetInserts() );

  // print out update result set
  Console.Write( "Updates" );
  printUpdateRSInfo( cur_table.GetUpdates() );

  // Print out delete result set.
  Console.Write( "Deletes" );
  printRSInfo( cur_table.GetDeletes() );
    }
 }

 public void printRSInfo( IDataReader dr )
 {
    // Obtain the result set metadata.
    DataTable dt = dr.GetSchemaTable();
    DataColumnCollection cc = dt.Columns;
    DataColumn dc;
    String columnHeading = "";

    // Print out column headings.
    for( int c=0; c < cc.Count; c = c + 1 ) {
  dc = cc[ c ];
  columnHeading += dc.ColumnName;
  if( c < cc.Count - 1 ) {
     columnHeading += ", ";
  }
    }
    Console.Write( columnHeading );
```

```
    while( dr.Read() ) {
 // Print out each row.
 printRow( dr, cc.Count );
    }

    // Close the java.sql.ResultSet.
    dr.Close();
}

public void printUpdateRSInfo( UpdateDataReader utr )
{
    // Obtain the result set metadata.
    DataTable dt = utr.GetSchemaTable();
    DataColumnCollection cc = dt.Columns;
    DataColumn dc;
    String columnHeading = "TYPE, ";

    // Print out column headings.
    for( int c = 0; c < cc.Count; c = c + 1 ) {
 dc = cc[ c ];
 columnHeading += dc.ColumnName;
 if( c < cc.Count - 1 ) {
     columnHeading += ", ";
 }
    }
    Console.Write( columnHeading );

    while( utr.Read() ) {
 // Print out the new values for the row.
 utr.SetNewRowValues();
 Console.Write( "NEW:" );
 printRow( utr, cc.Count );

 // Print out the old values for the row.
 utr.SetOldRowValues();
 Console.Write( "OLD:" );
 printRow( utr, cc.Count );
    }

    // Close the java.sql.ResultSet.
    utr.Close();
}

public void printRow( IDataReader dr, int col_count )
{
    String row = "( ";
    int c;

    for( c = 0; c < col_count; c = c + 1 ) {
 // Get a column value.
 String cur_col = dr.GetString( c );

 // Check for null values.
 if( cur_col == null ) {
     cur_col = "<NULL>";
 }

 // Add the column value to the row string.
 row += cur_col;
 if( c < col_count ) {
     row += ", ";
 }
    }
```

```
        row += " )";

        // Print out the row.
        Console.Write( row );
    }
      }
  }
```

modify_error_message 接続イベント

このスクリプトを使用すると、リモートデータベースに送信される (エラー、警告、情報) メッセージのテキストをカスタマイズできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.error_message	VARBINARY(1024)。これはエラーメッセージを表す INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.error_code	INTEGER。error_message に関連付けられている Mobile Link エラーコード。	3

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトを使用すると、error_message を、リモートユーザやアプリケーションが元のメッセージよりも理解しやすいメッセージに変更できます。

modify_error_message イベントの SQL スクリプトは、ストアドプロシージャとして実装してください。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、データベースが 1 日前から現在までの間に同期されたかどうかに関わらず、1 日前からのデータをすべてダウンロードします。

次の SQL 文は ModifyLastErrorMessage ストアドプロシージャを作成します。

```
CREATE PROCEDURE ModifyLastErrorMessage(
  inout error_message VARBINARY(1024),
  in username VARCHAR(128),
  in error_code INT )
BEGIN
  SELECT dateadd(day, -1, last_download_time )
  INTO last_download_time
END
```

次の Mobile Link システムプロシージャコールは ModifyLastErrorMessage を、スクリプトバージョン modify_ts_test の modify_error_message 接続イベントに割り当てます。

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_error_message',
  'CALL ModifyLastErrorMessage (
    {ml s.error_message},
    {ml s.username},
    {ml s.error_code} )' );
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、modifyLastErrorMessage という Java メソッドを modify_error_message 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'modify_error_message',
 'ExamplePackage.ExampleClass.modifyLastErrorMessage' )
```

次に示すのは、サンプルの Java メソッド modifyLastErrorMessage です。このメソッドは、現在のエラーメッセージとエラーコードを出力します。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void modifyLastErrorMessage(
  ianywhere.ml.script.InOutString lastErrorMessage,
  String userName,
  int errorCode ) {
  java.lang.System.out.println( "error message: " +
   lastErrorMessage );
  java.lang.System.out.println( "error code: " +
   String.valueOf(errorCode) );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ModifyLastErrorMessage という .NET メソッドを modify_error_message 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'modify_error_message',
 'TestScripts.Test.ModifyLastErrorMessage' )
```

次に示すのは、サンプルの .NET メソッド ModifyLastErrorMessage です。このメソッドは、現在のエラーコードとエラーメッセージを出力します。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void ModifyLastErrorMessage (
  ref string errorMessage,
  string userName,
  string errorCode  ) {
  System.Console.WriteLine( "error message: " + errorMessage );
  System.Console.WriteLine( "error code: " +  errorCode );
  }}}
```

modify_last_download_timestamp 接続イベント

このスクリプトを使用して、現在の同期の最終ダウンロード時刻を修正できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_download	TIMESTAMP。同期テーブルの最初のダウンロード時刻。これは INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトは、現在の同期に対する last_download タイムスタンプを修正するために使用します。このスクリプトが定義されている場合、Mobile Link サーバは、ダウンロードスクリプトに渡す last_download タイムスタンプとして、修正した last_download タイムスタンプを使用します。通常、このスクリプトは、リモートで失われたデータをリカバリするために使用します。つまり、last_download タイムスタンプを 1900-01-01 00:00 などの過去の時間にリセットして、次の同期ですべてのデータをダウンロードできます。また、DBMS のレプリケーションなどによって、統合データベースのテーブルに対する更新のタイムスタンプが実際の更新の時刻よりも早い

時刻になる場合、ダウンロード時にこれらの更新が欠落しないように、このスクリプトで最終ダウンロード時刻を調整できます。

modify_last_download_timestamp イベントの SQL スクリプトは、ストアドプロシージャとして実装してください。

このスクリプトは、同じトランザクション内にある prepare_for_download スクリプトの直前に実行されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ
- 「ダウンロードタイムスタンプの生成および使用方法」110 ページ
- 「modify_next_last_download_timestamp 接続イベント」394 ページ
- 「generate_next_last_download_timestamp イベント」368 ページ

SQL の例

次の SQL 文はストアドプロシージャを作成します。次は Oracle 統合データベース用の構文です。パラメータの入出力があるストアドプロシージャを Oracle で作成する場合、次のとおりパラメータが **IN OUT** とマーク付けされるようにします。

```
CREATE OR REPLACE PROCEDURE ModifyDownloadTimestamp (
   download_timestamp  IN OUT TIMESTAMP,
   user_name   IN VARCHAR)
AS
BEGIN
    -- N is the maximum replication latency in consolidated cluster
   download_timestamp := download_timestamp - 1;
END;
```

次は、SQL Anywhere、Adaptive Server Enterprise、SQL Server 統合データベース用の構文です。

```
CREATE PROCEDURE ModifyDownloadTimestamp
      @download_timestamp DATETIME OUTPUT,
      @user_name    VARCHAR( 128 )
      AS
      BEGIN
       -- N is the maximum replication latency in consolidated cluster
      SELECT @download_timestamp = @download_timestamp - N
      END
```

次の Mobile Link システムプロシージャコールは、ModifyDownloadTimestamp ストアドプロシージャを modify_last_download_timestamp イベントに割り当てます。次は SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_connection_script(
 'my_version',
 'modify_last_download_timestamp',
 '{CALL ModifyDownloadTimestamp(
  {ml s.last_download},
  {ml s.username} ) }' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、modifyLastDownloadTimestamp という Java メソッドを modify_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'modify_last_download_timestamp',
 'ExamplePackage.ExampleClass.modifyLastDownloadTimestamp' )
```

次に示すのは、サンプルの Java メソッド modifyLastDownloadTimestamp です。このメソッドは、現在の新しいタイムスタンプを出力し、渡されたタイムスタンプを修正します。

```
public void modifyLastDownloadTimestamp(
    Timestamp lastDownloadTime,
    String userName ) {
  java.lang.System.out.println( "old date: " +
   lastDownloadTime.toString() );
  lastDownloadTime.setDate(
   lastDownloadTime.getDate() -1 );
  java.lang.System.out.println( "new date: " +
   lastDownloadTime.toString() );
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ModifyLastDownloadTimestamp という .NET メソッドを modify_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'modify_last_download_timestamp',
 'TestScripts.Test.ModifyLastDownloadTimestamp' )
```

次に示すのは、サンプルの .NET メソッド ModifyLastDownloadTimestamp です。

```
public void ModifyLastDownloadTimestamp(
 ref DateTime lastDownloadTime,
 string userName ) {
 System.Console.WriteLine( "old date: " +
  last_download_time.ToString() );
 last_download_time = DateTime::Now;
 System.Console.WriteLine( "new date: " +
  last_download_time.ToString() );
 }
```

modify_next_last_download_timestamp 接続イベント

このスクリプトを使用して、次回の同期の最終ダウンロード時刻を修正できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.next_last_download	TIMESTAMP。これは INOUT パラメータです。Mobile Link サーバは、アップロードがコミットされた直後にこの値を生成します。	1
s.last_download	TIMESTAMP。これは現在の同期の最終ダウンロード時刻です。	2
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	3
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトを使用すると、next_last_download タイムスタンプを変更でき、次回の同期の last_download スタンプが効率的に変更されます。これにより、現在の同期に影響を与えずに次回の同期をリセットできます。通常の同期中、next_last_download は last_download の時刻より後になりますが、last_download と同じになる場合もあります。

modify_next_last_download_timestamp イベントの SQL スクリプトは、ストアドプロシージャとして実装してください。Mobile Link サーバは、ストアドプロシージャへの最初のパラメータとして next_last_download タイムスタンプを渡し、タイムスタンプをストアドプロシージャが渡した最初の値で置き換えます。

このスクリプトは、ユーザテーブルのダウンロード後にダウンロードトランザクションで実行されます。ただし、ダウンロードトランザクション中に変更されたローが次の同期時にダウンロードされるように、ストアドプロシージャの出力値はダウンロードトランザクションの開始時刻に対応している必要があります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ
- 「ダウンロードタイムスタンプの生成および使用方法」110 ページ
- 「modify_last_download_timestamp 接続イベント」391 ページ
- 「generate_next_last_download_timestamp イベント」368 ページ

SQL の例

次の例は、このスクリプトの適用例の 1 つを示します。ストアドプロシージャを作成します。次は SQL Anywhere 統合データベース用の構文です。

```
CREATE PROCEDURE ModifyNextDownloadTimestamp(
  inout next_last_download TIMESTAMP ,
  in last_download TIMESTAMP ,
  in user_name VARCHAR(128) )
  BEGIN
    SELECT dateadd(hour, -1, next_last_download )
    INTO next_last_download
END
```

スクリプトを SQL Anywhere 統合データベースにインストールします。

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_next_last_download_timestamp',
  'CALL  ModifyNextDownloadTimestamp (
   {ml s.next_last_download},
   {ml s.last_download},
   {ml s.username} )' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、modifyNextDownloadTimestamp という Java メソッドを modify_next_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'modify_next_last_download_timestamp',
   'ExamplePackage.ExampleClass.modifyNextDownloadTimestamp' )
```

次に示すのは、サンプルの Java メソッド modifyNextDownloadTimestamp です。このメソッドは、ダウンロードタイムスタンプを 1 時間前に設定します。

```
public void modifyNextDownloadTimestamp(
  Timestamp NextLastDownload,
  Timestamp lastDownload,
  String userName ) {
```

```
NextLastDownload.setHours(
NextLastDownload.getHours() -1 );
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ModifyNextDownloadTimestamp という .NET メソッドを modify_next_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'modify_next_last_download_timestamp',
  'TestScripts.Test.ModifyNextDownloadTimestamp' )
```

次に示すのは、サンプルの .NET メソッド ModifyNextDownloadTimestamp です。このメソッドは、ダウンロードタイムスタンプを 1 時間前に設定します。

```
using System;
using System.Data;
namespace TestScripts {
public class Test {
   String _curUser = null;
public void ModifyNextDownloadTimestamp (
  ref DateTime next_last_download,
  DateTime last_download,
  string user_name ) {
  next_last_download = next_last_download.AddHours( -1 );
  }}}
```

modify_user 接続イベント

Mobile Link ユーザ名を変更します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。これは INOUT パラメータです。	1
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトは、認証トランザクションの最後に呼び出されます。

Mobile Link サーバは、ユーザ名をパラメータとして指定します。このパラメータによってスクリプトが呼び出され、Mobile Link クライアントからユーザ名が送信されます。必要に応じて、代替ユーザ名を作成することもできます。このスクリプトを使用すると、Mobile Link スクリプトの呼び出しで使用するユーザ名を変更できます。

username パラメータは、ユーザ名を格納するのに十分な長さでなければなりません。

modify_user イベント用の SQL スクリプトは、ストアドプロシージャとして実装してください。

注意
より柔軟な方法で Mobile Link ユーザ名をマッピングするには、ユーザ定義の名前付きパラメータを使用します。 「ユーザ定義の名前付きパラメータ」250 ページを参照してください。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「authenticate_user 接続イベント」291 ページ
- 「authenticate_user_hashed 接続イベント」297 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、マッピングテーブル user_device を使用して、リモートデータベースのユーザ名をデバイスを使用しているユーザの ID にマッピングします。この方法は、同じ人物が複数のリモート (PDA やラップトップなど) を所有し、(ユーザの名前または ID に基づいて) 同じ同期論理が必要なときに使用できます。

次の Mobile Link システムプロシージャコールは、ModifyUser ストアドプロシージャを modify_user イベントに割り当てます。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_connection_script(
 'ver1',
 'modify_user',
 'call ModifyUser( {ml s.username} )' )
```

次の SQL 文は ModifyUser ストアドプロシージャを作成します。

```
CREATE PROCEDURE ModifyUser( INOUT u_name varchar(128) )
BEGIN
 SELECT user_name
   INTO u_name
   FROM user_device
   WHERE device_name = u_name;
END
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、modifyUser という Java メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_user',
  'ExamplePackage.ExampleClass.modifyUser' )
```

次に示すのは、サンプルの Java メソッド modifyUser です。このメソッドは、データベースからユーザ ID を取得し、それを使用してユーザ名を設定します。

```
package ExamplePackage;
import java.lang.Integer;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass
{
   DBConnectionContext curConn;

   public ExampleClass( DBConnectionContext cc )
   {
   curConn = cc;
   }

   public void modifyUser( InOutString ioUserName )
   throws SQLException
   {
   Connection conn = curConn.getConnection();
   PreparedStatement uidSelect =
      conn.prepareStatement( "SELECT rep_id FROM SalesRep WHERE name = ?" );
   try {
      uidSelect.setString( 1, ioUserName.getValue() );
```

```
            ResultSet uidResult = uidSelect.executeQuery();
            try {
            if( uidResult.next() ) {
                ioUserName.setValue( Integer.toString(uidResult.getInt( 1 )));
            }
            } finally {
            uidResult.close();
            }
        } finally {
            uidSelect.close();
        }
        }
    }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ModUser という .NET メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'modify_user',
 'TestScripts.Test.ModUser'
)
```

次に示すのは、サンプルの .NET メソッド ModUser です。

```
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
    DBConnectionContext curConn;

    public Test( DBConnectionContext cc )
    {
        curConn = cc;
    }

    public void ModifyUser( ref string ioUserName )
    {
        DBCommand cmd = curConn.GetConnection().CreateCommand();
        cmd.CommandText = "SELECT rep_id FROM SalesRep WHERE name = ?";
        cmd.Parameters[0] = ioUserName;
        DBRowReader r = cmd.ExecuteReader();
        object[] row;
        if( (row = r.NextRow()) != null ) {
        ioUserName = (string) row[0];
        }
    }
    }
}
```

nonblocking_download_ack 接続イベント

ダウンロード確認を使用する場合、このスクリプトは、ダウンロードが正常に適用されたという情報を記録したり、確認するダウンロードに依存するビジネスロジックを必要に応じてトリガしたりする場所を提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.last_download	TIMESTAMP。これは現在の同期の最終ダウンロード時刻です。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

備考

このイベントを使って、リモートデータベースでダウンロードが正常に適用された時間を記録できます。

このイベントが呼び出されるのは、ダウンロード確認を使用している場合のみです。ダウンロードが送信されたときに、ダウンロードトランザクションがコミットされ、同期が終了します。このイベントは、同期クライアントが正常なダウンロードを確認したときに呼び出されます。このイベントは、新しい接続で、元の同期の end_synchronization スクリプトの後に呼び出されます。このイベントのアクションは、Mobile Link システムテーブルのダウンロード時刻の更新とともにコミットされます。

このスクリプトが持つ特殊な性質によって、同期中に設定された接続レベルの変数は、このイベントを実行するときには使用できません。

> **注意**
> ダウンロードに失敗した場合、またはネットワーク接続が切断された場合は、確認は行われず、このスクリプトは呼び出されません。タイムリーなダウンロード確認が業務上重要な場合は、ダウンロード確認処理の予備として、prepare_for_download スクリプトの last_download パラメータまたは begin_publication スクリプトの last_publication_download パラメータを使用してください。

参照

- 「publication_nonblocking_download_ack 接続イベント」406 ページ
- dbmlsync： 「SendDownloadAck (sa) 拡張オプション」『Mobile Link クライアント管理』
- Ultra Light： 「Send Download Acknowledgement 同期パラメータ」『Ultra Light データベース管理とリファレンス』

SQL の例

次のスクリプトは、テーブル download_pubs_acked にレコードを追加します。レコードには、リモート ID、最初の認証パラメータ、ダウンロードタイムスタンプが含まれます。

```
INSERT INTO download_pubs_acked( rem_id, auth_parm, last_download )
  VALUES( {ml s.remote_id},  {ml a.1}, {ml s.last_download} )
```

Java の例

次の Mobile Link システムプロシージャの呼び出しは、スクリプトバージョン ver1 を同期するときに、confirmDownload という Java メソッドを nonblocking_download_ack イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'nonblocking_download_ack',
  'ExamplePackage.ExampleClass.confirmDownload' )
```

次に示すのは、サンプルの Java メソッド confirmDownload です。指定されたユーザと、指定されたタイムスタンプまでを対象として、確認するダウンロードに基づいてビジネスロジックを実行する Java メソッドを呼び出します。

```
package ExamplePackage;
import ianywhere.ml.script.*;
import java.sql.*;
public class ExampleClass
{
   DBConnectionContext _cc;

   public ExampleClass( DBConnectionContext cc )
   {
   _cc = cc;
   }

   public void confirmDownload( String user,
              Timestamp ts )
   throws SQLException
   {
   Connection conn = _cc.getConnection();
   PreparedStatement stmt = conn.prepareStatement(
      "INSERT INTO download_pubs_acked( rem_id, last_download ) " +
      "VALUES( ?,  ? )" );
   stmt.setString( 1, _cc.getRemoteID() );
```

```
        stmt.setTimestamp( 2, ts );
        stmt.executeUpdate();
        stmt.close();
        }
    }
```

.NET の例

次の Mobile Link システムプロシージャの呼び出しは、スクリプトバージョン ver1 を同期するときに、ConfirmDownload という .NET メソッドを nonblocking_download_ack 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'nonblocking_download_ack',
 'TestScripts.Test.ConfirmDownload'
)
```

次に示すのは、サンプルの .NET メソッド ConfirmDownload です。指定されたユーザと、指定されたタイムスタンプまでを対象として、確認されたダウンロードに基づいてビジネスロジックを実行する、.NET メソッドを呼び出します。

```
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
    DBConnectionContext _cc;

    public Test( DBConnectionContext cc )
    {
        _cc = cc;
    }

    public void ConfirmDownload( string user,
                 DateTime dt )
    {
        DBConnection conn = _cc.GetConnection();

        DBCommand cmd = conn.CreateCommand();
        cmd.CommandText =
            "INSERT INTO download_pubs_acked( rem_id, last_download ) " +
            "VALUES( ?,  ? )";
        cmd.Parameters[0] = _cc.GetRemoteID();
        cmd.Parameters[1] = dt;
        cmd.ExecuteNonQuery();
    }
    }
}
```

prepare_for_download 接続イベント

アップロードトランザクションとダウンロードトランザクション間で必要な操作を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.last_download	TIMESTAMP。同期テーブルの最初のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

Mobile Link サーバは、アップロードトランザクションからダウンロードトランザクションの開始までの間に、このスクリプトを別個のトランザクションで実行します。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「end_upload 接続イベント」358 ページ
- 「begin_download 接続イベント」302 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』
- 「スクリプトでの最終ダウンロード時刻」109 ページ

SQL の例

次の Mobile Link システムプロシージャの呼び出しは、スクリプトバージョン ver1 を同期するときに、prepareForDownload という SQL ストアドプロシージャを prepare_for_download イベント用のスクリプトとして登録します。

```
CALL ml_add_connection_script(
  'ver1',
  'prepare_for_download',
  'CALL prepareForDownload(
     { ml s.username } )' )
```

次に示すのは、サンプルの SQL メソッド prepareForDownload です。このストアドプロシージャは、2 つのテーブルのダウンロードを準備します。たとえば、多くのテーブルから情報を取得し、その情報をテーブル T1 と T2 用に download_cursor スクリプトによって参照されるテンポラリテーブルに格納します。

```
CREATE PROCEDURE prepareForDownload (
  IN ts TIMESTAMP,
  IN "user" VARCHAR(128))
BEGIN
  CALL prepareT1Download( user, ts );
  CALL prepareT2Download( user, ts );
END;
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、prepareForDownload という Java メソッドを prepare_for_download イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'prepare_for_download',
  'ExamplePackage.ExampleClass.prepareForDownload' )
```

次に示すのは、サンプルの Java メソッド prepareForDownload です。このメソッドは、2 つのテーブルのダウンロードを準備します。たとえば、多くのテーブルからの情報と Java からアクセス可能なその他の情報を取得し、その情報をテーブル T1 と T2 用に download_cursor スクリプトによって参照されるテンポラリテーブルに格納します。

```
public void prepareForDownload(
  Timestamp ts,
  String user ) {
  prepareT1ForDownload( _syncconn, user, ts );
  prepareT2ForDownload( _syncconn, user, ts );
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、PrepareForDownload という .NET メソッドを prepare_for_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'prepare_for_download',
 'TestScripts.Test.PrepareForDownload'
)
```

次に示すのは、サンプルの .NET メソッド PrepareForDownload です。このメソッドは、2 つのテーブルのダウンロードを準備します。たとえば、多くのテーブルからの情報と .NET からアクセス可能なその他の情報を取得し、その情報をテーブル T1 と T2 用に download_cursor スクリプトによって参照されるテンポラリテーブルに格納します。

```
public void PrepareForDownload(
  DateTime ts,
  string user ) {
  PrepareT1ForDownload ( _syncConn, user, ts );
  PrepareT2ForDownload ( _syncConn, user, ts );
  }
```

publication_nonblocking_download_ack 接続イベント

ダウンロード確認を使用する場合、このスクリプトは、パブリケーションが正常にダウンロードされたという情報を記録します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.last_publication_download	TIMESTAMP。同期テーブルの最後のダウンロード時刻。	2
s.publication name	VARCHAR(128)。パブリケーションの名前。	3
s.subscription_id	VARCHAR(128)。リモートサブスクリプション ID。	4
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

備考

このイベントを使って、リモートデータベースでこのパブリケーションのダウンロードが正常に適用された時間を記録できます。

このイベントが呼び出されるのは、ダウンロード確認を使用している場合のみです。非ブロッキングモードでは、ダウンロードが送信されたときに、ダウンロードトランザクションがコミットされ、同期が終了します。このイベントは、同期クライアントが正常なダウンロードを確認したときに、ダウンロードのパブリケーションごとに 1 回呼び出されます。このイベントは、新しい接続で、元の同期の end_synchronization スクリプトの後に呼び出されます。このイベントのアクションは、Mobile Link システムテーブルのダウンロード時刻の更新とともにコミットされます。

注意
ダウンロードに失敗した場合、またはネットワーク接続が切断された場合は、確認は行われず、このスクリプトは呼び出されません。タイムリーなダウンロード確認が業務上重要な場合は、ダウンロード確認処理の予備として、prepare_for_download スクリプトの last_download パラメータまたは begin_publication スクリプトの last_publication_download パラメータを使用してください。

このスクリプトが持つ特殊な性質によって、同期中に設定された接続レベルの変数は、このイベントを実行するときには使用できません。

参照

- 「nonblocking_download_ack 接続イベント」400 ページ
- dbmlsync： 「SendDownloadAck (sa) 拡張オプション」『Mobile Link クライアント管理』
- Ultra Light： 「Send Download Acknowledgement 同期パラメータ」『Ultra Light データベース管理とリファレンス』

SQL の例

次のスクリプトは、download_pubs_acked というテーブルにレコードを追加します。レコードには、パブリケーション名、最初の認証パラメータ、ダウンロードタイムスタンプが含まれます。

```
INSERT INTO download_pubs_acked( pub_name, auth_parm, last_download )
  VALUES( {ml s.publication_name},  {ml a.1}, {ml s.last_publication_download} )
```

Java の例

次の Mobile Link システムプロシージャの呼び出しは、スクリプトバージョン ver1 を同期するときに、publicationDownloadACK という Java メソッドを publication_nonblocking_download_ack 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script (
 'ver1',
 'publication_nonblocking_download_ack',
 'ExamplePackage.ExampleClass.publicationDownloadACK' )
```

次に示すのは、サンプルの Java メソッド publicationDownloadACK です。特に重要なパブリケーションがダウンロードされたかどうかを確認し、その結果に基づいて何らかのビジネスロジックを実行します。

```
package ExamplePackage;
import ianywhere.ml.script.*;
import java.sql.*;
public class ExampleClass
{
   DBConnectionContext _cc;

   public ExampleClass( DBConnectionContext cc )
   {
   _cc = cc;
   }

   public void confirmDownload( String user,
            Timestamp ts,
            String pubName )
   throws SQLException
   {
   Connection conn = _cc.getConnection();
   PreparedStatement stmt = conn.prepareStatement(
      "INSERT INTO download_pubs_acked( rem_id, last_download, pub_name ) " +
      "VALUES( ?,  ?, ? )" );
   stmt.setString( 1, _cc.getRemoteID() );
   stmt.setTimestamp( 2, ts );
   stmt.setString( 3, pubName );
   stmt.executeUpdate();
   stmt.close();
   }
}
```

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

.NET の例

次の Mobile Link システムプロシージャの呼び出しは、スクリプトバージョン ver1 を同期するときに、EndTableDownload という .NET メソッドを end_download テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'publication_nonblocking_download_ack',
  'TestScripts.Test.EndTableDownload'
)
```

次に示すのは、サンプルの .NET メソッド EndTableDownload です。特に重要なパブリケーションがダウンロードされたかどうかを確認し、その結果に基づいて何らかのビジネスロジックを実行します。

```
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
   public class Test
   {
   DBConnectionContext _cc;

   public Test( DBConnectionContext cc )
   {
      _cc = cc;
   }

   public void ConfirmDownload( string user,
                  DateTime dt,
                  string pubName )
   {
      DBConnection conn = _cc.GetConnection();

      DBCommand cmd = conn.CreateCommand();
      cmd.CommandText =
         "INSERT INTO download_pubs_acked( rem_id, last_download, pub_name ) " +
         "VALUES( ?,  ?, ? )";
      cmd.Parameters[0] = _cc.GetRemoteID();
      cmd.Parameters[1] = dt;
      cmd.Parameters[2] = pubName;
      cmd.ExecuteNonQuery();
   }
   }
}
```

report_error 接続イベント

エラーのログを取ったり、handle_error スクリプトによって選択されたアクションを記録したりできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.action_code	INTEGER。これは INOUT パラメータです。このパラメータは必須です。	1
s.error_code	INTEGER。ネイティブの DBMS エラーコード。	2
s.error_message	TEXT。ネイティブの DBMS エラーメッセージ。	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。スクリプトがエラーの原因となったテーブル。	5
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトを使用すると、エラーのログを取ったり、handle_error スクリプトによって選択されたアクションを記録したりできます。このスクリプトは、handle_error スクリプトが定義されているかどうかに関わらず handle_error イベントの後に実行されます。また、同期接続とは異なるデータベース接続 (管理／情報接続) の専用トランザクションで常に実行されます。

Mobile Link サーバでは、エラーが回復可能な場合は常にエラーをレポートし、handle_error スクリプトまたは handle_odbc_error スクリプトを呼び出そうとします。たとえば、Mobile Link サーバが挿入をアップロードしようとしたときにエラーが発生した場合、Mobile Link サーバはこのエラーをレポートし、hande_error スクリプトを呼び出します。handle_script から返されたアクションが 1000 の場合、サーバはエラーを無視して同期を続行します。ただし、Mobile Link サーバが統合データベースに何かを送信する前にエラーを検出した場合、エラーが回復不能なため

サーバはエラーをレポートしないことがあります。より厳密には、Mobile Link サーバでは、ODBC ドライバと統合データベースによって生成されたエラーをレポートします。

エラーの内容は、エラーコードとエラーメッセージで識別できます。現在のエラーの原因となった SQL 操作について、エラー処理スクリプトの最後の呼び出しによってアクションコード値が返されます。

同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、リモートデータベースでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、リモートデータベースのテーブル名が統合データベースのテーブル名にどのようにマッピングされているかによって異なります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「handle_error 接続イベント」374 ページ
- 「handle_odbc_error 接続イベント」379 ページ
- 「report_odbc_error 接続イベント」413 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これは、同期エラーを記録するために使用されるテーブルにローを挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'report_error',
 'INSERT INTO sync_error(
   action_code,
   error_code,
   error_message,
   user_name,
   table_name )
  VALUES (
   {ml s.action_code},
   {ml s.error_code},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )'  )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、reportError という Java メソッドを report_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'report_error',
   'ExamplePackage.ExampleClass.reportError' )
```

次に示すのは、サンプルの Java メソッド reportError です。このメソッドは、Mobile Link が提供する JDBC 接続を使用してテーブルにエラーのログを取ります。また、アクションコードも設定します。

```
package ExamplePackage;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass
{
   DBConnectionContext _cc;

   public ExampleClass( DBConnectionContext cc )
   {
   _cc = cc;
   }

   public void reportError( ianywhere.ml.script.InOutInteger    actionCode,
            int              errorCode,
            String              errorMessage,
            String              user,
            String              table )
   throws SQLException
   {
   actionCode.setValue( errorCode );
   // Insert error information in a table,
   Connection conn = _cc.getConnection();
   PreparedStatement stmt = conn.prepareStatement(
      "INSERT INTO sync_error( action_code, error_code, error_message, " +
      "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )" );
   stmt.setInt( 1, actionCode.getValue() );
   stmt.setInt( 2, errorCode );
   stmt.setString( 3, errorMessage );
   stmt.setString( 4, user );
   stmt.setString( 5, table );
   stmt.executeUpdate();
   stmt.close();
   }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ReportError という .NET メソッドを report_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'report_error',
  'TestScripts.Test.ReportError' )
```

次に示すのは、サンプルの .NET メソッド ReportError です。このメソッドは、.NET メソッドを使用してテーブルにエラーのログを取ります。

```
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
   public class Test
   {
   DBConnectionContext _cc;
```

```
    public Test( DBConnectionContext cc )
    {
        _cc = cc;
    }

    public void ReportError( ref int actionCode,
                int errorCode,
                string errorMessage,
                string user,
                string table )
    {
        actionCode = errorCode;
        DBConnection conn = _cc.GetConnection();

        DBCommand cmd = conn.CreateCommand();
        cmd.CommandText =
        "INSERT INTO sync_error( action_code, error_code, error_message, " +
        "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )";
        cmd.Parameters[0] = actionCode;
        cmd.Parameters[1] = errorCode;
        cmd.Parameters[2] = errorMessage;
        cmd.Parameters[3] = user;
        cmd.Parameters[4] = table;
        cmd.ExecuteNonQuery();
    }
    }
}
```

report_odbc_error 接続イベント

エラーのログを取ったり、handle_odbc_error スクリプトによって選択されたアクションを記録したりできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.action_code	INTEGER。これは INOUT パラメータです。このパラメータは必須です。	1
s.odbc_state	VARCHAR(5)。ODBC SQLSTATE。	2

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.error_message	TEXT。ODBC エラーメッセージ。	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。スクリプトがエラーの原因となったテーブル。	5
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

このスクリプトを使用すると、エラーのログを取ったり、handle_odbc_error スクリプトによって選択されたアクションを記録したりできます。このスクリプトは、handle_odbc_error スクリプトが定義されているかどうかに関わらず handle_odbc_error イベントの後に実行されます。また、同期接続とは異なるデータベース接続 (管理／情報接続) の専用トランザクションで常に実行されます。

エラーの内容は、ODBC ステータスとエラーメッセージで識別できます。現在のエラーの原因となった SQL 操作について、エラー処理スクリプトの最後の呼び出しによってアクションコード値が返されます。

同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、リモートデータベースでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、リモートデータベースのテーブル名が統合データベースのテーブル名にどのようにマッピングされているかによって異なります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「handle_error 接続イベント」374 ページ
- 「handle_odbc_error 接続イベント」379 ページ
- 「report_error 接続イベント」409 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これは、同期エラーを記録するために使用されるテーブルにローを挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'report_odbc_error',
 'INSERT INTO sync_error(
   action_code,
   odbc_state,
   error_message,
   user_name,
   table_name )
  VALUES(
   {ml s.action_code},
   {ml s.odbc_state},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、reportODBCError という Java メソッドを report_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'report_odbc_error',
  'ExamplePackage.ExampleClass.reportODBCError' )
```

次に示すのは、サンプルの Java メソッド reportODBCError です。このメソッドは、Mobile Link が提供する JDBC 接続を使用してテーブルにエラーのログを取ります。また、アクションコードも設定します。

```
package ExamplePackage;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass
{
   DBConnectionContext _cc;

   public ExampleClass( DBConnectionContext cc )
   {
   _cc = cc;
   }

   public void reportODBCError( InOutInteger    actionCode,
            String       odbcState,
            String       odbcMessage,
```

```
                String      user,
                String      table )
    throws SQLException
    {
    // Insert error information in a table,
    Connection    conn   = _cc.getConnection();
    PreparedStatement stmt = conn.prepareStatement(
        "INSERT INTO sync_error( action_code, odbc_state, error_message, " +
        "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )" );
    stmt.setInt( 1, actionCode.getValue() );
    stmt.setString( 2, odbcState );
    stmt.setString( 3, odbcMessage );
    stmt.setString( 4, user );
    stmt.setString( 5, table );
    stmt.executeUpdate();
    stmt.close();
    }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ReportODBCError という .NET メソッドを report_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'report_odbc_error',
  'TestScripts.Test.ReportODBCError' )
```

次に示すのは、サンプルの .NET メソッド ReportODBCError です。このメソッドは、.NET メソッドを使用してテーブルにエラーのログを取ります。

```
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
    DBConnectionContext _cc;

    public Test( DBConnectionContext cc )
    {
        _cc = cc;
    }

    public void ReportODBCError( ref int actionCode,
                string odbcState,
                string errorMessage,
                string user,
                string table )
    {
        DBConnection conn = _cc.GetConnection();

        DBCommand cmd = conn.CreateCommand();
        cmd.CommandText =
        "INSERT INTO sync_error( action_code, odbc_state, error_message, " +
        "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )";
        cmd.Parameters[0] = actionCode;
        cmd.Parameters[1] = odbcState;
        cmd.Parameters[2] = errorMessage;
        cmd.Parameters[3] = user;
```

```
            cmd.Parameters[4] = table;
            cmd.ExecuteNonQuery();
        }
        }
    }
```

resolve_conflict テーブルイベント

特定のテーブルの競合を解決する処理を定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

リモートデータベースでローが更新されると、Mobile Link クライアントは元の値のコピーを保存します。クライアントは、Mobile Link サーバに古い値と新しい値の両方を送信します。

Mobile Link サーバは更新されたローを受信すると、元の値と統合データベース内の現在の値を比較します。比較は、upload_fetch スクリプトを使用して行われます。

アップロードされた古い値が統合データベース内の現在の値と一致しない場合は、そのローに競合が発生します。ローを更新する代わりに、Mobile Link サーバは古い値と新しい値の両方を統合データベースに挿入します。古いローと新しいローは、それぞれスクリプト upload_old_row_insert と upload_new_row_insert を使用して処理されます。

値が挿入されると、Mobile Link サーバは resolve_conflict スクリプトを実行します。ここで、競合を解決できます。どのスキームでも選択して実装できます。

このスクリプトは競合ごとに 1 回実行されます。

別の方法として、resolve_conflict スクリプトを定義する代わりに、end_upload_rows スクリプトまたは end_upload テーブルスクリプトで競合解決論理を使用して、集合指向型の方法で競合を解決することもできます。

リモートデータベースのテーブルごとに、resolve_conflict スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「upload_old_row_insert テーブルイベント」443 ページ
- 「upload_new_row_insert テーブルイベント」441 ページ
- 「upload_update テーブルイベント」455 ページ
- 「end_upload_rows テーブルイベント」366 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の文は、Oracle インストール環境用の CustDB サンプルアプリケーションに適した resolve_conflict スクリプトを定義します。このスクリプトは、ストアドプロシージャ ULResolveOrderConflict を呼び出します。

```
exec ml_add_table_script(
  'custdb', 'ULOrder', 'resolve_conflict',
  'begin ULResolveOrderConflict();
end; ')
CREATE OR REPLACE PROCEDURE ULResolveOrderConflict()
AS
  new_order_id integer;
  new_status   varchar(20);
  new_notes   varchar(50);
BEGIN
  -- approval overrides denial
  SELECT order_id, status, notes
    INTO new_order_id, new_status, new_notes
    FROM ULNewOrder
   WHERE syncuser_id = SyncUserID;
```

```
  IF new_status = 'Approved' THEN
    UPDATE ULOrder o
      SET o.status = new_status, o.notes =
       new_notes
    WHERE o.order_id = new_order_id;
  END IF;
  DELETE FROM ULOldOrder;
  DELETE FROM ULNewOrder;
END;
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、resolveConflict という Java メソッドを resolve_conflict テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'resolve_conflict',
  'ExamplePackage.ExampleClass.resolveConflict' )
```

次に示すのは、サンプルの Java メソッド resolveConflict です。このメソッドは、競合を解決するために Mobile Link が提供する JDBC 接続を使用する Java メソッドを呼び出します。

```
package ExamplePackage;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass
{
   DBConnectionContext _cc;

   public ExampleClass( DBConnectionContext cc )
   {
   _cc = cc;
   }

   public void resolveConflict( String user,
            String table )
   throws java.sql.SQLException
   {
   if( table == "Order" ) {
      // Insert error information in a table,
      Connection    conn   = _cc.getConnection();
      String conflictTable = "New" + table;
      PreparedStatement stmt = conn.prepareStatement(
      "SELECT order_id, new_status, new_notes " +
      "FROM " + conflictTable +
      "WHERE rid = " + _cc.getRemoteID() );
      ResultSet rs = stmt.executeQuery();
      PreparedStatement updt = conn.prepareStatement(
      "UPDATE ULOrder SET status = ?, notes = ? WHERE order_id = ?" );

      while( rs.next() ) {
      if( rs.getString( 2 ) == "Approved" ) {
         updt.setString( 1, rs.getString( 2 ) );
         updt.setString( 2, rs.getString( 3 ) );
         updt.setInt( 3, rs.getInt( 1 ) );
      }
      }
      updt.close();
      stmt.close();
```

```
    }
    }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、ResolveConflict という .NET メソッドを resolve_conflict テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'resolve_conflict',
  'TestScripts.Test.ResolveConflict' )
```

次に示すのは、サンプルの .NET メソッド ResolveConflict です。このメソッドは、競合を解決する .NET メソッドを呼び出します。

```
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
   public class Test
   {
   DBConnectionContext _cc;

   public Test( DBConnectionContext cc )
   {
      _cc = cc;
   }

   public void ResolveConflict( string user,
                 string table )
   {
      if( table == "Order" ) {
      // Insert error information in a table,
      DBConnection    conn   = _cc.GetConnection();
      String conflictTable = "New" + table;
      DBCommand cmd = conn.CreateCommand();
      cmd.CommandText =
         "SELECT order_id, new_status, new_notes " +
         "FROM " + conflictTable +
         "WHERE rid = " + _cc.GetRemoteID();
      DBRowReader dr = cmd.ExecuteReader();
      DBCommand updt = conn.CreateCommand();
      updt.CommandText =
         "UPDATE ULOrder SET status = ?, notes = ? WHERE order_id = ?";

      object[] row;
      while( (row = dr.NextRow() ) != null ) {
         if( row.[1].Equals( "Approved" ) ) {
         updt.Parameters[0] = row[1];
         updt.Parameters[1] = row[2];
         updt.Parameters[2] = row[0];
         }
      }
      updt.Close();
      cmd.Close();
      conn.Close();
      }
   }
```

```
    }
  }
```

synchronization_statistics 接続イベント

同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.warnings	INTEGER。同期中に発行された警告の数。	2
s.errors	INTEGER。同期で発生したエラーの数。	3
s.deadlocks	INTEGER。同期で検出された統合データベース内のデッドロックの数。	4
s.synchronized_tables	INTEGER。同期に関係したクライアントテーブルの数。	5
s.connection_retries	INTEGER。Mobile Link サーバが統合データベースとの接続をリトライした回数。	6

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

synchronization_statistics イベントを使用すると、任意のユーザと接続について、現在の同期に関する各種の統計を収集できます。最後の同期トランザクション終了時のコミット直前に、synchronization_statistics 接続スクリプトが呼び出されます。

注意
コマンドラインによっては、一部の警告のログが取られないことがあります。このスクリプトに渡される警告数は、警告が無効になっていない場合にログに取られる警告数であり、ログに取られる警告数より多くなることがあります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_statistics 接続イベント」334 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、同期の統計を sync_con_audit テーブルに挿入します。

```
CALL ml_add_connection_script(
  'ver1',
  'synchronization_statistics',
 'INSERT INTO sync_con_audit(
   ml_user,
   warnings,
   errors,
```

```
 deadlocks,
 synchronized_tables,
 connection_retries)
VALUES (
 {ml s.username},
 {ml s.warnings},
 {ml s.errors},
 {ml s.deadlocks},
 {ml s.synchronized_tables},
 {ml s.connection_retries})' )
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、synchronizationStatisticsConnection という Java メソッドを synchronization_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'synchronization_statistics',
 'ExamplePackage.ExampleClass.synchronizationStatisticsConnection'
)
```

次に示すのは、サンプルの Java メソッド synchronizationStatisticsConnection です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
   String _curUser = null;
public void synchronizationStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int deadlocks,
  int synchronizedTables,
  int connectionRetries ) {
  java.lang.System.out.println(
   "synch statistics number of deadlocks: "
   + deadlocks );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、SyncStats という .NET メソッドを synchronization_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'synchronization_statistics',
 'TestScripts.Test.SyncStats'
)
```

次に示すのは、サンプルの .NET メソッド SyncStats です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts
{
   public class Test
   {
   public void SyncStats( string    user,
                int    warnings,
                int    errors,
                int    deadLocks,
                int    syncedTables,
                int    connRetries )
   {
      System.Console.WriteLine( "synch statistics number of deadlocks: " + deadLocks );
   }
   }
}
```

synchronization_statistics テーブルイベント

同期統計へのアクセスを提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.warnings	INTEGER。同期中にテーブルに対して発生した警告の数。	3

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.errors	INTEGER。同期中に発生したテーブルに関係するエラーの数。	4
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

synchronization_statistics イベントを使用すると、任意のユーザとテーブルについて、同期中に発生した警告とエラーの数を収集できます。最後の同期トランザクション終了時のコミット直前に、synchronization_statistics テーブルスクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_statistics 接続イベント」334 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、同期の統計を sync_tab_audit テーブルに挿入します。

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_insert',
  'INSERT INTO sync_tab_audit (
    ml_user,
    table,
    warnings,
    errors)
```

```
VALUES (
 {ml s.username},
 {ml s.table},
 {ml s.warnings},
 {ml s.errors} ) ' )
```

監査テーブルに同期統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、synchronizationStatisticsTable という Java メソッドを synchronization_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'synchronization_statistics',
 'ExamplePackage.ExampleClass.synchronizationStatisticsTable'
)
```

次に示すのは、サンプルの Java メソッド synchronizationStatisticsTable です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
   String _curUser = null;
public void synchronizationStatisticsTable(
 String user,
 String table,
 int warnings,
 int errors ) {
 java.lang.System.out.println( "synch statistics for table: "
 + table + " errors: " + errors );
 }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、SyncTableStats という .NET メソッドを synchronization_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'synchronization_statistics',
 'TestScripts.Test.SyncTableStats'
)
```

次に示すのは、サンプルの .NET メソッド SyncTableStats です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
 string _curUser = null;
public void SyncTableStats(
```

```
string user,
string table,
int warnings,
int errors ) {
System.Console.WriteLine( "synch statistics for table: "
+ table + " errors: " + errors );
}}}
```

time_statistics 接続イベント

ユーザ別、イベント別の時間統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.event_name	VARCHAR(128)。統計がレポートされるイベント。	2
s.number_of_calls	INTEGER。スクリプトが呼び出された回数。	3
s.minimum_time	INTEGER。ミリ秒。この同期中にスクリプトを実行するのにかかった最短の時間。	4

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.maximum_time	INTEGER。ミリ秒。この同期中にスクリプトを実行するのにかかった最長の時間。	5
s.total_time	INTEGER。ミリ秒。同期ですべてのスクリプトを実行するのにかかった合計時間 (これは同期の長さとは異なります)。	6
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

time_statistics イベントを使用すると、同期の時間統計を収集できます。対応するスクリプトがあるイベントについてのみ、統計が収集されます。単一のイベントが数回発生する場合、スクリプトは集合データを収集します。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「download_statistics 接続イベント」334 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、統計情報を time_statistics テーブルに挿入します。

```
CALL ml_add_connection_script(
 'ver1',
```

```
 'time_statistics',
 'INSERT INTO time_statistics (
  id,
  ml_user,
  event_name,
  number_of_calls,
  minimum_time,
  maximum_time,
  total_time)
 VALUES (
  ts_id.nextval,
  {ml s.username},
  {ml s.event_name},
  {ml s.number_of_calls},
  {ml s.minimum_time},
  {ml s.maximum_time},
  {ml s.total_time} ) ' )
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、timeStatisticsConnection という Java メソッドを time_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'time_statistics',
 'ExamplePackage.ExampleClass.timeStatisticsConnection' )
```

次に示すのは、サンプルの Java メソッド timeStatisticsConnection です。このメソッドは、prepare_for_download イベントの統計を出力します (統計を Mobile Link メッセージログに記録すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass
{
   public void timeStatisticsConnection(
                 String   username,
                 String   eventName,
                 int      numberOfCalls,
                 int      minimumTime,
                 int      maximumTime,
                 int      totalTime )
   {
   if( eventName.equals( "prepare_for_download") ) {
      System.out.println( "prepare_for_download num_calls: " + numberOfCalls +
            " total_time: " + totalTime );
   }
   }
}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、TimeStats という .NET メソッドを time_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'time_statistics',
```

```
  'TestScripts.Test.TimeStats'
)
```

次に示すのは、サンプルの .NET メソッド TimeStats です。このメソッドは、prepare_for_download イベントの統計を出力します (統計を Mobile Link メッセージログに記録すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts
{
   public class test
   {
   public void TimeStats( string    user,
                  string    eventName,
                  int    numberOfCalls,
                  int    minimumTime,
                  int    maximumTime,
                  int    totTime )
   {
      if( eventName == "prepare_for_download" ) {
      System.Console.WriteLine( "prepare_for_download num_calls: " + numberOfCalls +
                  "total_time: " + totTime );
      }
   }
   }
}
```

time_statistics テーブルイベント

時間統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.event_name	VARCHAR(128)。統計がレポートされるイベント。	3
s.number_of_calls	INTEGER。スクリプトが呼び出された回数。	4
s.minimum_time	INTEGER。ミリ秒。このテーブルの同期中にスクリプトを実行するのにかかった最短の時間。	5
s.maximum_time	INTEGER。ミリ秒。このテーブルの同期中にスクリプトを実行するのにかかった最長の時間。	6
s.total_time	INTEGER。ミリ秒。テーブルの同期ですべてのスクリプトを実行するのにかかった合計時間 (これは同期の長さとは異なります)。	7
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

time_statistics テーブルイベントを使用すると、テーブルについて同期中の時間統計を収集できます。対応するスクリプトがあるイベントについてのみ、統計が収集されます。単一のイベントが数回発生する場合、スクリプトは集合データを収集します。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「time_statistics 接続イベント」427 ページ
- 「download_statistics 接続イベント」334 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、統計情報を time_statistics テーブルに挿入します。

```
CALL ml_add_table_script (
 'ver1',
 'table1',
 'time_statistics',
 'INSERT INTO time_statistics(
   ml_user,
   table,
   event_name,
   number_of_calls,
   minimum_time,
   maximum_time,
   total_time)
  VALUES (
   {ml s.username},
   {ml s.table},
   {ml s.event_name},
   {ml s.number_of_calls},
   {ml s.minimum_time},
   {ml s.maximum_time},
   {ml s.total_time} )' );
```

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、timeStatisticsTable という Java メソッドを time_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'time_statistics',
  'ExamplePackage.ExampleClass.timeStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド timeStatisticsTable です。このメソッドは、upload_old_row_insert イベントの統計を出力します。

```
public void timeStatisticsTable(
  String username,
  String tableName,
  String eventName,
  int numberOfCalls,
```

```
  int minimumTime,
  int maximumTime,
  int totalTime ) {
  if( eventName.equals( "upload_old_row_insert") ) {
    java.lang.System.out.println(
     "upload_old_row_insert num_calls: " + numCalls +
     "total_time: " + totalTime );
  }
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、TimeTableStats という .NET メソッドを time_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'time_statistics',
  'TestScripts.Test.TimeTableStats'
)
```

次に示すのは、サンプルの .NET メソッド TimeTableStats です。このメソッドは、upload_old_row_insert イベントの統計を出力します。

```
public void TimeTableStats(
  string user,
  string table,
  string eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totTime ) {
  if( event_name == "upload_old_row_insert") {
    System.Console.WriteLine(
      "upload_old_row_insert num_calls: " + num_calls +
      "total_time: " + total_time );
  }
  }
```

upload_delete テーブルイベント

リモートデータベースから削除されたローを処理するために、Mobile Link サーバがアップロード処理中に使用するイベントを提供するデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.*remote_id*	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	N/A
s.*username*	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	適用外
s.*script_version*	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
r. *pk-column-1*	必須。カラム名またはカラム番号で参照される、最初に削除されたプライマリキーカラム値。	1
...	...	...
r. *pk-column-N*	必須。カラム名またはカラム番号で参照される、最後に削除されたプライマリキーカラム値。	*N*

デフォルトのアクション

なし。

備考

統合データベース側で実行される動作として DELETE 文を指定できますが、他の動作も指定できます。

リモートデータベースのテーブルごとに、upload_delete スクリプトを 1 つ指定できます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

注意
通常、競合検出は、upload_update スクリプトで一度に実行するとより速く実行されます。「upload_update テーブルイベント」455 ページを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「upload_insert テーブルイベント」439 ページ
- 「upload_update テーブルイベント」455 ページ

SQL の例

この例は Contact の例から抜粋したもので、*Samples¥MobiLink¥Contact¥build_consol.sql* にあります。リモートデータベースから削除される顧客に、非アクティブのマークを付けます。

```
CALL ml_add_table_script(
  'ver1',
  'Customer',
  'upload_delete',
  'UPDATE Customer
   SET active = 0
   WHERE cust_id={ml r.cust_id}' )
```

upload_fetch テーブルイベント

ローレベルの競合検出のために、統合データベース内の同期テーブルからローをフェッチするデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.*remote_id*	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.*username*	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	省略可

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.*script_version*	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
r. *primary-key-1*	必須。カラム名またはカラム番号で参照される最初のプライマリキーカラム値。	1 (username を参照する場合は 2)
r. *primary-key-2*	必須。カラム名またはカラム番号で参照される 2 番目のプライマリキーカラム値。	2
...	...	...
r. *primary-key-N*	必須。カラム名またはカラム番号で参照される最後のプライマリキーカラム値。	N (username を参照する場合は N+1)

デフォルトのアクション

なし。

備考

upload_fetch スクリプトは、upload_update イベントに対応します。

結果セットのカラムは、このテーブルについてリモートデータベースからアップロードされるカラムの数や順序と一致します。返される値がアップロードされるローの更新前のイメージと一致しないと、競合が識別されます。

upload_fetch スクリプトでは READPAST テーブルヒントを使用しないでください。スクリプトが READPAST を使用してロックされたローをスキップした場合、同期論理は、そのローが削除されたものとみなします。これにより、定義したスクリプトに応じて、アップロードされた更新が無視されるか、または競合解決がトリガされます。更新の無視は、許容されない動作であることが多く、問題になる場合があります。実装している解決論理によっては、競合解決がトリガされても問題にならない場合があります。

リモートデータベースのテーブルごとに、upload_fetch または upload_fetch_column_conflict スクリプトを 1 つのみ指定できます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

このスクリプトは、次のスクリプトがいずれも定義されていない場合、無視できます。upload_new_row_insert、upload_old_row_insert、および resolve_conflict。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「競合検出」125 ページ
- 「resolve_conflict テーブルイベント」417 ページ
- 「upload_delete テーブルイベント」433 ページ
- 「upload_insert テーブルイベント」439 ページ
- 「upload_update テーブルイベント」455 ページ
- 「FROM 句」『SQL Anywhere サーバ SQL リファレンス』の「Mobile Link 同期で READPAST を使用」

SQL の例

次の SQL スクリプトは、Contact の例から抜粋したもので、*%SQLANYSAMP16%¥MobiLink¥Contact¥build_consol.sql* にあります。リモートデータベースの Product テーブル内で更新されるローのアップロード時に発生する競合を識別するために使用されます。このスクリプトは、テーブル Product からローを選択しますが、統合データベースとリモートデータベースのスキーマによっては、2 つのテーブルの名前が一致しない場合があります。

```
CALL ml_add_table_script(
 'ver1',
 'Product',
 'upload_fetch',
 'SELECT id, name, size, quantity, unit_price
  FROM Product
  WHERE id={ml r.id}' )
```

upload_fetch_column_conflict テーブルイベント

カラムレベルの競合検出のために、統合データベース内の同期テーブルからローをフェッチするデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.*remote_id*	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.*username*	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	省略可
s.*script_version*	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
r. *pk-column-1*	必須。カラム名またはカラム番号で参照される最初のプライマリキーカラム値。	1 (username を参照する場合は 2)
...	...	...
r. *pk-column-N*	必須。カラム名またはカラム番号で参照される最後のプライマリキーカラム値。	*N* (username を参照する場合は *N*+1)

デフォルトのアクション

なし。

備考

upload_fetch_column_conflict スクリプトは、upload_update イベントに対応します。

このスクリプトは、BLOB がないリモートテーブルに対してのみ定義できます。

このスクリプトを使用すると、Mobile Link サーバは、最後の同期以降にリモートデータベースと統合データベースで同じカラムが更新された場合にのみローの競合を検出します。異なるユーザは、同じカラムを更新しないかぎり、同じローを更新することができ、競合は発生しません。

たとえば、upload_fetch_column_conflict スクリプトを使用すると、一方のリモートユーザが ULOrder テーブルの quant カラムを更新し、もう 1 人のリモートユーザが同じローの notes ローを更新した場合に競合が検出されないようにできます。両方のユーザが quant カラムを更新した場合のみ、競合が検出されます。

注意
通常、競合検出は、upload_update スクリプトで一度に実行するとより速く実行されます。「upload_update テーブルイベント」455 ページを参照してください。

upload_fetch_column_conflict スクリプトを使用し、競合が検出されない場合、upload_update スクリプトに渡されるロー値は、upload_fetch_column_conflict スクリプトによってリモートデータベースのアップロードまたは現在の統合値から取得されます。リモートデータベースで更新されたカラムにはリモートデータベースの値が使用されます。それ以外の場合は現在の統合値が使用されます。つまり、リモートデータベースで更新されたカラムのみが統合データベースで更新されます。

リモートデータベースのテーブルごとに、upload_fetch または upload_fetch_column_conflict スクリプトを 1 つのみ指定できます。

このスクリプトは、次のスクリプトがいずれも定義されていない場合、無視できます。upload_new_row_insert、upload_old_row_insert、および resolve_conflict。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「競合検出」125 ページ
- 「upload_fetch テーブルイベント」435 ページ
- 「resolve_conflict テーブルイベント」417 ページ
- 「upload_delete テーブルイベント」433 ページ
- 「upload_insert テーブルイベント」439 ページ
- 「upload_update テーブルイベント」455 ページ

upload_insert テーブルイベント

リモートデータベースに挿入されたローを処理するために、Mobile Link サーバがアップロード処理中に使用するイベントを提供するデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.*remote_id*	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.*username*	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	適用外
s.*script_version*	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
r. *column-1*	必須。カラム名またはカラム値で参照される、最初に挿入されたカラム値。	1
...	...	...
r. *column-N*	必須。カラム名またはカラム値で参照される、最後に挿入されたカラム値。	*N*

デフォルトのアクション

なし。

備考

リモートデータベースのテーブルごとに、upload_insert スクリプトを 1 つ指定できます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「upload_delete テーブルイベント」433 ページ
- 「upload_update テーブルイベント」455 ページ
- 「upload_fetch テーブルイベント」435 ページ

SQL の例

この例では、リモートデータベース内の Customer テーブルに対して行われた挿入を処理します。このスクリプトは、統合データベース内のテーブル Customer に値を挿入します。このテーブルの最後のカラムでは、Customer がアクティブであると識別されます。最後のカラムは、リモートデータベースには含まれません。

```
CALL ml_add_table_script(
 'ver1',
 'Customer',
 'upload_insert',
 'INSERT INTO Customer(
   cust_id,
   name,
   rep_id,
   active )
  VALUES (
   {ml r.cust_id},
   {ml r.name},
   {ml r.rep_id},
   1 )' );
```

upload_new_row_insert テーブルイベント

通常、文ベースのアップロード用の競合解決スクリプトは、リモートデータベースからアップロードされるローの古い値と新しい値にアクセスする必要があります。このデータスクリプトイベントを使用すると、リモートデータベースからアップロードされるローの更新済みの新しい値を処理できます。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	オプション (参照される場合は 1)

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
r. *column-1*	必須。カラム名またはカラム番号で参照される、新しい (更新後イメージ) ローの最初のカラム値。	1 (username を参照する場合は 2)
...	...	...
r. *column-N*	必須。カラム名またはカラム番号で参照される、新しい (更新後イメージ) ローの最後のカラム値。	*N* (username を参照する場合は *N*+1)

デフォルトのアクション

なし

備考

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

Mobile Link による競合の検出後、このイベントを使用すると、更新後イメージの値をテーブルに保存できます。このイベントは、更新用の競合解決プロシージャ開発を支援するために使用できます。このイベントのパラメータは、対応する統合データベーステーブルで更新が実行される前のリモートデータベースからの新しい値を保持しています。また、強制的な競合モードで、ローを挿入するために使用されます (強制的な競合モードは推奨されなくなりました)。

注意
通常、競合検出は、upload_update スクリプトで一度に実行するとより速く実行されます。
「upload_update テーブルイベント」455 ページを参照してください。

このイベントのスクリプトは、resolve_conflict スクリプトが使用するテンポラリテーブルに新しいローを挿入する INSERT 文である場合がほとんどです。

リモートデータベースのテーブルごとに、upload_new_row_insert スクリプトを 1 つ指定できます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「競合の解決」124 ページ
- 「resolve_conflict テーブルイベント」417 ページ
- 「upload_old_row_insert テーブルイベント」443 ページ
- 「upload_update テーブルイベント」455 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

この例は、リモートデータベース内の product テーブルに対する更新を処理します。このスクリプトは、ローの新しい値をグローバルなテンポラリテーブル product_conflict に挿入します。このテーブルの最後のカラムでは、ローが新しいローとして識別されます。

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'upload_new_row_insert',
 'INSERT INTO DBA.product_conflict(
   id,
   name,
   size,
   quantity,
   unit_price,
   row_type )
  VALUES(
   {ml r.id},
   {ml r.name},
   {ml r.size},
   {ml r.quantity},
   {ml r.unit_price},
   "New" )' )
```

upload_old_row_insert テーブルイベント

通常、文ベースのアップロード用の競合解決スクリプトは、リモートデータベースからアップロードされるローの古い値と新しい値にアクセスする必要があります。このデータスクリプトイベントを使用すると、リモートデータベースからアップロードされるローの古い値を処理できます。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名

前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	N/A
s.username	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	オプション (参照される場合は 1)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
r. *column-1*	必須。カラム名またはカラム番号で参照される、古い (更新前イメージ) ローの最初のカラム値。	1 (username を参照する場合は 2)
...	...	...
r. *column-N*	必須。カラム名またはカラム番号で参照される、古い (更新前イメージ) ローの最後のカラム値。	*N* (username を参照する場合は *N*+1)

デフォルトのアクション

なし。

備考

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

Mobile Link による競合の検出後、このイベントを使用すると、更新前イメージの値をテーブルに保存できます。このイベントは、競合解決プロシージャ開発を支援するために使用できます。このイベントのパラメータは、対応する統合データベーステーブルで更新が実行される前のリ

モートデータベースからの古い値を保持しています。また、強制的な競合モードで、ローを挿入するために使用されます (強制的な競合モードは推奨されなくなりました)。

> **注意**
> 通常、競合検出は、upload_update スクリプトで一度に実行するとより速くなります。
> 「upload_update テーブルイベント」455 ページを参照してください。

このイベントのスクリプトは、resolve_conflict スクリプトが使用するテンポラリテーブルに古いローを挿入する INSERT 文である場合がほとんどです。

リモートデータベースのテーブルごとに、upload_old_row_insert スクリプトを 1 つ指定できます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「競合の解決」124 ページ
- 「resolve_conflict テーブルイベント」417 ページ
- 「upload_new_row_insert テーブルイベント」441 ページ
- 「upload_update テーブルイベント」455 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

この例は、リモートデータベース内の product テーブルに対する更新を処理します。このスクリプトは、ローの古い値をグローバルなテンポラリテーブル product_conflict に挿入します。このテーブルの最後のカラムでは、ローが古いローとして識別されます。

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'upload_old_row_insert',
 'INSERT INTO DBA.product_conflict (
   id,
   name,
   size,
   quantity,
   unit_price,
   row_type )
  VALUES (
   {ml r.id},
   {ml r.name},
   {ml r.size},
   {ml r.quantity},
   {ml r.unit_price},
   "Old" )' )
```

upload_statistics 接続イベント

アップロード操作の同期統計へのアクセスを提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.warnings	INTEGER。発生した警告の数。	2
s.errors	INTEGER。発生したエラーの数。	3
s.inserted_rows	INTEGER。統合データベースに正常に挿入されたローの数。	4
s.deleted_rows	INTEGER。統合データベースから正常に削除されたローの数。	5
s.updated_rows	INTEGER。統合データベースで正常に更新されたローの数。	6

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.conflicted_updates	INTEGER。競合を引き起こした更新ローの数。対処する競合解決スクリプトが正常に呼び出された場合のみ、ローは含まれます。	9
s.ignored_inserts	INTEGER。無視されたアップロード挿入ローの合計数。無視される理由は次のとおりです。1) 通常モードで upload_insert スクリプトがない。2) Mobile Link サーバが対応するスクリプトを呼び出し中にエラーが発生し、handle_error または handle_odbc_error イベントが 1000 を返した。	10
s.ignored_deletes	INTEGER。upload_delete スクリプトを呼び出したとき、handle_error または handle_odbc_error が定義され、1000 を返した場合、または指定のテーブルに対して upload_delete スクリプトが定義されていない場合にエラーを発生したアップロード削除ローの数。	11
s.ignored_updates	INTEGER。競合を引き起こしたが、競合解決スクリプトが正常に呼び出されなかったり、upload_update スクリプトが定義されていなかったアップロード更新ローの数。	12
s.bytes	INTEGER。アップロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。	13
s.deadlocks	INTEGER。同期で検出された統合データベース内のデッドロックの数。	14

デフォルトのアクション

なし。

備考

upload_statistics イベントを使用すると、任意のユーザについてアップロードに関する統計を収集できます。アップロードトランザクション終了時のコミット直前に、upload_statistics 接続スクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_statistics 接続イベント」334 ページ
- 「download_statistics テーブルイベント」337 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、アップロード操作での同期の統計を upload_summary_audit テーブルに挿入します。

```
CALL ml_add_connection_script (
 'ver1',
 'upload_statistics',
 'INSERT INTO upload_summary_audit (
   ml_user,
   warnings,
   errors,
   inserted_rows,
   deleted_rows,
   updated_rows,
   conflicted_updates,
   ignored_inserts,
   ignored deletes,
   ignored_updates,
   bytes, deadlocks )
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
   {ml s.ignored_updates},
   {ml s.bytes},
   {ml s.deadlocks} ) ' )
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、uploadStatisticsConnection という Java メソッドを upload_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'upload_statistics',
  'ExamplePackage.ExampleClass.uploadStatisticsConnection' )
```

次に示すのは、サンプルの Java メソッド uploadStatisticsConnection です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void uploadStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  java.lang.System.out.println( "updated rows: " +
   updatedRows );
  }}
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、UploadStats という .NET メソッドを upload_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'upload_statistics',
  'TestScripts.Test.UploadStats'
)
```

次に示すのは、サンプルの .NET メソッド UploadStats です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void UploadStats (
  string user,
  int warnings,
  int errors,
```

```
int insertedRows,
int deletedRows,
int updatedRows,
int conflictInserts,
int conflictDeletes,
int conflictUpdates,
int ignoredInserts,
int ignoredDeletes,
int ignoredUpdates,
int bytes,
int deadlocks ) {
System.Console.WriteLine( "updated rows: " +
  updatedRows );
}}}
```

upload_statistics テーブルイベント

特定のテーブルに対するアップロード操作の同期統計へのアクセスを提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切な対応するデータ型を使用します。 「SQL データ型と Java データ型」462 ページと「SQL データ型と .NET データ型」537 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用外
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.table	VARCHAR(128)。テーブル名。	2
s.warnings	INTEGER。テーブルのアップロードで発行された警告の数。	3
s.errors	INTEGER。処理済みのエラーを含め、テーブルのアップロードで発生したエラーの数。	4
s.inserted_rows	INTEGER。統合データベースに正常に挿入されたローの数。	5
s.deleted_rows	INTEGER。統合データベースから正常に削除されたローの数。	6
s.updated_rows	INTEGER。統合データベースで正常に更新されたローの数。	7
s.conflicted_updates	INTEGER。競合を引き起こした更新ローの数。対処する競合解決スクリプトが正常に呼び出された場合のみ、ローは含まれます。	10
s.ignored_inserts	INTEGER。無視されたアップロード挿入ローの合計数。無視される理由は次のとおりです。1) 通常モードで upload_insert スクリプトがない。2) Mobile Link サーバが対応するスクリプトを呼び出し中にエラーが発生し、handle_error または handle_odbc_error イベントが 1000 を返した。	11
s.ignored_deletes	INTEGER。upload_delete スクリプトを呼び出したとき、handle_error または handle_odbc_error が定義され、1000 を返した場合、または指定のテーブルに対して upload_delete スクリプトが定義されていない場合にエラーを発生したアップロード削除ローの数。	12

SQL スクリプトのパラメータ名	説明	順序 (SQL では非推奨)
s.ignored_updates	INTEGER。競合を引き起こしたが、競合解決スクリプトが正常に呼び出されなかったり、upload_update スクリプトが定義されていなかったアップロード更新ローの数。	13
s.bytes	INTEGER。アップロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。	14
s.deadlocks	INTEGER。同期で検出された統合データベース内のデッドロックの数。	15

デフォルトのアクション

なし。

備考

upload_statistics イベントを使用すると、任意のユーザについて、任意のテーブルに適用される同期発生の重要な統計を収集できます。アップロードトランザクション終了時のコミット直前に、upload_statistics テーブルスクリプトが呼び出されます。

注意
コマンドラインによっては、一部の警告のログが取られないことがあります。このスクリプトに渡される警告数は、警告が無効になっていない場合にログに取られる警告数であり、ログに取られる警告数より多くなることがあります。

参照

- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「download_statistics 接続イベント」334 ページ
- 「upload_statistics 接続イベント」445 ページ
- 「upload_statistics テーブルイベント」450 ページ
- 「synchronization_statistics 接続イベント」421 ページ
- 「synchronization_statistics テーブルイベント」424 ページ
- 「time_statistics 接続イベント」427 ページ
- 「time_statistics テーブルイベント」430 ページ
- 「Mobile Link プロファイラ」191 ページ
- 「スクリプトでのリモート ID と Mobile Link ユーザ名」『Mobile Link クライアント管理』

SQL の例

次の例は、アップロードの統計を追跡するために使用されるテーブルにローを挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'upload_statistics',
 'INSERT INTO my_upload_statistics (
   user_name,
   table_name,
   num_warnings,
   num_errors,
   inserted_rows,
   deleted_rows,
   updated_rows,
   conflicted_updates,
   ignored_inserts,
   ignored_deletes,
   ignored_updates, bytes,
   deadlocks )
  VALUES(
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
   {ml s.ignored_updates},
   {ml s.bytes},
   {ml s.deadlocks} )'  )
```

次の例は、Oracle 統合データベースで動作します。

```
CALL ml_add_connection_script(
 'ver1',
 'upload_statistics',
 'INSERT INTO upload_tables_audit (
  id,
  user_name,
  table,
  warnings,
  errors,
  inserted_rows,
  deleted_rows,
  updated_rows,
  conflicted_updates,
  ignored_inserts,
  ignored_deletes,
  ignored_updates,
  bytes,
  deadlocks )
 VALUES (
   ut_audit.nextval,
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
```

```
{ml s.ignored_updates},
{ml s.bytes},
{ml s.deadlocks} )' )
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 を同期するときに、uploadStatisticsTable という Java メソッドを upload_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'upload_statistics',
 'ExamplePackage.ExampleClass.uploadStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド uploadStatisticsTable です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します 統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります。

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
public void uploadStatisticsTable(
  String user,
  String table,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  java.lang.System.out.println( "updated rows: " +
    updatedRows );
  }
```

.NET の例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン ver1 とテーブル table1 を同期するときに、UploadTableStats という .NET メソッドを upload_statistics テーブルイベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'upload_statistics',
 'TestScripts.Test.UploadTableStats'
)
```

次に示すのは、サンプルの .NET メソッド uploadStatisticsTable です。このメソッドは、Mobile Link メッセージログに統計の一部を出力します (統計を Mobile Link メッセージログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
namespace TestScripts {
public class Test {
  string _curUser = null;
public void UploadTableStats(
  string user,
  string table,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictInserts,
  int conflictDeletes,
  int conflictUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  System.Console.WriteLine( "updated rows: " +
    updatedRows );
  }}}
```

upload_update テーブルイベント

リモートデータベースで更新されるローを処理するために、Mobile Link サーバがアップロード処理中に使用するイベントを提供するデータスクリプトです。

パラメータ

SQL スクリプトでは、名前または疑問符を使用してイベントパラメータを指定できます。疑問符の使用は推奨されなくなりました。名前付きパラメータを使用することをおすすめします。1 つのスクリプト内で名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

パラメータ	説明	順序 (SQL では非推奨)
s. *script_version*	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

パラメータ	説明	順序 (SQL では非推奨)
r. *column-1*	必須。カラム名またはカラム番号で参照される、新しい (更新後イメージ) カラム値の最初の非プライマリキーカラム値。	1
...	...	...
r. *column-M*	必須。カラム名またはカラム番号で参照される、新しい (更新後イメージ) カラム値の最後の非プライマリキーカラム値。	M
r. *pk-column-1*	必須。カラム名またはカラム番号で参照される、新しい (更新後イメージ) カラム値の最初のプライマリキーカラム値。	M + 1
...	...	...
r. *pk-column-N*	必須。カラム名またはカラム番号で参照される、新しい (更新後イメージ) カラム値の最後のプライマリキーカラム値。	M + N
o. *column-N*	省略可。カラム名またはカラム番号で参照される、古い (更新前イメージ) カラム値の最初の非プライマリキーカラム値。	M + N + 1
...	...	...
o. *column-M*	省略可。カラム名またはカラム番号で参照される、古い (更新前イメージ) カラム値の最後の非プライマリキーカラム値。	M + N + M

パラメータ	説明	順序 (SQL では非推奨)
s.script_version	VARCHAR(128)。Mobile Link サーバが現在の同期に使用しているスクリプトバージョン文字列をこのパラメータに渡すことを指定する、オプションの IN パラメータ。このパラメータの指定に疑問符を使用することはできません。	適用外

デフォルトのアクション

なし。

備考

WHERE 句には、同期するプライマリキーカラムをすべて含めます。必要に応じて非プライマリキーカラムを含めることもできます。SET 句には、同期する非プライマリキーカラムをすべて含めます。

名前付きパラメータは任意の順序で使用できます。同じ名前付きパラメータを同じスクリプトで必要な回数だけ使用できます。名前付きパラメータが指定されたスクリプトでは、カラムのサブセットのみを指定できます。

たとえば、MyTable テーブルの upload_update スクリプトは、次のようの記述できます。

```
UPDATE MyTable
  SET column_2 = { ml r.column_2 }, column_1 = { ml r.column_1 }, ..., column_M = { ml r.column_M }
  WHERE pk_column_1 = { ml r.pk_column_1 } AND ... AND pk_column_N = { ml r.pk_column_N }
```

リモートデータベースのテーブルごとに、upload_update スクリプトを 1 つ指定できます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

upload_update スクリプトを使用して競合を検出するには、以下のようにすべての非プライマリキーカラムを WHERE 句に含めます。

UPDATE *table-name*
SET *col1* **= {ml r.***col1***},** *col2* **= {ml r.** *col2* **}** …
WHERE *pk1* **= {ml r.***pk1***} AND** *pk2* **= {ml r.***pk2***}** …
  AND *col1* **= {ml o.***col1***} AND** *col2* **={ml o.***col2***}** …

この文では、col1 と col2 はプライマリキーカラムではありませんが、pk1 と pk2 はプライマリキーカラムです。非プライマリキーカラムの 2 番目のセットに渡される値は、更新ローの更新前イメージです。WHERE 句は、リモートデータベースから更新された古い値と、統合データベースの現在の値を比較します。これらの値が一致しないと更新は無視されるので、すでに統合データベースにあった値は保持されます。

このスクリプトは SQL で実装してください。 Java または .NET のローの処理については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- 「データスクリプト」281 ページ
- 「スクリプトのパラメータ」234 ページ
- 「スクリプトの追加と削除」256 ページ
- 「upload_update スクリプトによる競合の検出」128 ページ
- 「upload_update スクリプトによる競合の解決」130 ページ
- 「upload_delete テーブルイベント」433 ページ
- 「upload_fetch テーブルイベント」435 ページ
- 「upload_insert テーブルイベント」439 ページ

SQL の例

この例は、リモートデータベース内の Customer テーブルに対する更新を処理します。このスクリプトは、統合データベース内のテーブル Customer 内の値を更新します。

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_update',
'UPDATE Customer
   SET name = {ml r.name}, rep_id = {ml r.rep_id}
      WHERE cust_id = {ml o.cust_id}')
```

次の例は、同様の更新を実行しますが、古い (更新前イメージの) 値を使用して、競合がない場合にのみ更新が行われるようにします。競合がある場合、更新はこの「先入れ勝ち」競合解決ポリシーで無視されます。

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_update',
'UPDATE Customer
   SET name = {ml r.name}, rep_id = {ml r.rep_id}
      WHERE cust_id = {ml o.cust_id}
        AND name = {ml o.name}
        AND rep_id = {ml o.rep_id}')
```

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

Mobile Link サーバ API

この項では、Java と .NET 用の Mobile Link サーバ API について説明します。

Java による同期スクリプトの作成

Mobile Link サーバの動作を制御するには、同期スクリプトを作成します。これらのスクリプトの実装には、SQL、.NET または Java を使用できます。Java の同期論理は、SQL 論理と同様に機能します。Mobile Link サーバは、Mobile Link イベントの発生時に SQL スクリプトにアクセスするのと同様に、Java メソッドを呼び出すことができます。Java メソッドは、SQL 文字列を Mobile Link に返すことができます。

この項では、Java 同期論理を設定、開発、実行する方法について説明します。また、サンプルアプリケーションと Java 用 Mobile Link サーバ API リファレンスも含まれています。

参照

- 「サーバ側の同期論理の作成オプション」『Mobile Link クイックスタート』
- 「同期スクリプトの作成」229 ページ

Java 同期論理の設定

次の手順を使用して、Java に同期スクリプトを実装します。

前提条件

この作業を実行するための前提条件はありません。

内容と備考

SQL Anywhere をインストールすると、インストーラによって Java 用 Mobile Link サーバ API クラスのロケーションが自動的に設定されます。これらのクラスは、Mobile Link サーバの起動時にクラスパスに自動的に組み込まれます。Java 用 Mobile Link サーバ API クラスは、*%SQLANY16%¥Java¥mlscript.jar* にあります。

♦ タスク

1. 1 つまたは複数の独自クラスを作成します。必要な同期スクリプトごとに、メソッドを作成します。これらのメソッドは、パブリックにしてください。クラスは、パッケージ内ではパブリックにしてください。

   「メソッド」464 ページを参照してください。

   非静的メソッドを持つ各クラスには、パブリックコンストラクタが必要です。Mobile Link サーバは、各クラスのメソッドが初めて呼び出されるときに、そのクラスを自動的にインスタンス化します。

   「コンストラクタ」463 ページを参照してください。

2. クラスをコンパイルするときは、JAR ファイル *java¥mlscript.jar* をインクルードしてください。

   次に例を示します。

   ```
   javac MyClass.java -classpath "C:¥Program Files¥SQL Anywhere 16¥java¥mlscript.jar"
   ```

3. 統合データベースの Mobile Link システムテーブルで、各同期スクリプトについて、呼び出すパッケージ、クラス、メソッドの名前を指定します。スクリプトのバージョンごとに、クラスを 1 つずつ使用します。

   たとえば、ml_add_java_connection_script ストアドプロシージャまたは ml_add_java_table_script ストアドプロシージャを使用して、この情報を Mobile Link システムテーブルに追加できます。

   たとえば、次の SQL 文は、SQL Anywhere データベース内で実行すると、スクリプトバージョン ver1 に対して、authenticate_user 接続レベルイベントが発生するたびに myPackage.myClass.myMethod を実行するように指定します。指定されるメソッドはパブリック Java メソッドの完全修飾名で、大文字と小文字が区別されます。

   ```
   call ml_add_java_connection_script('ver1',
   'authenticate_user', 'myPackage.myClass.myMethod')
   ```

   スクリプト追加の詳細については、以下の項目を参照してください。

   - 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
   - 「ml_add_java_connection_script システムプロシージャ」651 ページ
   - 「ml_add_java_table_script システムプロシージャ」652 ページ

4. Mobile Link サーバがクラスをロードするよう指定します。Java 同期ロジックの設定の中で最も重要な部分は、Java クラスの検索場所を Java VM に対して指定することです。このようにするには、次の 2 つの方法があります。

   - クラスを検索するディレクトリまたは jar ファイルを指定するには、mlsrv16 -sl java -cp オプションを使用します。たとえば、次のコマンドを実行します。

     ```
     mlsrv16 -c "DSN=consolidated1" -sl java (-cp %classpath%;c:¥local¥Java¥myclasses.jar)
     ```

     Mobile Link サーバは、一連のディレクトリまたは jar ファイルに Java 用 Mobile Link サーバ API クラスのロケーション (java¥mlscript.jar) を自動的に追加します。また、-sl java オプションは、Java VM がサーバの起動時にロードされるよう指定します。

     使用可能な Java オプションの詳細については、「-sl java mlsrv16 オプション」78 ページを参照してください。

   - 明示的にクラスパスを設定します。ユーザ定義クラスのクラスパスを設定するには、次のような文を使用します。

     ```
     SET classpath=%classpath%;c:¥local¥Java¥myclasses.jar
     ```

     システムのクラスパスに Java 同期論理のクラスが含まれている場合、Mobile Link サーバのコマンドラインを変更する必要はありません。

-sl java オプションを使用すると、サーバ起動時に Java VM を強制的にロードできます。このオプションを使用しない場合は、Java メソッドが最初に実行されるときに Java VM が起動します。

使用可能な Java オプションの詳細については、「-sl java mlsrv16 オプション」78 ページを参照してください。

5. UNIX で特定の JRE をロードする場合は、LD_LIBRARY_PATH (IBM AIX の場合は LIBPATH、HP-UX の場合は SHLIB_PATH) を設定して、JRE を含むディレクトリを指定します。ディレクトリは、すべての SQL Anywhere インストールディレクトリの前に指定します。

結果

Java 同期論理が設定されます。

参照

- 「Java 同期論理」461 ページ
- 「Java 同期の例」468 ページ
- 「Mobile Link サーバ Java API リファレンス」473 ページ
- 「サーバ側の同期論理の作成オプション」『Mobile Link クイックスタート』
- 「同期スクリプトの作成」229 ページ

Java 同期論理

Java 同期論理を作成するには、Mobile Link イベントの知識、Java に関する若干の知識、Java 用 Mobile Link サーバ API の知識が必要です。

API の完全な説明については、「Mobile Link サーバ Java API リファレンス」473 ページを参照してください。

Java 同期論理を使用すると、ステータス情報を管理し、アップロードイベントとダウンロードイベント関連の論理を実装できます。たとえば、Java で作成された begin_synchronization スクリプトを使用すると、Mobile Link ユーザ名を変数に格納できます。同期処理中に後で呼び出されるスクリプトは、この変数にアクセスできます。また、Java は、コミットの実行前または実行後に統合データベースのローにアクセスするために使用できます。

Java を使用すると、統合データベースへの依存度が減少します。また、統合データベースを新バージョンにアップグレードしたり、別のデータベース管理システムに切り替えたりする場合も、動作に与える影響が少なくて済みます。

ダイレクトローハンドリング

Mobile Link のダイレクトローハンドリングを使用して、リモートデータと中央のデータソース、アプリケーション、または Web サービスとの通信ができます。ダイレクトローハンドリングでは、Java または .NET 用 Mobile Link サーバ API の特別なクラスを使用して、同期対象のデータに直接アクセスします。

「ダイレクトローハンドリング」623 ページを参照してください。

クラスインスタンス

Mobile Link サーバは、クラスを接続レベルでインスタンス化します。非静的 Java メソッドをあるイベントに対して作成した場合、そのイベントに達すると、現在の接続でクラスが作成されていなければ、Mobile Link サーバが自動的にクラスのインスタンスを作成します。

「コンストラクタ」537 ページを参照してください。

1 つのスクリプトバージョンの接続レベルイベントまたはテーブルレベルイベントに直接関連するすべてのメソッドは、「同じクラスに属している必要があります」。

データベース接続ごとに、インスタンス化されたクラスはその接続が終了するまで持続します。したがって、連続する複数の同期セッションで、同じインスタンスを使用することができます。明示的にクリアされないかぎり、パブリック変数またはプライベート変数内の情報は、同じ接続で発生するすべての同期を通して持続されます。

また、静的なクラスや変数も使用できます。この場合、値はすべての接続を通して使用できます。

統合データベースへの接続が終了した場合にのみ、Mobile Link サーバはクラスインスタンスを自動的に削除します。

トランザクション

Java メソッドには、トランザクションに関する通常のルールが適用されます。データベーストランザクションの開始と継続期間は、同期処理に重要になります。トランザクションの開始と終了は、Mobile Link サーバのみが行います。Java メソッド内の同期接続でトランザクションを明示的にコミットまたはロールバックすると、同期処理の整合性違反になり、エラーが発生することがあります。

これらのルールは、Mobile Link サーバによって作成されるデータベース接続、特に、メソッドから返される SQL 文にのみ適用されます。クラスによって他のデータベース接続が作成される場合は、既存の管理ルールを使用して、他のデータベース接続によって作成されたクラスを管理してください。

SQL データ型と Java データ型

次の表は、SQL データ型とそれに対応する Java データ型を示します。

SQL データ型	対応する Java データ型
VARCHAR	java.lang.String
CHAR	java.lang.String
INTEGER	int または Integer
BINARY	byte[ ]

SQL データ型	対応する Java データ型
BIGINT	long
TIMESTAMP	java.sql.Timestamp
INOUT INTEGER	ianywhere.ml.script.InOutInteger
INOUT VARCHAR	ianywhere.ml.script.InOutString
INOUT CHAR	ianywhere.ml.script.InOutString
INOUT BYTEARRAY	ianywhere.ml.script.InOutByteArray
INOUT TIMESTAMP	java.sql.Timestamp

ianywhere.ml.script パッケージが存在しない場合は、Mobile Link サーバがクラスパスに自動的に追加します。ただし、クラスをコンパイルするときは、*%SQLANY16%¥java¥mlscript.jar* のパスを追加する必要があります。

コンストラクタ

クラスのコンストラクタは、次の 2 つのシグネチャのどちらかを持ちます。

```
public MyScriptClass(ianywhere.ml.script.DBConnectionContext sc)
```

または

```
public MyScriptClass()
```

渡される同期コンテキストは、Mobile Link サーバが現在のユーザの同期に使用している接続です。

DBConnectionContext.getConnection メソッドは、Mobile Link が現在のユーザの同期に使用しているのと同じデータベース接続を返します。この接続で文を実行することはできますが、トランザクションのコミットやロールバックは行わないでください。トランザクションは Mobile Link サーバによって管理されます。

Mobile Link サーバは、最初のシグネチャを持つコンストラクタを使用しようとします。引数のないコンストラクタが使用されるのは、最初のシグネチャを持つコンストラクタが存在しない場合のみです。

参照

- DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページ

メソッド

通常は、同期イベントごとにメソッドを 1 つずつ実装します。これらのメソッドは、パブリックにしてください。プライベートメソッドの場合、Mobile Link サーバでは使用できず、その存在を認識できません。

統合データベース内の ml_script テーブルで指定されている名前と一致していれば、メソッド名は重要ではありません。ただし、このマニュアルの例では、メソッド名は Mobile Link イベント名と同じです。これは、Java コードを読みやすくするためです。

メソッドのシグネチャは、そのイベント用スクリプトのシグネチャと一致していなければなりません。ただし、パラメータリストの最後にパラメータ値が必要でない場合は、リストをトランケートできます。パラメータを渡すとオーバヘッドが生じる可能性があるため、必要なパラメータのみを受け入れてください。

ただし、メソッドはオーバロードできません。ml_script システムテーブルには、クラスごとにメソッドプロトタイプが 1 つだけ格納されます。

メソッドの登録

メソッドを作成したら、それを登録します。メソッドを登録すると、統合データベースの Mobile Link システムテーブル内にメソッドへの参照が作成されて、イベントが発生すると、そのメソッドが呼び出されます。メソッドの登録方法は、同期スクリプトの追加方法と同じです。ただし、SQL スクリプト全体を Mobile Link システムテーブルに追加する代わりに、メソッド名のみを追加します。

「スクリプトの追加と削除」256 ページを参照してください。

戻り値

SQL ベースのアップロードまたはダウンロードに対して呼び出されるメソッドは、有効な SQL 言語の文を返さなければなりません。これらのメソッドの戻り値の型は、java.lang.String にしてください。他の戻り値の型は使用できません。

他のすべてのスクリプトの戻り値の型は、java.lang.String または void にします。他の型は使用できません。戻り値の型が NULL ではなく文字列の場合、Mobile Link サーバはその文字列に有効な SQL 文が含まれていると見なして、この文を通常の SQL 言語による同期スクリプトと同様に統合データベース内で実行します。通常、メソッドは文字列を返しますが、その戻り値によってデータベースに対して SQL 文を実行しない場合は null を返すことができます。

Java クラスのデバッグ

Mobile Link は、Java コードのデバッグ時に役立つ各種の情報と機能を提供します。この項では、これらの情報の格納場所と機能の活用方法について説明します。

Mobile Link サーバのログファイル内の情報

Mobile Link サーバは、メッセージをメッセージログファイルに書き込みます。サーバのログファイルには、次の情報が書き込まれます。

- Java Runtime Environment-jrepath オプションを使用すると、Mobile Link サーバの起動時に、特定の JRE を要求できます。デフォルトパスは、SQL Anywhere 16 でインストールされた JRE のパスです。
- ロードされている標準 Java クラスのパス。これらのパスを明示的に指定しない場合、Mobile Link サーバはクラスパスに自動的に追加してから、Java VM を起動します。
- 呼び出された特定のメソッドの完全指定名。この情報を使用すると、Mobile Link サーバが適切なメソッドを呼び出していることを確認できます。
- Java メソッドで java.lang.System.out または java.lang.System.err に書き込まれた出力。すべて Mobile Link サーバのログファイルにリダイレクトされます。
- mlsrv16 コマンドラインのオプション -verbose を使用できます。

  「-v mlsrv16 オプション」85 ページを参照してください。

Java デバッガの使用

標準 Java デバッガを使用して、Java クラスをデバッグできます。mlsrv16 コマンドラインで -sl java オプションを使用して、必要なパラメータを指定します。

「-sl java mlsrv16 オプション」78 ページを参照してください。

デバッガを指定すると、Java VM は一時停止し、Java デバッガからの接続を待機します。

Java からの情報の出力

もう 1 つの方法として、java.lang.System.err か java.lang.System.out を使用して、Mobile Link メッセージログに情報を出力する文を Java メソッドに追加することもできます。これにより、クラスの進行状況と動作を追跡できます。

> **注意**
> この方法で情報を出力すると、モニタツールとして活用できますが、運用環境では推奨いたしません。

これと同じ方法を利用して、任意の同期情報のログを取ったり、スクリプトの使用方法に関する統計情報を収集したりできます。

独自のテストドライバの作成

独自のドライバを作成して、Java クラスをテストできます。このアプローチでは、Java メソッドのアクションが Mobile Link システムの残りの部分から分離されるため便利な場合があります。

Java での Mobile Link サーバエラーの処理

ログをスキャンするだけでは不十分な場合は、プログラムによってアプリケーションをモニタリングできます。たとえば、特定のタイプのメッセージを電子メールで送信できます。

ログに出力される各エラーメッセージまたは各警告メッセージを表すクラスに渡されるメソッドを作成することも可能です。この方法は、Mobile Link サーバをモニタおよび監査するのに役立ちます。

次のコードは、すべての警告メッセージ用に LogListener をインストールし、その情報をファイルに書き込みます。

```
class TestLogListener implements LogListener {
   FileOutputStream _out_file;

   public TestLogListener(FileOutputStream out_file) {
      _out_file = out_file;
   }

   public void messageLogged(ServerContext sc, LogMessage msg) {
      String  type;
      String  user;

      try {
         if (msg.getType() == LogMessage.ERROR) {
            type = "ERROR";
         } else if (msg.getType() == LogMessage.WARNING) {
            type = "WARNING";
         }
         else {
            type = "UNKNOWN!!!";
         }

         user = msg.getUser();
         if (user == null) {
            user = "NULL";
         }
         _out_file.write(("Caught msg type="
            + type
            + " user=" + user
            + " text=" + msg.getText()
            + "¥n").getBytes()
         );
         _out_file.flush();
      }
      catch(Exception e) {
         // Print some error output to the MobiLink log.
         e.printStackTrace();
      }
   }
}
```

次のコードは TestLogListener を登録して、警告メッセージを受信します。クラスコンストラクタや同期スクリプトなど、ServerContext にアクセスできる任意の場所からこのコードを呼び出してください。

```
// ServerContext serv_context;
serv_context.addWarningListener(
      new MyLogListener(ll_out_file)
);
```

参照

- ServerContext.addErrorListener メソッド [Mobile Link サーバ Java]503 ページ
- ServerContext.removeErrorListener メソッド [Mobile Link サーバ Java]508 ページ
- ServerContext.addWarningListener メソッド [Mobile Link サーバ Java]505 ページ
- ServerContext.removeWarningListener メソッド [Mobile Link サーバ Java]509 ページ
- LogListener インタフェース [Mobile Link サーバ Java]491 ページ
- LogMessage クラス [Mobile Link サーバ Java]492 ページ

ユーザ定義起動クラス

サーバの起動時に自動的にロードされる起動クラスを定義できます。この機能の目的は、最初の同期の前に Mobile Link サーバが Java VM を起動する時点で実行される Java コードを記述できるようにすることです。つまり、ユーザ同期要求の前に、接続の作成またはデータのキャッシュを実行できます。

この操作を行うには、mlsrv16 -sl java オプションの DMLStartClasses オプションを使用します。たとえば、次に示すのは mlsrv16 コマンドラインの一部です。mycl1 と mycl2 が起動クラスとしてロードされます。

```
-sl java(-DMLStartClasses=com.test.mycl1,com.test.mycl2)
```

クラスはリスト内の順序でロードされます。同じクラスがリストに 2 回以上指定されている場合は、複数のインスタンスが作成されます。

起動クラスはすべてパブリックにしてください。また、引数を受け付けないか、ianywhere.ml.script.ServerContext 型の引数を 1 つ受け入れるパブリックコンストラクタが必要です。

ロードされた起動クラスの名前は、「Java 起動クラス *classname* がロードされました。」というメッセージとともに Mobile Link ログに出力されます。

Java VM オプションの詳細については、「-sl java mlsrv16 オプション」78 ページを参照してください。

サーバ起動時に構築される起動クラスを表示する方法については、ServerContext.getStartClassInstances メソッド [Mobile Link サーバ Java]507 ページを参照してください。

例

次に示すのは、起動クラスのテンプレートです。これは、イベントを処理してデータベース接続を確立するデーモンスレッドを起動します (すべての起動クラスがスレッドの作成を必要とするわけではありませんが、スレッドを作成する場合はデーモンスレッドでなければなりません)。

```
import ianywhere.ml.script.*;
import java.sql.*;

public class StartTemplate extends
   Thread implements ShutdownListener {
   ServerContext   _sc;
   Connection      _conn;
   boolean         _exit_loop;

   public StartTemplate(ServerContext sc) throws SQLException {

      // Perform setup first so that an exception
      // causes MobiLink startup to fail.
      _sc = sc;

      // Create a connection for use later.
      _conn = _sc.makeConnection();

      _exit_loop = false;
      setDaemon(true);
```

```
        start();
    }

    public void run() {
        _sc.addShutdownListener(this);

        // run() cannot throw exceptions.
        try {
            handlerLoop();
            _conn.close();
            _conn = null;
        }
        catch(Exception e) {

            // Print some error output to the MobiLink log.
            e.printStackTrace();

            // This thread shuts down and so does not
            // need to be notified of shutdown.
            _sc.removeShutdownListener(this);

            // Ask server to shutdown so that this fatal
            // error is fixed.
            _sc.shutdown();
        }
        // Shortly after return this thread no longer exists.
        return;
    }

    // stop our event handler loop
    public void shutdownPerformed(ServerContext sc) {
        try {
            // Wait max 10 seconds for thread to die.
            join(10*1000);
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }

    private void handlerLoop() throws InterruptedException {
        while (!_exit_loop) {
            // Handle events in this loop. Sleep not
            // needed, block on event queue.
            sleep(1 * 1000);
        }
    }
}
```

Java 同期の例

Java 同期論理は Mobile Link や共通 Java クラスと連動し、Mobile Link サーバを使用したアプリケーションの配備に柔軟性を提供します。次の項では、単純な例を使用して、この広範囲な機能について説明します。

この項では、Java 同期論理の実例を挙げて説明します。このクラスを使用したり独自のクラスを作成したりする前に、次のチェックリストを使用してすべてを満たしているかどうかを確認してから、クラスをコンパイルしてください。

- 擬似コードなどを使用する機能の計画。
- データベーステーブルとカラムのマッピングの作成。
- Mobile Link システムテーブルで、Java 同期メソッドの言語タイプとロケーションを指定していることを確認して、Java 同期用の統合データベースを設定。

  「Java 同期論理の設定」459 ページを参照してください。
- Java クラスの実行中に呼び出される関連 Java クラスのリストの作成。
- Mobile Link サーバのクラスパスに含まれるロケーションに Java クラスを格納。

プラン

この例の Java 同期論理は、例の処理に必要な機能を指定する関連 Java ファイルとクラスを指しています。この例は、クラス CustEmpScripts の作成方法を示します。また、接続用同期コンテキストの設定方法も示します。最後に、次の用途を持つ Java メソッドを提供します。

- Mobile Link ユーザを認証する。
- 各データベーステーブル用のカーソルを使用して、ダウンロード操作とアップロード操作を実行する。

スキーマ

同期対象となるテーブルは emp と cust です。emp テーブルには、3 つのカラム emp_id、emp_name、manager があります。cust テーブルには、cust_id、cust_name、emp_id という 3 つのカラムがあります。各テーブルのすべてのカラムが同期されます。統合データベースからリモートデータベースへのマッピングにより、テーブル名とカラム名はどちらのデータベースでも同じです。さらに、監査テーブルが統合データベースに追加されます。

Java クラスファイル

この例で使用するファイルは *Samples¥MobiLink¥JavaAuthentication* フォルダにあります。

設定

次のコードは Java 同期論理を設定します。import 文は、Java VM に対して、必要なファイルのロケーションを指定します。public class 文はクラスを宣言します。

```
// Use a package when you create your own script.
import ianywhere.ml.script.InOutInteger;
import ianywhere.ml.script.DBConnectionContext;
import ianywhere.ml.script.ServerContext;
import java.sql.*;

public class CustEmpScripts {
   // Context for this synchronization connection.
   DBConnectionContext _conn_context;

   // Same connection MobiLink uses for sync.
   // Do not commit or close this.
   Connection _sync_connection;
   Connection _audit_connection;

   //Get a user id given the user name. On audit connection.
```

```
    PreparedStatement _get_user_id_pstmt;

    // Add record of user logins added. On audit connection.
    PreparedStatement _insert_login_pstmt;

    // Prepared statement to add a record to the audit table.
    // On audit connection.
    PreparedStatement _insert_audit_pstmt;

    // ...
}
```

CustEmpScripts コンストラクタは、authenticateUser メソッドの準備文をすべて設定します。また、メンバーデータも設定します。

```
public CustEmpScripts(DBConnectionContext cc) throws SQLException {
    try {
        _conn_context = cc;
        _sync_connection = _conn_context.getConnection();

        ServerContext serv_context =
        _conn_context.getServerContext();
        _audit_connection = serv_context.makeConnection();

        // Get the prepared statements ready.
        _get_user_id_pstmt =
            _audit_connection.prepareStatement(
                "select user_id from ml_user where name = ?"
            );

        _insert_login_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_added(ml_user, add_time) "
                    + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) })"
            );

        _insert_audit_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_audit(ml_user_id, audit_time, audit_action) "
                    + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) }, ?)"
            );
    }
    catch(SQLException e) {
        freeJDBCResources();
        throw e;
    }
    catch(Error e) {
        freeJDBCResources();
        throw e;
    }
}
```

finalize メソッドは、end_connection が呼び出されなければ JDBC リソースをクリーンアップします。freeJDBCResources メソッドを呼び出し、割り付けられたメモリを解放して、監査接続を終了します。

```
protected void finalize() throws SQLException, Throwable {
    super.finalize();
    freeJDBCResources();
}

private void freeJDBCResources() throws SQLException {
```

```
    if (_get_user_id_pstmt != null) {
        _get_user_id_pstmt.close();
    }
    if (_insert_login_pstmt != null) {
        _insert_login_pstmt.close();
    }
    if (_insert_audit_pstmt != null) {
        _insert_audit_pstmt.close();
    }
    if (_audit_connection != null) {
        _audit_connection.close();
    }
    _conn_context       = null;
    _sync_connection    = null;
    _audit_connection   = null;
    _get_user_id_pstmt  = null;
    _insert_login_pstmt = null;
    _insert_audit_pstmt = null;
}
```

endConnection メソッドは、不要になった時点でリソースをクリーンアップします。

```
public void endConnection() throws SQLException {
    freeJDBCResources();
}
```

次の authenticateUser メソッドは、すべてのユーザログインを承認し、データベーステーブルにユーザ情報のログを取ります。ユーザが ml_user テーブルにない場合は、そのログが login_added に書き込まれます。ユーザ ID が ml_user 内で見つかると、そのログが login_audit に書き込まれます。実際のシステムでは user_password を無視することはありませんが、この例では単純にするためにすべてのユーザを承認しています。データベース操作のいずれかが失敗して例外が発生すると、endConnection メソッドは SQLException を発行します。

```
public void authenticateUser(
    InOutInteger authentication_status,
    String user_name) throws SQLException
{

    boolean new_user;
    int user_id;

    // Get ml_user id.
    _get_user_id_pstmt.setString(1, user_name);

    ResultSet user_id_rs =
    _get_user_id_pstmt.executeQuery();
    new_user = !user_id_rs.next();
    if (!new_user) {
        user_id = user_id_rs.getInt(1);
    }
    else {
        user_id = 0;
    }

    user_id_rs.close();
    user_id_rs = null;

    // In this tutorial always allow the login.
    authentication_status.setValue(1000);

    if (new_user) {
```

```
        _insert_login_pstmt.setString(1, user_name);
        _insert_login_pstmt.executeUpdate();
        java.lang.System.out.println("user: " + user_name + " added. ");
    }
    else {
        _insert_audit_pstmt.setInt(1, user_id);
        _insert_audit_pstmt.setString(2, "LOGIN ALLOWED");
        _insert_audit_pstmt.executeUpdate();
    }
    _audit_connection.commit();
    return;
}
```

次のメソッドは、SQL 文を使用して、データベーステーブル上でカーソルとして動作します。これらはカーソルスクリプトであるため、SQL 文字列を返します。

```
public static String empUploadInsertStmt() {
    return("INSERT INTO emp(emp_id, emp_name) VALUES(?, ?)");
}

public static String empUploadDeleteStmt() {
    return("DELETE FROM emp WHERE emp_id = ?");
}

public static String empUploadUpdateStmt() {
    return("UPDATE emp SET emp_name = ? WHERE emp_id = ?");
}

public static String empDownloadCursor() {
    return("SELECT emp_id, emp_name FROM emp");
}

public static String custUploadInsertStmt() {
    return("INSERT INTO cust(cust_id, emp_id, cust_name) VALUES (?, ?, ?)");
}

public static String custUploadDeleteStmt() {
    return("DELETE FROM cust WHERE cust_id = ?");
}

public static String custUploadUpdateStmt() {
    return("UPDATE cust SET emp_id = ?, cust_name = ? WHERE cust_id = ?");
}

public static String custDownloadCursor() {
    return("SELECT cust_id, emp_id, cust_name FROM cust");
}
```

次のコマンドを使用して、コードをコンパイルします。

```
javac -cp %sqlany16%¥java¥mlscript.jar CustEmpScripts.java
```

クラスパスの CustEmpScripts.class のロケーションを使用して Mobile Link サーバを実行します。次に、コマンドラインの一部を示します。

```
mlsrv16 ... -sl java (-cp <class_location>)
```

Mobile Link サーバ Java API リファレンス

この項では、Mobile Link Java のインタフェースとクラス、これらに関連するメソッド、コンストラクタについて説明します。これらのクラスを使用するには、*%SQLANY16%¥java¥* にある *mlscript.jar* アセンブリを参照してください。

パッケージ

```
ianywhere.ml.script
```

DBConnectionContext インタフェース

現在のデータベース接続に関する情報を取得します。

構文

```
public interface DBConnectionContext
```

メンバー

継承されたメンバーを含む DBConnectionContext インタフェースのすべてのメンバー。

名前	説明
getConnection メソッド	Mobile Link 統合データベースへの既存の接続を返します。
getDownloadData メソッド	現在の同期に対する DownloadData を返します。
getNetworkData メソッド	同期用のネットワークストリームの情報を返します。
getProperties メソッド	スクリプトバージョンに基づいて、この接続のプロパティを返します。
getRemoteID メソッド	この接続で現在同期中のデータベースのリモート ID を返します。
getServerContext メソッド	この Mobile Link サーバの ServerContext を返します。
getVersion メソッド	この接続のバージョン文字列を返します。

備考

このインタフェースは、スクリプトを含むクラスのコンストラクタに渡されます。コンテキストがバックグラウンドスレッドに必要な場合や、接続期間を超えて必要な場合は、**ServerContext** インタフェースを使用してください。

コンストラクタの詳細については、「コンストラクタ」463 ページを参照してください。

> **注意**
> **DBConnectionContext** インスタンスは、Java コードに呼び出すスレッド以外で使用しないでください。

参照

- ServerContext インタフェース [Mobile Link サーバ Java]501 ページ

例

次の例は、同期スクリプトで使用するクラスレベルの **DBConnectionContext** インスタンスを作成する方法を示します。**DBConnectionContext.getConnection** メソッドは、Mobile Link 統合データベースとの現在の接続を表す **DBConnection** インスタンスを取得します。

```
using iAnywhere.MobiLink.Script;
using System.Data;

public class OrderProcessor {
   DBConnectionContext _cc;

   public OrderProcessor( DBConnectionContext cc ) {
      _cc = cc;
   }

   // The method used for the handle_DownloadData event.
   public void HandleEvent() {
      DBConnection my_connection = _cc.GetConnection();
      // ...
   }

   // ...
}
```

getConnection メソッド

Mobile Link 統合データベースへの既存の接続を返します。

構文

```
java.sql.Connection DBConnectionContext.getConnection()
    throws SQLException
```

戻り値

JDBC 接続として存在する既存の接続。

例外

- **java.sql.SQLException**　既存の接続を JDBC 接続としてバインドするときにエラーが発生するとスローされます。

備考

この接続は、この同期用に Mobile Link が SQL スクリプトの実行時に使用するものと同じです。

この接続は、Mobile Link サーバのこの接続での使用に影響する方法でコミット、クローズ、または変更しないでください。返される接続は、基本となる Mobile Link 接続の期間中にのみ有効です。

注意
接続に対して end_connection イベントが呼び出された後は、その接続を使用しないでください。

参照

- ServerContext.makeConnection メソッド [Mobile Link サーバ Java]507 ページ
- DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページ

getDownloadData メソッド

現在の同期に対する DownloadData を返します。

構文

```
DownloadData DBConnectionContext.getDownloadData()
```

戻り値

現在の同期の DownloadData。同期がダウンロードされてない場合は Null です。

備考

ダイレクトローハンドリング用のダウンロードを作成する場合は、DownloadData クラスを使用してください。ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadData インタフェース [Mobile Link サーバ Java]478 ページ

例

次の例は、**DBConnectionContext.getDownloadData** メソッドを使用して、現在の同期に対する **DownloadData** インスタンスを取得する方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void HandleDownload() throws SQLException {
   // Get the DownloadData for the current synchronization
   DownloadData my_dd = _cc.getDownloadData();
   // ...
}

// ...
```

getNetworkData メソッド

同期用のネットワークストリームの情報を返します。

構文

```
NetworkData DBConnectionContext.getNetworkData()
```

戻り値

要求に使用されたネットワークストリームの情報、コレクションが有効でない場合は NULL

備考

この方法は、クライアント側証明書と HTTP ヘッダを使用するエンタープライズ内の別のサーバを認証する際に便利です。

ネットワークストリームのデータの収集を有効にするには、-x スイッチに collect_network_data=1 を追加します。このオプションにより、同期ごとのデータの保存時に、メモリのオーバヘッドが増えます。

参照

- NetworkData インタフェース [Mobile Link サーバ Java]497 ページ

getProperties メソッド

スクリプトバージョンに基づいて、この接続のプロパティを返します。

構文

```
Properties DBConnectionContext.getProperties()
```

戻り値

この接続のプロパティ

備考

プロパティは、**ml_property** テーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

java.util.Properties の詳細については、Java Software Development Kit マニュアルを参照してください。

例

次の例は、**DBConnectionContext** のプロパティを出力する方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used to output the connection properties.
public void outputProperties() {
    // Output the properties for the current synchronization
    java.util.Properties properties = _cc.getProperties();
    System.out.println(properties.toString());
}
```

getRemoteID メソッド

この接続で現在同期中のデータベースのリモート ID を返します。

構文

```
String DBConnectionContext.getRemoteID()
```

戻り値

リモート ID。

備考

リモート ID の詳細については、「リモート ID」『Mobile Link クライアント管理』を参照してください。

例

次の例は、**DBConnectionContext** のリモート ID を出力する方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used to output the remote ID.
public void outputRemoteID() {
    // output the Remote ID for the current synchronization
    String remoteID = _cc.getRemoteID();
    System.out.println(remoteID);
}
```

getServerContext メソッド

この Mobile Link サーバの ServerContext を返します。

構文

```
ServerContext DBConnectionContext.getServerContext()
```

戻り値

Mobile Link サーバコンテキスト。

参照

- ServerContext インタフェース [Mobile Link サーバ Java]501 ページ

例

次の例は、**DBConnectionContext** の **ServerContext** のインスタンスを取得する方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// A method that uses a ServerContext instance to shut down the server
public void shutDownServer() {
    ServerContext context = _cc.getServerContext();
    context.shutdown();
}
```

getVersion メソッド

この接続のバージョン文字列を返します。

構文

```
String DBConnectionContext.getVersion()
```

戻り値

スクリプトバージョン。

備考

詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

例

次の例は、スクリプトバージョンを取得して、その値に基づいて決定を行う方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// A method that uses the script version
public void handleEvent() {
    // ...

    String version = _cc.getVersion();
    if (version.equals("My Version 1")) {
        // ...
    } else if (version.equals("My Version 2")) {
        // ...
    }
}
// ...
```

DownloadData インタフェース

ダイレクトローハンドリングで使用するダウンロードデータ操作をカプセル化します。

構文

```
public interface DownloadData
```

メンバー

継承されたメンバーを含む DownloadData インタフェースのすべてのメンバー。

名前	説明
getDownloadTableByName メソッド	現在の同期に使用する名前付きダウンロードテーブルを取得します。
getDownloadTables メソッド	現在の同期のダウンロードのすべてのテーブルを含む配列を取得します。

備考

DownloadData のインスタンスを取得するには、**DBConnectionContext.getDownloadData** メソッドを使用します。

DownloadTableData のインスタンスを返すには、**getDownloadTables** メソッドと **getDownloadTableByName** メソッドを使用します。

ダウンロードデータは **DBConnectionContext** を使用して取得できます。begin_synchronization イベントの前または end_download イベントの後にダウンロードデータにアクセスすることはできません。また、**DownloadData** にアップロード専用同期でアクセスすることもできません。詳細については、「handle_DownloadData 接続イベント」371 ページを参照してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DBConnectionContext.getDownloadData メソッド[Mobile Link サーバ Java]475 ページ
- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ

例

次の例は、**DBConnectionContext.getDownloadData** メソッドを使用して、現在の同期に対する **DownloadData** インスタンスを取得する方法を示します。

```
DBConnectionContext _cc;

// Your class constructor.
public OrderProcessor(DBConnectionContext cc) {
    _cc = cc;
}

// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // ...
}
```

getDownloadTableByName メソッド

現在の同期に使用する名前付きダウンロードテーブルを取得します。

構文

DownloadTableData **DownloadData.getDownloadTableByName**(String *table_name*)

パラメータ

- **table_name**　ダウンロードデータの取得先テーブルの名前

戻り値

指定されたテーブルを表す **DownloadTableData** インスタンス。指定された名前のテーブルが現在の同期で存在しない場合は null。

備考

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadData インタフェース [Mobile Link サーバ Java]478 ページ
- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ
- DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページ

例

次の例では **getDownloadTableByName** メソッドを使用して、**remoteOrders** テーブルの **DownloadTableData** インスタンスを返します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.getDownloadData();

   // Get the DownloadTableData for the remoteOrders table.
   DownloadTableData my_download_table = my_dd.getDownloadTableByName("remoteOrders");

   // ...
}
```

getDownloadTables メソッド

現在の同期のダウンロードのすべてのテーブルを含む配列を取得します。

構文

```
DownloadTableData[] DownloadData.getDownloadTables()
```

戻り値

現在の同期の **DownloadTableData** オブジェクトの配列。配列内でのテーブルの順序は、リモートのアップロード順と同じです。

備考

このテーブルに対して実行された操作はリモートデータベースに送信されます。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadData インタフェース [Mobile Link サーバ Java]478 ページ
- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ
- DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページ

例

次の例では **DownloadData.getDownloadTables** メソッドを使用して、現在の同期の **DownloadTableData** オブジェクトの配列を取得します。例では、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.getDownloadData();

   // Get an array of tables to set download operations.
   DownloadTableData[] download_tables = my_dd.getDownloadTables();

   // Get the first table in the DownloadTableData array.
   DownloadTableData my_download_table = download_tables[0];

   // ...
}
```

DownloadTableData インタフェース

1 つのダウンロードテーブルの情報を同期用にカプセル化します。

構文

```
public interface DownloadTableData
```

メンバー

継承されたメンバーを含む DownloadTableData インタフェースのすべてのメンバー。

名前	説明
getDeletePreparedStatement メソッド	ユーザが削除操作をダウンロードに追加できるようにする **java.sql.PreparedStatement** インスタンスを返します。
getLastDownloadTime メソッド	このテーブルの最終ダウンロード時刻を返します。
getMetaData メソッド	**DownloadTableData** インスタンスのメタデータを取得します。
getName メソッド	DownloadTableData インスタンスのテーブル名を返します。

名前	説明
getUpsertPreparedStatement メソッド	ユーザがアップサート (更新または挿入) 操作を同期のダウンロードに追加できるようにする **java.sql.PreparedStatement** インスタンスを返します。

備考

このインタフェースを使用して、クライアントにダウンロードされるデータ操作を設定します。

DownloadData インタフェースを使用して、現在の同期の **DownloadTableData** インスタンスを取得できます。**getUpsertPreparedStatement** と **getDeletePreparedStatement** メソッドを使用して、それぞれ更新／挿入操作と削除操作を行う Java 準備文を取得できます。

テーブルをリモートでトランケートするため、すべてのプライマリキーを NULL に設定して削除文を実行できます。

java.sql.PreparedStatement.executeUpdate メソッドは、ダウンロードに使用する操作を登録します。**java.sql.PreparedStatement** の詳細については、Java Software Development Kit マニュアルを参照してください。

> **注意**
> 挿入と更新用の準備文のすべてのカラム値を設定してください。削除操作の場合は、プライマリキー値を設定します。削除とアップサート準備文を両方同時に開いておくことはできません。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadData インタフェース [Mobile Link サーバ Java]478 ページ

例

この例では、次の SQL 文を使用して作成された Mobile Link クライアントデータベースに **remoteOrders** というテーブルがあることを前提にしています。

```
CREATE TABLE remoteOrders (
   pk INT NOT NULL,
   col1 VARCHAR(200),
   PRIMARY KEY (pk)
);
```

次の例では **DownloadData.getDownloadTableByName** メソッドを使用して、**remoteOrders** テーブルを表す **DownloadTableData** インスタンスを返します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {

   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.getDownloadData();
```

 Copyright © 2013, SAP AG or an SAP affiliate company.

```
    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // User defined-methods to set download operations.
    setDownloadInserts(td);
    setDownloadDeletes(td);
    // ...
}
```

この例では、**SetDownloadInserts** メソッドは **GetUpsertCommand** を使用して、挿入または更新するローのコマンドを取得します。**IDbCommand** は、リモートデータベースに挿入する値の設定先となるパラメータを保持します。

```
void setDownloadInserts(DownloadTableData td) {
    java.sql.PreparedStatement insert_ps = td.getUpsertPreparedStatement();

    // The following method calls are the same as the following SQL statement:
    // INSERT INTO remoteOrders(pk, col1) values(2300, "truck");
    insert_ps.setInt(1, 2300);
    insert_ps.setString(2, "truck");

    int update_result = insert_ps.executeUpdate();
    if (update_result == 0) {
        // Insert was filtered because it was uploaded
        // in the same synchronization.
    }
    else {
        // Insert was not filtered.
    }
}
```

setDownloadDeletes メソッドは、**DownloadTableData.getDeletePreparedStatement** を使用して、削除したいローの準備文を取得します。**java.sql.PreparedStatement.setInt** メソッドは、リモートデータベースの削除したいローのプライマリキーの値を設定し、**java.sql.PreparedStatement.executeUpdate** メソッドは、ダウンロードに使用するローの値を登録します。

```
void setDownloadDeletes(DownloadTableData td) {
    java.sql.PreparedStatement delete_ps = td.getDeletePreparedStatement();

    // The following method calls are the same as the following SQL statement:
    // DELETE FROM remoteOrders  where pk=2300;
    delete_ps.setInt(1, 2300);
    delete_ps.executeUpdate();
}
```

getDeletePreparedStatement メソッド

ユーザが削除操作をダウンロードに追加できるようにする **java.sql.PreparedStatement** インスタンスを返します。

構文

```
java.sql.PreparedStatement DownloadTableData.getDeletePreparedStatement()
    throws java.sql.SQLException
```

戻り値

削除操作をダウンロードに追加するための **java.sql.PreparedStatement** インスタンス

例外

- **java.sql.SQLException**　削除された **java.sql.PreparedStatement** インスタンスの取得で問題が発生した場合にスローされます。

備考

この準備文は **DownloadTableData** インスタンスに適用され、テーブルの各プライマリキーカラムに対するパラメータを含んでいます。

ダウンロードに削除操作を含めるには、**java.sql.PreparedStatement** のすべてのカラムを設定してから、**java.sql.PreparedStatement.executeUpdate** メソッドを呼び出します。

テーブルをリモートデータベースでトランケートするため、すべてのパラメータを NULL に設定します。

> **注意**
> ダウンロード削除操作対象のすべてのプライマリキー値を設定するか、トランケート操作対象のすべてのプライマリキー値を NULL に設定してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ

例

次の例では、**setDownloadDeletes** メソッドは、**getDeletePreparedStatement** を使用して、削除したいローの準備文を取得しています。**java.sql.PreparedStatement.setInt** メソッドは、リモートデータベースの削除したいローのプライマリキーの値を設定し、**java.sql.PreparedStatement.executeUpdate** メソッドは、ダウンロードのローの値を設定します。

```
void setDownloadDeletes(DownloadTableData td) {
   java.sql.PreparedStatement delete_ps = td.getDeletePreparedStatement();
   // This is the same as executing the following SQL statement:
   // DELETE FROM remoteOrders  where pk=2300;
   delete_ps.setInt(1, 2300);
   delete_ps.executeUpdate();
   delete_ps.close();
}
```

getLastDownloadTime メソッド

このテーブルの最終ダウンロード時刻を返します。

構文

```
java.sql.Timestamp DownloadTableData.getLastDownloadTime()
```

戻り値

このダウンロードテーブルの最終ダウンロード時刻

備考

これは、テーブルごとのダウンロードイベントの多くで渡される最終ダウンロード時刻と同じです。

最終ダウンロード時刻は、特定の同期に対してテーブルダウンロードデータを生成する場合に便利です。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ

例

次の例では、前回のダウンロード時刻を使用した挿入によって、ダウンロードのテーブルにデータを移植する方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.getDownloadData();

   // Get the DownloadTableData for the remoteOrders table.
   DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

   // Get the inserts given a last download time.
   ResultSet inserts_rs  = makeInsertsFromTimestamp(td.getLastDownloadTime());

   // Fill the DownloadTableData using the inserts resultset.
   setDownloadInsertsFromRS(td, inserts_rs);
   inserts_rs.close();

   // ...
}
```

getMetaData メソッド

DownloadTableData インスタンスのメタデータを取得します。

構文

`java.sql.ResultSetMetaData` **`DownloadTableData.getMetaData`**`()`

戻り値

DownloadTableData インスタンスのメタデータ

備考

メタデータは標準 **java.sql.ResultSetMetaData** オブジェクトです。

メタデータにカラム名情報を含める場合は、クライアントで、アップロードとともにカラム名が送信されるように指定します。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

java.sql.ResultSetMetaData の詳細については、Java Software Development Kit マニュアルを参照してください。

参照

- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ

例

次の例は、**DownloadTableData** インスタンスのクエリで使用されるカラムの数を取得する方法を示します。

```
import java.sql.ResultSetMetaData;

// The method used to return the number of columns in a DownloadTableData instance query
public int getNumColumns(DownloadTableData td) {
    ResultSetMetaData rsmd = td.getMetaData();
    return rsmd.getColumnCount();
}
```

getName メソッド

DownloadTableData インスタンスのテーブル名を返します。

構文

```
String DownloadTableData.getName()
```

戻り値

DownloadTableData インスタンスのテーブル名

備考

getMetaData メソッドによって返される **java.sql.ResultSetMetaData** インスタンスを使用することによっても、テーブル名にアクセスできます。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ
- DownloadTableData.getMetaData メソッド [Mobile Link サーバ Java]485 ページ

例

次の例は、**DownloadTableData** インスタンスのテーブル名を出力する方法を示します。この例は、**_cc** という **DBConnectionContext** インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // Print the table name to standard output (remoteOrders)
    System.out.println(td.getName());

    // User defined-methods to set download operations.
    setDownloadInserts(td);
    setDownloadDeletes(td);

    // ...
}
```

getUpsertPreparedStatement メソッド

ユーザがアップサート (更新または挿入) 操作を同期のダウンロードに追加できるようにする **java.sql.PreparedStatement** インスタンスを返します。

構文

```
java.sql.PreparedStatement DownloadTableData.getUpsertPreparedStatement()
    throws java.sql.SQLException
```

戻り値

アップサート操作をダウンロードに追加するための **java.sql.PreparedStatement** インスタンス

例外

- **java.sql.SQLException**　更新された **java.sql.PreparedStatement** インスタンスの取得で問題が発生した場合にスローされます。

備考

この準備文は DownloadTableData インスタンスに適用され、テーブルの各カラムに対するパラメータを含んでいます。

ダウンロードに挿入または更新操作を含めるには、**java.sql.PreparedStatement** ですべてのカラム値を設定してから、**java.sql.PreparedStatement.executeUpdate** メソッドを呼び出します。準備文に対する **java.sql.PreparedStatement.executeUpdate** の呼び出しでは、挿入/更新操作がフィルタされた場合は 0 を返し、フィルタされていない場合は 1 を返します。同じ同期でアップロードされた場合、操作はフィルタされます。

> **注意**
> ダウンロード挿入と更新操作のすべてのカラム値を設定してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ

例

次の例では、**setDownloadInserts** メソッドは、**getUpsertPreparedStatement** を使用して、挿入または更新したいローの準備文を取得しています。**java.sql.PreparedStatement.setInt** メソッドと **PreparedStatement.setString** メソッドはダウンロードのカラム値を設定し、**PreparedStatement.executeUpdate** メソッドはロー値を設定します。

```
void setDownloadInserts(DownloadTableData td) {
   java.sql.PreparedStatement insert_ps = td.getUpsertPreparedStatement();

   // This is the same as executing the following SQL statement:
   // INSERT INTO remoteOrders(pk, col1) VALUES (2300, "truck");
   insert_ps.setInt(1, 2300);
   insert_ps.setString(2, "truck");

   int update_result = insert_ps.executeUpdate();
   if (update_result == 0) {
      // Insert was filtered because it was uploaded
      // in the same synchronization.
   }
   else {
      // Insert was not filtered.
   }
   insert_ps.close();
}
```

InOutInteger インタフェース

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

構文

```
public interface InOutInteger
```

メンバー

継承されたメンバーを含む InOutInteger インタフェースのすべてのメンバー。

名前	説明
getValue メソッド	この整数パラメータの値を返します。
setValue メソッド	この整数パラメータの値を設定します。

例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン **ver1** を同期するときに、**handleError** という Java メソッドを **handle_error connection** イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'handle_error',
   'ExamplePackage.ExampleClass.handleError'
)
```

次に示すサンプルは、**handleError** という Java メソッドです。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラーコードも判別します。

```
public String handleError(
   ianywhere.ml.script.InOutInteger actionCode,
   int errorCode,
   String errorMessage,
   String user,
   String table)
{

   int new_ac;
   if (user == null) {
      new_ac = handleNonSyncError(errorCode, errorMessage);
   } else if (table == null) {
      new_ac = handleConnectionError(errorCode, errorMessage, user);
   }
   else {
      new_ac = handleTableError(errorCode, errorMessage, user, table);
   }

   // Keep the most serious action code.
   if (actionCode.getValue() < new_ac) {
      actionCode.setValue(new_ac);
   }
}
```

getValue メソッド

この整数パラメータの値を返します。

構文

`int` **`InOutInteger.getValue`**`()`

戻り値

この整数の値。

参照

- InOutInteger インタフェース [Mobile Link サーバ Java]488 ページ

setValue メソッド

この整数パラメータの値を設定します。

構文

`void` **`InOutInteger.setValue`**`(int` *`new_value`*`)`

パラメータ

- **new_value**　この整数が取る値

参照

- InOutInteger インタフェース [Mobile Link サーバ Java]488 ページ

InOutString インタフェース

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

構文

```
public interface InOutString
```

メンバー

継承されたメンバーを含む InOutString インタフェースのすべてのメンバー。

名前	説明
getValue メソッド	この文字列パラメータの値を返します。
setValue メソッド	この文字列パラメータの値を設定します。

例

次の Mobile Link システムプロシージャコールは、スクリプトバージョン **ver1** を同期するときに、**modifyUser** という Java メソッドを **modify_user** 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'modify_user',
   'ExamplePackage.ExampleClass.modifyUser'
)
```

次に示すサンプルは、**modifyUser** という Java メソッドです。このメソッドは、データベースからユーザ ID を取得し、それを使用してユーザ名を設定します。

```
public String modifyUser(InOutString io_user_name) throws SQLException {
   Statement uid_select = curConn.createStatement();
   ResultSet uid_result = uid_select.executeQuery(
      "SELECT rep_id FROM SalesRep WHERE name = '"
         + io_user_name.getValue() + "' "
   );
   uid_result.next();
   io_user_name.setValue(java.lang.Integer.toString(uid_result.getInt(1));
   uid_result.close();
   uid_select.close();
   return (null);
}
```

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

getValue メソッド

この文字列パラメータの値を返します。

構文

```
String InOutString.getValue()
```

戻り値

この文字列パラメータの値

参照

- InOutString インタフェース [Mobile Link サーバ Java]490 ページ

setValue メソッド

この文字列パラメータの値を設定します。

構文

```
void InOutString.setValue(String new_value)
```

パラメータ

- **new_value**　この文字列が取る値

参照

- InOutString インタフェース [Mobile Link サーバ Java]490 ページ

LogListener インタフェース

ログに出力されるメッセージを取得するために使用されます。

構文

```
public interface LogListener
```

メンバー

継承されたメンバーを含む LogListener インタフェースのすべてのメンバー。

名前	説明
messageLogged メソッド	メッセージがログに出力されたときに呼び出されます。

備考

詳細については、「Java での Mobile Link サーバエラーの処理」465 ページを参照してください。

messageLogged メソッド

メッセージがログに出力されたときに呼び出されます。

構文

```
void LogListener.messageLogged(ServerContext sc, LogMessage message)
```

パラメータ

- **sc** メッセージを出力しているサーバのコンテキスト
- **message** Mobile Link ログに送信された **LogMessage**。

備考

詳細については、「Java での Mobile Link サーバエラーの処理」465 ページを参照してください。

LogMessage クラス

ログメッセージに関連付けられたデータを保持します。

構文

```
public class LogMessage
```

メンバー

継承されたメンバーを含む LogMessage クラスのすべてのメンバー。

名前	説明
getText メソッド	このメッセージに対応するテキストにアクセスします。
getType メソッド	このメッセージタイプにアクセスします。
getUser メソッド	このメッセージに対応するユーザ名にアクセスします。
ERROR 変数	ログメッセージがエラーであることを示します。
INFO 変数	メッセージログに情報が表示されることを示します。
WARNING 変数	ログメッセージが警告であることを示します。

備考

詳細については、「Java での Mobile Link サーバエラーの処理」465 ページを参照してください。

getText メソッド

このメッセージに対応するテキストにアクセスします。

構文

```
String LogMessage.getText()
```

戻り値

このメッセージの本文

参照

- LogMessage.ERROR 変数 [Mobile Link サーバ Java]494 ページ
- LogMessage.INFO 変数 [Mobile Link サーバ Java]495 ページ
- LogMessage.WARNING 変数 [Mobile Link サーバ Java]496 ページ

getType メソッド

このメッセージタイプにアクセスします。

構文

```
int LogMessage.getType()
```

戻り値

このメッセージのタイプ (**LogMessage.ERROR**、**LogMessage.INFO**、または **LogMessage.WARNING**)。

参照

- LogMessage.ERROR 変数 [Mobile Link サーバ Java]494 ページ
- LogMessage.INFO 変数 [Mobile Link サーバ Java]495 ページ
- LogMessage.WARNING 変数 [Mobile Link サーバ Java]496 ページ

getUser メソッド

このメッセージに対応するユーザ名にアクセスします。

構文

```
String LogMessage.getUser()
```

戻り値

このメッセージに対応するユーザメッセージに対応するユーザがない場合には、この値が NULL になる場合があります。

参照

- LogMessage.ERROR 変数 [Mobile Link サーバ Java]494 ページ
- LogMessage.INFO 変数 [Mobile Link サーバ Java]495 ページ
- LogMessage.WARNING 変数 [Mobile Link サーバ Java]496 ページ

ERROR 変数

ログメッセージがエラーであることを示します。

構文

```
final int LogMessage.ERROR
```

参照

- LogMessage クラス [Mobile Link サーバ Java]492 ページ

例

次のコードは、すべてのエラーメッセージ用に **LogListener** オブジェクトをインストールしてから、その情報をファイルに書き込みます。

```
class ErrorLogListener implements LogListener {
   FileOutputStream _outFile;

   public ErrorLogListener(FileOutputStream outFile) {
      _outFile = outFile;
   }

   public void messageLogged(ServerContext sc, LogMessage msg) {
      String user;

      try {
         if (msg.getType() != LogMessage.ERROR) {
            //this class deals exclusively with errors
            return;
         }

         user = msg.getUser();
         if (user == null) {
            user = "NULL";
         }

         _outFile.write(("Caught error"
               + " user=" + user
               + " text=" + msg.getText()
               + "¥n").getBytes()
         );
         _outFile.flush();
      }
      catch(Exception e) {
         // Print some error output to the MobiLink log.
         e.printStackTrace();
      }
   }
}
```

次のコードは **ErrorLogListener** オブジェクトを登録して、エラーメッセージを受信します。クラスコンストラクタや同期スクリプトなど、**ServerContext** にアクセスできる任意の場所からこのコードを呼び出してください。

```
ServerContext serv_context;
FileOutputStream outFile
serv_context.addErrorListener(new ErrorLogListener(outFile));
```

INFO 変数

メッセージログに情報が表示されることを示します。

構文

`final int` **`LogMessage.INFO`**

参照

- LogMessage クラス [Mobile Link サーバ Java]492 ページ
- ServerContext.addInfoListener メソッド [Mobile Link サーバ Java]503 ページ

例

次のコードは、すべての情報メッセージ用に **LogListener** オブジェクトをインストールしてから、その情報をファイルに書き込みます。

```
class InfoLogListener implements LogListener {
   FileOutputStream _outFile;

   public InfoLogListener(FileOutputStream outFile) {
      _outFile = outFile;
   }

   public void messageLogged(ServerContext sc, LogMessage msg) {
      String user;

      try {
         if (msg.getType() != LogMessage.INFO) {
            // this class deals exclusively with info
            return;
         }

         user = msg.getUser();
         if (user == null) {
            user = "NULL";
         }

         _outFile.write(("Caught info"
               + " user=" + user
               + " text=" + msg.getText()
               + "¥n").getBytes()
         );
         _outFile.flush();
      }
      catch(Exception e) {
         // Print some info output to the MobiLink log.
         e.printStackTrace();
      }
```

```
    }
  }
```

次のコードは **InfoLogListener** オブジェクトを登録して、情報メッセージを受信します。クラスコンストラクタや同期スクリプトなど、**ServerContext** にアクセスできる任意の場所からこのコードを呼び出してください。

```
ServerContext serv_context;
FileOutputStream outFile
serv_context.addInfoListener(new InfoLogListener(outFile));
```

WARNING 変数

ログメッセージが警告であることを示します。

構文

`final int` **`LogMessage.WARNING`**

参照

- LogMessage クラス [Mobile Link サーバ Java]492 ページ

例

次のコードは、すべての警告メッセージ用に **LogListener** オブジェクトをインストールしてから、その情報をファイルに書き込みます。

```
class WarningLogListener implements LogListener {
  FileOutputStream _outFile;

  public WarningLogListener(FileOutputStream outFile) {
    _outFile = outFile;
  }

  public void messageLogged(ServerContext sc, LogMessage msg) {
    String user;

    try {
      if (msg.getType() != LogMessage.WARNING) {
        // This class deals exclusively with warnings.
        return;
      }
      user = msg.getUser();

      if (user == null) {
        user = "NULL";
      }

      _outFile.write(("Caught warning"
        + " user=" + user
        + " text=" + msg.getText()
        + "¥n").getBytes()
      );
      _outFile.flush();
    }
    catch(Exception e) {
      // Print some warning output to the MobiLink log.
      e.printStackTrace();
    }
```

```
    }
  }
```

次のコードは **WarningLogListener** オブジェクトを登録して、警告メッセージを受信します。クラスコンストラクタや同期スクリプトなど、**ServerContext** にアクセスできる任意の場所からこのコードを呼び出してください。

```
ServerContext serv_context;
FileOutputStream outFile
serv_context.addWarningListener(new WarningLogListener(outFile));
```

NetworkData インタフェース

同期用のネットワークストリームの情報を含みます。

構文

`public interface` **`NetworkData`**

メンバー

継承されたメンバーを含む NetworkData インタフェースのすべてのメンバー。

名前	説明
getCertificateChain メソッド	クライアントから送信されたすべての証明書を含む java.security.cert.CertPath オブジェクトを返します。
getHTTPHeaders メソッド	ヘッダ名をヘッダ値のリストにマップするマップオブジェクトを返します。
getHTTPHeaderValue メソッド	指定した名前でサーバが受信した最終ヘッダ値を返します。
getHTTPHeaderValues メソッド	指定した名前と関連付けられている、サーバが受信したすべてのヘッダ値を返します。
isEndToEndEncrypted メソッド	同期がエンドツーエンドで暗号化されるか確認します。
isHTTP メソッド	同期で HTTP または HTTPS のどちらが使用されるかを確認します。
isTLS メソッド	同期で TLS が使用されるかを確認します。

備考

このインタフェースは、クライアント側証明書と HTTP ヘッダを使用するエンタープライズ内の別のサーバを認証する際に便利です。

ネットワークストリームのデータの収集を有効にするには、-x スイッチに collect_network_data=1 を追加します。このオプションにより、同期ごとのデータの保存時に、メモリのオーバヘッドが増えます。TLS または HTTPS をクライアント側証明書で使用する場合に、trusted_certificates=<certificate file>=""> を追加すると、サーバは TLS ハンドシェイクの間に証明書を送信するようクライアントに要求します。これにより、時間とネットワークコストがかかるようになります。

NetworkData オブジェクトは、DBConnectionContext クラスの getNetworkData メソッドを呼び出して取得できます。HTTP または HTTPS を使用する場合、これには、認証スクリプトが呼び出される前にサーバが受信した、最終の HTTP 要求のヘッダデータが含まれます。

参照

- DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページ

例

次に、DBConnectionContext オブジェクトから NetworkData オブジェクトを取得し、データを出力する例を示します。

```
public class OrderProcessor {
   DBConnectionContext _cc;

   public OrderProcessor( DBConnectionContext cc ) {
      _cc = cc;
   }

   // The method used for the authenticate_user event.
   public void AuthUser() {
      NetworkData nd = _cc.getNetworkData();
      if( nd != null ) {
         if( nd.isHTTP() ) {
            System.out.println( "http" );
            String user_agent = nd.getHTTPHeaderValue( "user-agent" );
            System.out.println( "  user-agent: " + user_agent.substring( 0, user_agent.indexOf( '/' ) ) );
         } else {
            System.out.println( "no http" );
         }
         if( nd.isTLS() ) {
            System.out.println( "tls" );
            CertPath certs = nd.getCertificateChain();
            if( certs != null ) {
               System.out.println( "  client-side cert:" );
               int n = 1;
               for( Certificate c : certs.getCertificates() ) {
                  System.out.println( "    cert " + n++ );
                  X509Certificate c509 = (X509Certificate) c;
                  System.out.println( "      Subject: " + c509.getSubjectX500Principal().getName() );
                  System.out.println( "      Issuer: " + c509.getIssuerX500Principal().getName() );
               }
            } else {
               System.out.println( "  no client cert" );
            }
         } else {
            System.out.println( "no tls" );
         }
         if( nd.isEndToEndEncrypted() ) {
            System.out.println( "e2ee" );
         } else {
            System.out.println( "no e2ee" );
```

```
            }
        } else {
            System.out.println( "NULL networkdata" );
        }
    }
}
```

getCertificateChain メソッド

クライアントから送信されたすべての証明書を含む java.security.cert.CertPath オブジェクトを返します。

構文

`java.security.cert.CertPath` **`NetworkData.getCertificateChain`**`()`

戻り値

クライアントを識別する X509 証明書を含む CertPath。そのような証明書が提供されなかった場合は NULL。

備考

すべての証明書は java.security.cert.X509Certificate オブジェクトです。

このメソッドによって NULL 以外の値が返されるのは、isTLS メソッドが true であり、クライアントが「identity」ストリームパラメータを使用して証明書を提供し、サーバ上で trusted_certificates オプションが設定される場合のみです。NULL 以外の CertPath 値には、自己署名証明書からピアの証明書の順の証明書が含まれます。

参照

- 「Java 用の trusted_certificates の例」 101 ページ

getHTTPHeaders メソッド

ヘッダ名をヘッダ値のリストにマップするマップオブジェクトを返します。

構文

`Map< String, List< String > >` **`NetworkData.getHTTPHeaders`**`()`

戻り値

サーバが受信したすべてのヘッダを含むマップ。

参照

- NetworkData.getHTTPHeaderValue メソッド [Mobile Link サーバ Java]500 ページ
- NetworkData.getHTTPHeaderValues メソッド [Mobile Link サーバ Java]500 ページ

getHTTPHeaderValue メソッド

指定した名前でサーバが受信した最終ヘッダ値を返します。

構文

```
String NetworkData.getHTTPHeaderValue(String name)
```

パラメータ

- **name**　値を返すヘッダの名前

戻り値

指定したヘッダ名に関連付けられている最終ヘッダ値。

参照

- NetworkData.getHTTPHeaderValues メソッド [Mobile Link サーバ Java]500 ページ
- NetworkData.getHTTPHeaders メソッド [Mobile Link サーバ Java]499 ページ

getHTTPHeaderValues メソッド

指定した名前と関連付けられている、サーバが受信したすべてのヘッダ値を返します。

構文

```
List< String > NetworkData.getHTTPHeaderValues(String name)
```

パラメータ

- **name**　値を返すヘッダの名前

戻り値

指定したヘッダ名に関連付けられているヘッダ値。

参照

- NetworkData.getHTTPHeaderValue メソッド [Mobile Link サーバ Java]500 ページ
- NetworkData.getHTTPHeaders メソッド [Mobile Link サーバ Java]499 ページ

isEndToEndEncrypted メソッド

同期がエンドツーエンドで暗号化されるか確認します。

構文

```
boolean NetworkData.isEndToEndEncrypted()
```

戻り値

同期がエンドツーエンドで暗号化されている場合は true、それ以外の場合は false を返します。

isHTTP メソッド

同期で HTTP または HTTPS のどちらが使用されるかを確認します。

構文

```
boolean NetworkData.isHTTP()
```

戻り値

同期で HTTP または HTTPS が使用される場合は true、それ以外の場合は false を返します。

isTLS メソッド

同期で TLS が使用されるかを確認します。

構文

```
boolean NetworkData.isTLS()
```

戻り値

同期で TLS が使用される場合は true、それ以外の場合は false を返します。

ServerContext インタフェース

Mobile Link サーバの継続期間中に存在する、すべてのコンテキストのインスタンス化。

構文

```
public interface ServerContext
```

メンバー

継承されたメンバーを含む ServerContext インタフェースのすべてのメンバー。

名前	説明
addErrorListener メソッド	エラーが出力されたときに通知を受信するために、指定した LogListener オブジェクトを追加します。
addInfoListener メソッド	情報が出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを追加します。
addShutdownListener メソッド	サーバコンテキストが破壊される前に通知を受信する指定の **ShutdownListener** オブジェクトを追加します。

名前	説明
addWarningListener メソッド	警告が出力されたときに通知を受信するために、指定した **LogListener** オブジェクトを追加します。
getProperties メソッド	指定されたコンポーネントとプロパティセットに関連する一連のプロパティを返します。
getPropertiesByVersion メソッド	スクリプトバージョンに関連する一連のプロパティを返します。
getPropertySetNames メソッド	指定したコンポーネントのプロパティセット名のリストを返します。
getStartClassInstances メソッド	サーバ起動時に構築された起動クラスの配列を取得します。
makeConnection メソッド	新しいサーバ接続を開いて、返します。
removeErrorListener メソッド	エラーが出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを削除します。
removeInfoListener メソッド	情報が出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを削除します。
removeShutdownListener メソッド	この **ServerContext** が破棄される前に通知を受信するリスナオブジェクトのリストから、指定した **ShutdownListener** オブジェクトを削除します。
removeWarningListener メソッド	警告が出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを削除します。
shutdown メソッド	サーバを強制的に停止します。

備考

このコンテキストを静的なデータとして保持し、バックグラウンドスレッドで使用できます。Mobile Link によって起動される Java VM の継続期間中は有効です。

ServerContext インスタンスにアクセスするには、**DBConnectionContext.getServerContext** メソッドを使用します。

addErrorListener メソッド

エラーが出力されたときに通知を受信するために、指定した LogListener オブジェクトを追加します。

構文

```
void ServerContext.addErrorListener(LogListener ll)
```

パラメータ

- **ll** エラーが通知される LogListener オブジェクト。

備考

エラーが出力されたときは、**LogListener.messageLogged(ServerContext, LogMessage)** メソッドが呼び出されます。

参照

- LogListener.messageLogged メソッド [Mobile Link サーバ Java]492 ページ
- LogMessage クラス [Mobile Link サーバ Java]492 ページ

addInfoListener メソッド

情報が出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを追加します。

構文

```
void ServerContext.addInfoListener(LogListener ll)
```

パラメータ

- **ll** 情報が通知される **LogListener** オブジェクト。

備考

LogListener.messageLogged メソッドが呼び出されます。

参照

- LogListener.messageLogged メソッド [Mobile Link サーバ Java]492 ページ

例

次のコードは **MyLogListener** オブジェクトを登録して、情報メッセージの通知を受信します。

```
// ServerContext serv_context;
serv_context.addInfoListener(new MyLogListener(ll_out_file));

// The following code shows an example of processing those messages:
class MyLogListener implements LogListener {
   FileOutputStream _out_file;
   public TestLogListener(FileOutputStream out_file) {
      _out_file = out_file;
   }
```

```
    public void messageLogged(ServerContext sc, LogMessage msg) {
        String type;
        String user;

        try {
            if (msg.getType() == LogMessage.ERROR) {
                type = "ERROR";
            } else if (msg.getType() == LogMessage.WARNING) {
                type = "WARNING";
            } else if (msg.getType() == LogMessage.INFO) {
                type = "INFO";
            } else {
                type = "UNKNOWN!!!";
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }
            _out_file.write(("Caught msg type="
                + type
                + " user=" + user
                + " text=" +msg.getText()
                + "¥n").getBytes()
            );
            _out_file.flush();
        }
        catch(Exception e) {

            // if we print the exception from processing an info message,
            // we may recurse indefinitely
            if (msg.getType() != LogMessage.ERROR) {
                // Print some error output to the MobiLink log.
                e.printStackTrace();
            }
        }
    }
}
```

addShutdownListener メソッド

サーバコンテキストが破壊される前に通知を受信する指定の **ShutdownListener** オブジェクトを追加します。

構文

```
void ServerContext.addShutdownListener(ShutdownListener sl)
```

パラメータ

- **sl**　シャットダウン時に通知される ShutdownListener オブジェクト。

備考

シャットダウン時に、**ShutdownListener.shutdownPerformed(ServerContext)** メソッドが呼び出されます。

参照

- ShutdownListener インタフェース [Mobile Link サーバ Java]511 ページ
- ShutdownListener.shutdownPerformed メソッド [Mobile Link サーバ Java]512 ページ

addWarningListener メソッド

警告が出力されたときに通知を受信するために、指定した **LogListener** オブジェクトを追加します。

構文

```
void ServerContext.addWarningListener(LogListener ll)
```

パラメータ

- **ll**　警告が通知される LogListener オブジェクト。

備考

LogListener.messageLogged(ServerContext, LogMessage) メソッドが呼び出されます。

参照

- LogMessage クラス [Mobile Link サーバ Java]492 ページ
- LogListener.messageLogged メソッド [Mobile Link サーバ Java]492 ページ

getProperties メソッド

指定されたコンポーネントとプロパティセットに関連する一連のプロパティを返します。

構文

```
Properties ServerContext.getProperties(String component, String set)
```

パラメータ

- **component**　コンポーネント
- **set**　プロパティセット

戻り値

一連のプロパティ。空の場合があります。

備考

これらのプロパティは、**ml_property** システムテーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

例

次の例では、**ServerContext** インスタンスのすべてのプロパティをリストする方法を示します。

```
import java.util.*;
ServerContext serverContext;
PrintStream out
Properties prop = serverContext.getProperties();
prop.list(out);
```

getPropertiesByVersion メソッド

スクリプトバージョンに関連する一連のプロパティを返します。

構文

`Properties` **`ServerContext.getPropertiesByVersion`**`(String` *`script_version`*`)`

パラメータ

- **script_version**　関連するプロパティを返すスクリプトバージョン

戻り値

指定したスクリプトバージョンに関連する一連のプロパティ

備考

これらは、**ml_property** システムテーブルに格納されます。**component_name** が **ScriptVersion** の場合、スクリプトバージョンは **property_set_name** カラムに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

例

次の例では、指定したスクリプトバージョンに関連付けられた **ServerContext** インスタンスのすべてのプロパティをリストする方法を示します。

```
import java.util.*;
ServerContext serverContext;
PrintStream out
Properties prop = serverContext.getPropertiesByVersion("MyScriptVersion");
prop.list(out);
```

getPropertySetNames メソッド

指定したコンポーネントのプロパティセット名のリストを返します。

構文

`Iterator` **`ServerContext.getPropertySetNames`**`(String` *`component`*`)`

パラメータ

- **component**　プロパティ名をリストするコンポーネントの名前。

戻り値

指定したコンポーネントのプロパティセット名のリスト

備考

これらのプロパティは、**ml_property** システムテーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

例

次の例では、指定したコンポーネントに関連付けられた **ServerContext** インスタンスのすべてのプロパティをリストする方法を示します。

```
import java.util.*;
ServerContext serverContext;
PrintStream out
Properties prop = serverContext.getPropertySetNames("Component Name");
prop.list(out);
```

getStartClassInstances メソッド

サーバ起動時に構築された起動クラスの配列を取得します。

構文

`Object[] ServerContext.getStartClassInstances()`

戻り値

サーバ起動時に構築された起動クラスの配列。起動クラスがない場合は、長さが 0 の配列。

備考

詳細については、「ユーザ定義起動クラス」467 ページを参照してください。

例

次の例は、**getStartClassInstances** メソッドの使用方法を示します。

```
Object objs[] = sc.getStartClassInstances();
int i;
for (i=0; i < objs.length; i += 1) {
    if (objs[i] instanceof MyClass) {
        // Use class.
    }
}
```

makeConnection メソッド

新しいサーバ接続を開いて、返します。

構文

`java.sql.Connection ServerContext.makeConnection() throws SQLException`

戻り値

新しく作成されたサーバ接続。

例外

- **java.sql.SQLException**　新しい接続を開く間にエラーが発生した場合に発行されます。

備考

この接続は、ユーザの Java コードが所有します。ユーザがコミットし、閉じる必要があります。

サーバコンテキストにアクセスするには、現在の接続の **DBConnectionContext** の **DBConnectionContext.getServerContext** メソッドを使用します。

> **注意**
> 接続を開く負荷が大きくなる可能性があります。このメソッドの呼び出し数が最小になるように、ロジックを書いてください。

removeErrorListener メソッド

エラーが出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを削除します。

構文

```
void ServerContext.removeErrorListener(LogListener ll)
```

パラメータ

- **ll**　通知が中止されるリスナオブジェクト。

例

次のコードは、エラーリスナオブジェクトのリストから **LogListener** を削除します。

```
ServerContext serverContext;
LogListener myErrorListener
serverContext.removeErrorListener(myErrorListener);
```

removeInfoListener メソッド

情報が出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを削除します。

構文

```
void ServerContext.removeInfoListener(LogListener ll)
```

パラメータ

- **ll**　通知が中止されるリスナオブジェクト。

例

次のコードは、情報リスナオブジェクトのリストから **LogListener** オブジェクトを削除します。

```
ServerContext serverContext;
LogListener myInfoListener
serverContext.removeInfoListener(myInfoListener);
```

removeShutdownListener メソッド

この **ShutdownListener** が破棄される前に通知を受信するリスナオブジェクトのリストから、指定した **ServerContext** オブジェクトを削除します。

構文

```
void ServerContext.removeShutdownListener(ShutdownListener sl)
```

パラメータ

- **sl**　通知が中止されるリスナオブジェクト。

例

次のコードは、**ServerContext** が破壊される前に通知を受信するリスナオブジェクトのリストから、**ShutdownListener** オブジェクトを削除します。

```
ServerContext serverContext;
ShutdownListener myShutdownListener
serverContext.removeShutdownListener(myShutdownListener);
```

removeWarningListener メソッド

警告が出力されたときに通知を受信するリスナオブジェクトのリストから、指定した **LogListener** オブジェクトを削除します。

構文

```
void ServerContext.removeWarningListener(LogListener ll)
```

パラメータ

- **ll**　通知が中止されるリスナオブジェクト。

例

次のコードは、警告リスナオブジェクトのリストから **LogListener** オブジェクトを削除します。

```
ServerContext serverContext;
LogListener myWarningListener
serverContext.removeWarningListener(myWarningListener);
```

shutdown メソッド

サーバを強制的に停止します。

構文

```
void ServerContext.shutdown()
```

備考

登録されたすべての **ShutdownListener** オブジェクトで、**shutdownPerformed** メソッドが呼び出されます。

参照

- ShutdownListener.shutdownPerformed メソッド [Mobile Link サーバ Java]512 ページ

例

次のコードは、サーバを強制的に停止します。

```
ServerContext serverContext;
serverContext.shutdown();
```

ServerException クラス

サーバで同期の進行を妨げるエラー状態が存在することを示すために発行されます。

構文

`public class` **`ServerException`**

メンバー

継承されたメンバーを含む ServerException クラスのすべてのメンバー。

名前	説明
ServerException コンストラクタ	**ServerException** を構成します。

備考

この例外が発行されると、Mobile Link サーバはシャットダウンされます。

例

次の例では、致命的な問題が発生したときに **ServerException** をスローし、Mobile Link サーバをシャットダウンする方法を示します。

```
public void handleUpload(UploadData ud)
   throws SQLException, IOException, ServerException
{

   UploadedTableData tables[] = ud.getUploadedTables();
   if (tables == null) {
      throw new ServerException("Failed to read uploaded tables");
   }

   for (int i = 0; i < tables.length; i++) {
      UploadedTableData currentTable = tables[i];
      println("table " + java.lang.Integer.toString(i)
         + " name: " + currentTable.getName());

      // Print out delete result set.
      println("Deletes");
```

```
            printRSInfo(currentTable.getDeletes());

            // Print out insert result set.
            println("Inserts");
            printRSInfo(currentTable.getInserts());

            // print out update result set
            println("Updates");
            printUpdateRSInfo(currentTable.getUpdates());
        }
    }
```

ServerException コンストラクタ

ServerException を構成します。

オーバロードリスト

名前	説明
ServerException() コンストラクタ	詳細メッセージのない **ServerException** を構成します。
ServerException(String) コンストラクタ	指定した詳細メッセージを含む **ServerException** を構成します。

ServerException() コンストラクタ

詳細メッセージのない **ServerException** を構成します。

構文

```
ServerException.ServerException()
```

ServerException(String) コンストラクタ

指定した詳細メッセージを含む **ServerException** を構成します。

構文

```
ServerException.ServerException(String s)
```

パラメータ

- **s** 詳細メッセージ。

ShutdownListener インタフェース

サーバのシャットダウンを取得するリスナインタフェースです。

構文

```
public interface ShutdownListener
```

メンバー

継承されたメンバーを含む ShutdownListener インタフェースのすべてのメンバー。

名前	説明
shutdownPerformed メソッド	サーバのシャットダウンによって **ServerContext** が破壊される前に起動されます。

備考

このインタフェースを使用して、Mobile Link サーバが終了する前に、スレッド、接続、およびその他のリソースがすべてクリーンアップされるようにします。

例

次に、**ServerContext** インスタンスの **ShutdownListener** オブジェクトをインストールする方法の例を示します。

```
class MyShutdownListener implements ShutdownListener {
   FileOutputStream _outFile;
   public MyShutdownListener(FileOutputStream outFile) {
      _outFile = outFile;
   }

   public void shutdownPerformed(ServerContext sc) {
      // Add shutdown code
      try {
         _outFile.write(("Shutting Down" + "¥n").getBytes());
         _outFile.flush();
      }
      catch(Exception e) {
         // Print some error output to the MobiLink log.
         e.printStackTrace();
      }
      // ...
   }
}
```

次のコードは、**MyShutdownListener** オブジェクトを登録します。クラスコンストラクタや同期スクリプトなど、**ServerContext** にアクセスできる任意の場所からこのコードを呼び出してください。

```
ServerContext serv_context;
FileOutputStream outFile;
serv_context.addShutdownListener(new MyShutdownListener(outFile));
```

shutdownPerformed メソッド

サーバのシャットダウンによって **ServerContext** が破壊される前に起動されます。

構文

```
void ShutdownListener.shutdownPerformed(ServerContext sc)
```

パラメータ

- **sc**　シャットダウンされるサーバのコンテキスト。

SpatialUtilities クラス

空間値を操作する静的メソッドのコレクション。

構文

```
public class SpatialUtilities
```

メンバー

継承されたメンバーを含む SpatialUtilities クラスのすべてのメンバー。

名前	説明
createSpatialValue メソッド	ダウンロード用にフォーマットされた空間値を含む新しいバイト配列を返します。
getBytes メソッド	指定したバイト配列と同じ空間データを含む新しいバイト配列を返しますが、SRID は削除されています。
getSRID メソッド	指定した空間値の SRID を返します。
setSRID メソッド	指定した SRID を、指定したバイト配列の最初の 4 バイトに格納します。

createSpatialValue メソッド

ダウンロード用にフォーマットされた空間値を含む新しいバイト配列を返します。

構文

```
byte[] SpatialUtilities.createSpatialValue(
    int srid,
    byte[] spatial_value
)
```

パラメータ

- **srid**　SRID。
- **spatial_value**　空間データ。

戻り値

ダウンロード用にフォーマットされた空間値。

備考

最初の 4 バイトにはリトルエンディアンで指定した SRID が含まれ、残りのバイトには指定したバイト配列に渡された空間データが含まれます。

getBytes メソッド

指定したバイト配列と同じ空間データを含む新しいバイト配列を返しますが、SRID は削除されています。

構文

```
byte[] SpatialUtilities.getBytes(byte[] spatial_value)
```

パラメータ

- **spatial_value**　SRID を削除する必要がある空間値。

戻り値

新しいバイト配列

getSRID メソッド

指定した空間値の SRID を返します。

構文

```
int SpatialUtilities.getSRID(byte[] spatial_value)
```

パラメータ

- **spatial_value**　アップロードされた値。最初の 4 バイトには、リトルエンディアンでエンコードされた SRID が含まれている必要があります。

戻り値

SRID。

setSRID メソッド

指定した SRID を、指定したバイト配列の最初の 4 バイトに格納します。

構文

```
void SpatialUtilities.setSRID(byte[] spatial_value, int srid)
```

パラメータ

- **spatial_value**　SRID を格納する配列。

- **srid**　格納する SRID。

SynchronizationException クラス

現在の同期の完了を妨げるエラー状態が存在することを示すために発行されます。

構文

```
public class SynchronizationException
```

メンバー

継承されたメンバーを含む SynchronizationException クラスのすべてのメンバー。

名前	説明
SynchronizationException コンストラクタ	**SynchronizationException** を構成します。

備考

この例外がスローされると、Mobile Link サーバでは現在の同期が強制的にロールバックされ、接続が再初期化されます。

例

次の例では、問題が発生した場合に、**SynchronizationException** をスローすることによって、Mobile Link サーバでロールバックが実行されるようにする方法を示します。

```
public void handleUpload(UploadData ud)
   throws SQLException, IOException, SynchronizationException
{

   UploadedTableData tables[] = ud.getUploadedTables();

   for (int i = 0; i < tables.length; i++) {
      UploadedTableData currentTable = tables[i];
      println("table " + java.lang.Integer.toString(i)
         + " name: " + currentTable.getName());

      // Print out delete result set.
      println("Deletes");
      printRSInfo(currentTable.getDeletes());

      // Print out insert result set.
      println("Inserts");
      printRSInfo(currentTable.getInserts());

      // print out update result set
      println("Updates");
      printUpdateRSInfo(currentTable.getUpdates());

      if (/* Reason for Sync failure /*) {
         throw new SynchronizationException("Sync Failed");
      }
   }
}
```

SynchronizationException コンストラクタ

SynchronizationException を構成します。

オーバロードリスト

名前	説明
SynchronizationException() コンストラクタ	詳細メッセージのない **SynchronizationException** を構成します。
SynchronizationException(String) コンストラクタ	指定した詳細メッセージを含む **SynchronizationException** を構成します。

SynchronizationException() コンストラクタ

詳細メッセージのない **SynchronizationException** を構成します。

構文

```
SynchronizationException.SynchronizationException()
```

SynchronizationException(String) コンストラクタ

指定した詳細メッセージを含む **SynchronizationException** を構成します。

構文

```
SynchronizationException.SynchronizationException(String s)
```

パラメータ

- **s** 詳細メッセージ。

TimestampWithTimeZone クラス

タイムゾーンを取得および設定するメソッドでの **java.sql.Timestamp**。

構文

```
public class TimestampWithTimeZone
```

基本クラス

- Timestamp

メンバー

継承されたメンバを含む TimestampWithTimeZone クラスのすべてのメンバ。

名前	説明
TimestampWithTimeZone コンストラクタ	指定された年、月、日、時間、分、秒、ナノ、タイムゾーンの時間、タイムゾーンの分を含む新しい **TimestampWithTimeZone** を構成します。
equals メソッド	**o** がこのメソッドの Timestamp 部分に等しく、**o** がこのメソッドと同じタイムゾーンオフセットを持つ **TimestampWithTimeZone** であるか、**o** が **TimestampWithTimeZone** ではなくタイムゾーンオフセットが 00:00 である場合は、true を返します。
getTimeZoneOffsetHours メソッド	タイムゾーンオフセットの時間の部分を取得します。
getTimeZoneOffsetMinutes メソッド	タイムゾーンオフセットの分の部分を取得します。
setTimeZoneOffsetHours メソッド	タイムゾーンオフセットの時間の部分を設定します。
setTimeZoneOffsetMinutes メソッド	タイムゾーンオフセットの分の部分を設定します。
toString メソッド	このタイムスタンプを表す文字列を **yyyy-mm-dd hh:mm:ss.ffffff Shh:mm** の形式で返します。ここで、**S** は時間フィールドの符合です。
toTimestampWithTimeZone メソッド	指定された **Timestamp** を **TimestampWithTimeZone** に変換します。
valueOf メソッド	文字列を **TimestampWithTimeZone** 値に変換します。

備考

Mobile Link ダイレクトロー API を使用して、TIMESTAMP WITH TIME ZONE カラムのタイムゾーンオフセットを指定するときに使用します。Mobile Link ダイレクトロー API 以外の JDBC ドライバの **PreparedStatement** オブジェクトと **ResultSet** オブジェクトでは、これが通常の **Timestamp** として処理されます。

TimestampWithTimeZone コンストラクタ

指定された年、月、日、時間、分、秒、ナノ、タイムゾーンの時間、タイムゾーンの分を含む新しい **TimestampWithTimeZone** を構成します。

構文

```
TimestampWithTimeZone.TimestampWithTimeZone(
    int year,
    int month,
    int date,
    int hour,
    int minute,
    int second,
    int nano,
    int tz_hour,
    int tz_minute
)
```

パラメータ

- **year**　年から 1900 を差し引いた値。
- **month**　0 ～ 11 の範囲の整数。
- **date**　1 ～ 31 の範囲の整数。
- **hour**　0 ～ 23 の範囲の整数。
- **minute**　0 ～ 59 の範囲の整数。
- **second**　0 ～ 59 の範囲の整数。
- **nano**　0 ～ 999,999,999 の範囲の整数。
- **tz_hour**　-14 ～ 14 の範囲の整数。
- **tz_minute**　0 ～ 59 の範囲の整数。

例外

- **java.lang.IllegalArgumentException**　**tz_minute** が 0 ～ 59 の範囲にないか、または **tz_hour** が適切な範囲内でない場合にスローされます。

equals メソッド

o がこのメソッドの Timestamp 部分に等しく、o がこのメソッドと同じタイムゾーンオフセットを持つ TimestampWithTimeZone であるか、o が TimestampWithTimeZone ではなくタイムゾーンオフセットが 00:00 である場合は、true を返します。

オーバロードリスト

名前	説明
equals(Object) メソッド	**o** がこのメソッドの Timestamp 部分に等しく、**o** がこのメソッドと同じタイムゾーンオフセットを持つ **TimestampWithTimeZone** であるか、**o** が **TimestampWithTimeZone** ではなくタイムゾーンオフセットが 00:00 である場合は、true を返します。
equals(Timestamp) メソッド	**o** がこのメソッドの Timestamp 部分に等しく、**o** がこのメソッドと同じタイムゾーンオフセットを持つ **TimestampWithTimeZone** であるか、**o** が **TimestampWithTimeZone** ではなくタイムゾーンオフセットが 00:00 である場合は、true を返します。

equals(Object) メソッド

o がこのメソッドの Timestamp 部分に等しく、**o** がこのメソッドと同じタイムゾーンオフセットを持つ **TimestampWithTimeZone** であるか、**o** が **TimestampWithTimeZone** ではなくタイムゾーンオフセットが 00:00 である場合は、true を返します。

構文

```
boolean TimestampWithTimeZone.equals(Object o)
```

パラメータ

- **o**　比較元のオブジェクト。

戻り値

o がこれに等しい場合は true、そうでない場合は false。

参照

- TimestampWithTimeZone.equals メソッド [Mobile Link サーバ Java]518 ページ

equals(Timestamp) メソッド

o がこのメソッドの Timestamp 部分に等しく、**o** がこのメソッドと同じタイムゾーンオフセットを持つ **TimestampWithTimeZone** であるか、**o** が **TimestampWithTimeZone** ではなくタイムゾーンオフセットが 00:00 である場合は、true を返します。

構文

```
boolean TimestampWithTimeZone.equals(Timestamp o)
```

パラメータ

- **o**　比較元のオブジェクト。

戻り値

o がこれに等しい場合は true、そうでない場合は false。

参照

- TimestampWithTimeZone.equals メソッド [Mobile Link サーバ Java]518 ページ

getTimeZoneOffsetHours メソッド

タイムゾーンオフセットの時間の部分を取得します。

構文

```
int TimestampWithTimeZone.getTimeZoneOffsetHours()
```

戻り値

タイムゾーンオフセットの時間の部分。

getTimeZoneOffsetMinutes メソッド

タイムゾーンオフセットの分の部分を取得します。

構文

```
int TimestampWithTimeZone.getTimeZoneOffsetMinutes()
```

戻り値

タイムゾーンオフセットの分の部分。

setTimeZoneOffsetHours メソッド

タイムゾーンオフセットの時間の部分を設定します。

構文

```
void TimestampWithTimeZone.setTimeZoneOffsetHours(int tz_hour)
```

パラメータ

- **tz_hour**　タイムゾーンオフセットの新しい時間の部分。

例外

- **java.lang.IllegalArgumentException**　**tz_hour** が -14:00 ～ +14:00 の範囲にない場合にスローされます。

setTimeZoneOffsetMinutes メソッド

タイムゾーンオフセットの分の部分を設定します。

構文

```
void TimestampWithTimeZone.setTimeZoneOffsetMinutes(int tz_minute)
```

パラメータ

- **tz_minute**　タイムゾーンオフセットの新しい分の部分。

例外

- **java.lang.IllegalArgumentException**　**tz_minute** が 0:00 ～ 59:00 の範囲にない場合にスローされます。

toString メソッド

このタイムスタンプを表す文字列を **yyyy-mm-dd hh:mm:ss.fffffff Shh:mm** の形式で返します。ここで、**S** は時間フィールドの符合です。

構文

```
String TimestampWithTimeZone.toString()
```

戻り値

このタイムスタンプを表す文字列。

toTimestampWithTimeZone メソッド

指定された **Timestamp** を **TimestampWithTimeZone** に変換します。

構文

```
TimestampWithTimeZone TimestampWithTimeZone.toTimestampWithTimeZone(
    Timestamp ts
)
```

パラメータ

- **ts**　変換するタイムスタンプ。

戻り値

ts が **TimestampWithTimeZone** のインスタンスの場合は、**ts**。それ以外の場合は、タイムゾーンオフセットが 00:00 である **ts** と同等の新しい **TimestampWithTimeZone** を構築します。

valueOf メソッド

文字列を **TimestampWithTimeZone** 値に変換します。

構文

```
TimestampWithTimeZone TimestampWithTimeZone.valueOf(String val)
```

パラメータ

- **val**　**yyyy-mm-dd hh:mm:ss[.f...][[-+]hh:mm]** 形式によるタイムゾーンでのタイムスタンプ。小数点部分とタイムゾーンの部分は省略できます。タイムゾーンを指定する場合、符号は省略できます。

戻り値

新しい TimestampWithTimeZone。

例外

- **java.lang.IllegalArgumentException**　**val** が正しいフォーマットでない場合。

UpdateResultSet インタフェース

指定した行の更新前イメージ値 (古い値) と更新後イメージ値 (新しい値) にアクセスするための特別なメソッドを含む結果セットオブジェクトです。

構文

```
public interface UpdateResultSet
```

メンバー

継承されたメンバーを含む UpdateResultSet interface インタフェースのすべてのメンバー。

名前	説明
setNewRowValues メソッド	新しいカラム値 (更新後のロー) を返すように、この結果セットのモードを設定します。
setOldRowValues メソッド	古いカラム値 (更新前のロー) を返すように、この結果セットのモードを設定します。

備考

1 つのテーブルの 1 つのアップロードトランザクションのための更新操作を保持できます。

ResultSet のモードを変更することで、新しいローにも古いローにもアクセスできます。

UpdateResultSet のインスタンスを取得するには、**DownloadTableData.getUpdates** メソッドを使用します。

UpdateResultSet は **java.sql.ResultSet** を拡張し、**setNewRowValues** メソッドと **setOldRowValues** メソッドを追加します。それ以外の場合は、通常の **ResultSet** と同様に使用できます。

java.sql.ResultSet の詳細については、Java Software Development Kit マニュアルを参照してください。

ダイレクトアップロードでの競合の処理の詳細については、「ダイレクトアップロードでの競合」629 ページを参照してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

参照

- UploadedTableData.getUpdates メソッド [Mobile Link サーバ Java]531 ページ

setNewRowValues メソッド

新しいカラム値 (更新後のロー) を返すように、この結果セットのモードを設定します。

構文

```
void UpdateResultSet.setNewRowValues()
```

備考

結果セットは、リモートクライアントデータベース内の最新の更新値を表します。

これがデフォルトモードです。

ダイレクトアップロードでの競合の処理の詳細については、「ダイレクトアップロードでの競合」629 ページを参照してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

例

次のコードは、新しいカラム値を返すように、**UpdateResultSet** のモードを設定する方法を示します。

```
UpdateResultSet results
results.setNewRowValues();
```

setOldRowValues メソッド

古いカラム値 (更新前のロー) を返すように、この結果セットのモードを設定します。

構文

```
void UpdateResultSet.setOldRowValues()
```

備考

このモードでは、**UpdateResultSet** は、最後の同期中にクライアントが取得した古いカラム値を表します。

ダイレクトアップロードでの競合の処理の詳細については、「ダイレクトアップロードでの競合」629 ページを参照してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

例

次のコードは、古いカラム値を返すように、**UpdateResultSet** のモードを設定する方法を示します。

```
UpdateResultSet results
results.setOldRowValues();
```

UploadData インタフェース

ダイレクトローハンドリングで使用するアップロード操作をカプセル化します。

構文

```
public interface UploadData
```

メンバー

継承されたメンバーを含む UploadData インタフェースのすべてのメンバー。

名前	説明
getUploadedTableByName メソッド	指定されたテーブルを表す **UploadedTableData** インスタンスを返します。
getUploadedTables メソッド	現在の同期の **UploadedTableData** オブジェクトの配列を返します。

備考

単一のアップロードトランザクションを表す **UploadData** インスタンスが、handle_UploadData イベントに渡されます。

> **注意**
> ダイレクトローハンドリングのアップロード操作は、handle_UploadData イベントに対して登録したメソッドで処理してください。登録されたメソッドを呼び出した後、**UploadData** は破棄されます。後続のイベントで使用するために新しい **UploadData** インスタンスを作成しないでください。

UploadedTableData のインスタンスを取得するには、**getUploadedTables** メソッドまたは **getUploadedTableByName** メソッドを使用します。

リモートデータベースがトランザクションアップロードを使用している場合を除き、同期の **UploadData** は 1 つです。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

ダイレクトアップロードでの競合の処理の詳細については、「ダイレクトアップロードでの競合」629 ページを参照してください。

handle_UploadData 接続イベントの詳細については、「handle_UploadData 接続イベント」383 ページを参照してください。

参照

- UploadedTableData インタフェース [Mobile Link サーバ Java]526 ページ

getUploadedTableByName メソッド

指定されたテーブルを表す **UploadedTableData** インスタンスを返します。

構文

```
UploadedTableData UploadData.getUploadedTableByName(String table_name)
```

パラメータ

- **table_name**　アップロードデータの取得先アップロードテーブルの名前

戻り値

指定されたテーブルを表す **UploadedTableData** インスタンス。指定された名前のテーブルが現在の同期で存在しない場合は NULL。

参照

- UploadedTableData インタフェース [Mobile Link サーバ Java]526 ページ

例

handle_UploadData イベントに対して **HandleUpload** というメソッドを使用するものとします。次の例では、**GetUploadedTableByName** メソッドを使用して、**remoteOrderstable** の **UploadedTableData** インスタンスを返します。

```
public void handleUpload(UploadData ud)
   throws SQLException, IOException
{
   UploadedTableData uploaded_t1 = ud.GetUploadedTableByName("remoteOrders");
   //...
}
```

getUploadedTables メソッド

現在の同期の **UploadedTableData** オブジェクトの配列を返します。

構文

```
UploadedTableData[] UploadData.getUploadedTables()
```

戻り値

現在の同期の **UploadedTableData** オブジェクトの配列。配列内でのテーブルの順序は、クライアントのアップロード順と同じです。

備考

テーブルの配列内での順序は、Mobile Link による SQL のローハンドリングでの順序と同じで、参照整合性違反を防ぐ最適な順序になります。データソースがリレーショナルデータベースの場合は、このテーブル順序を使用してください。

参照

- UploadedTableData インタフェース [Mobile Link サーバ Java]526 ページ

例

handle_UploadData イベントに対して **HandleUpload** というメソッドを使用するものとします。次の例では、**getUploadedTables** メソッドを使用して、現在のアップロードトランザクションの **UploadedTableData** インスタンスを返します。

```
public void handleUpload(UploadData ud)
   throws SQLException, IOException
{
   UploadedTableData tables[] = ud.getUploadedTables();
   //...
}
```

UploadedTableData インタフェース

ダイレクトローハンドリングアップロードで使用するテーブル操作をカプセル化します。

構文

```
public interface UploadedTableData
```

メンバー

継承されたメンバーを含む UploadedTableData インタフェースのすべてのメンバー。

名前	説明
getDeletes メソッド	Mobile Link クライアントによってアップロードされた削除操作を表す **java.sql.ResultSet** オブジェクトを返します。
getInserts メソッド	Mobile Link クライアントによってアップロードされた挿入操作を表す **java.sql.ResultSet** オブジェクトを返します。
getMetaData メソッド	**UploadedTableData** インスタンスのメタデータを取得します。

名前	説明
getName メソッド	**UploadedTableData** インスタンスのテーブル名を返します。
getUpdates メソッド	Mobile Link クライアントによってアップロードされた更新操作を表す **UpdateResultSet** オブジェクトを返します。

備考

UploadedTableData インスタンスを使用して、単一アップロードトランザクションに対するテーブルの挿入、更新、削除操作を取得できます。**getInserts**、**getUpdates**、および **getDeletes** メソッドを使用して、標準の JDBC **java.sql.ResultSet** オブジェクトを返します。

java.sql.ResultSet と **java.sql.ResultSetMetaData** の詳細については、Java Software Development Kit マニュアルを参照してください。

テーブルメタデータには、**getMetaData** メソッドまたは **getInserts**、**getUpdates**、および **getDeletes** によって返された結果メソッドを使用してアクセスできます。削除の結果セットには、テーブルのプライマリキーカラムのみが含まれています。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

handle_UploadData 接続イベントの詳細については、「handle_UploadData 接続イベント」383 ページを参照してください。

参照

- UploadData インタフェース [Mobile Link サーバ Java]524 ページ

例

次のコードでは、アップロードされた削除を取得し、それぞれの最初のカラムを表示します。

```
void printFirstColOfDeletes( UploadedTableData tab_data )
{
   ResultSet deletes = tab_data.getDeletes();
   while( deletes.next() ) {
      java.lang.System.out.println( deletes.getString( 1 ) );
   }
   deletes.close();
}
```

次のコードでは、それぞれの更新の 3 番目のカラムの古い値と新しい値を表示します。:

```
void printThirdColOfUpdates( UploadedTableData tab_data )
{
   ResultSet updates = tab_data.getUpdates();
   while( updates.next() ) {
      updates.setOldRowValues();
      java.lang.System.out.println( "old row col: " + updates.getString( 3 ) );
      updates.setNewRowValues();
      java.lang.System.out.println( "new row col: " + updates.getString( 3 ) );
```

```
    }
    updates.close();
}
```

getDeletes メソッド

Mobile Link クライアントによってアップロードされた削除操作を表す **java.sql.ResultSet** オブジェクトを返します。

構文

`java.sql.ResultSet` **`UploadedTableData.getDeletes`**`()`

戻り値

Mobile Link クライアントによってアップロードされた削除操作を表す java.sql.ResultSet オブジェクト

備考

結果セットには、削除されたローのプライマリキー値が含まれています。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

handle_UploadData 接続イベントの詳細については、「handle_UploadData 接続イベント」383 ページを参照してください。

例

リモートクライアントには **remoteOrders** というテーブルがあるものとします。次の例は **DownloadTableData.getDeletes** メソッドを使用して、削除されたローの結果セットを取得します。この場合、削除結果セットには 1 つのプライマリキーカラムが含まれています。

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData for the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // Get deletes uploaded by the MobiLink client.
    java.sql.ResultSet delete_rs = remoteOrdersTable.getDeletes();

     while (delete_rs.next()) {
        // Get primary key values for deleted rows.
        int deleted_id = delete_rs.getInt(1);

        // ...
    }
    delete_rs.close();
}
```

getInserts メソッド

Mobile Link クライアントによってアップロードされた挿入操作を表す **java.sql.ResultSet** オブジェクトを返します。

構文

`java.sql.ResultSet` **`UploadedTableData.getInserts`**`()`

戻り値

Mobile Link クライアントによってアップロードされた挿入操作を表す **java.sql.ResultSet** オブジェクト

備考

各挿入は結果セットの 1 つのローで表されています。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

例

リモートクライアントには **remoteOrders** というテーブルがあるものとします。次の例は **DownloadTableData.getInserts** メソッドを使用して、挿入されたローの結果セットを取得します。このコードは、現在のアップロードトランザクションの各ローに対する発注額を取得します。

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // Get inserts uploaded by the MobiLink client.
    java.sql.ResultSet rs = remoteOrdersTable.getInserts();
    while (rs.next()) {
        // get the uploaded order_amount
        double order_amount = rs.getDouble("order_amount");

        // ...
    }
    rs.close();
}
```

getMetaData メソッド

UploadedTableData インスタンスのメタデータを取得します。

構文

`java.sql.ResultSetMetaData` **`UploadedTableData.getMetaData`**`()`

戻り値

UploadedTableData インスタンスのメタデータ

備考

メタデータは標準 **java.sql.ResultSetMetaData** インスタンスです。

ResultSetMetaData にカラム名情報を含める場合は、カラム名を送信するためのクライアント拡張オプションを指定してください。

java.sql.ResultSetMetaData の詳細については、Java Software Development Kit マニュアルを参照してください。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

handle_UploadData 接続イベントの詳細については、「handle_UploadData 接続イベント」383 ページを参照してください。

例

次の例では、**remoteOrders** という更新テーブルの **java.sql.ResultSetMetaData** インスタンスを取得しています。コードでは、**ResultSetMetaData.getColumnCount** メソッドと **getColumnLabel** メソッドを使用して、カラム名のリストをコンパイルしています。

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // get inserts uploaded by the MobiLink client
    java.sql.ResultSet rs = remoteOrdersTable.getInserts();

    // Obtain the result set metadata.
    java.sql.ResultSetMetaData md = rs.getMetaData();
    String columnHeading = "";

    // Compile a list of column names.
    for (int c=1; c <= md.getColumnCount(); c += 1) {
        columnHeading += md.getColumnLabel( c );

        if (c < md.getColumnCount()) {
            columnHeading += ", ";
        }
    }
    //...
}
```

この場合、**HandleUpload** というメソッドが handle_UploadData 同期イベントを処理します。

getName メソッド

UploadedTableData インスタンスのテーブル名を返します。

構文

```
String UploadedTableData.getName()
```

戻り値

UploadedTableData インスタンスのテーブル名

備考

java.sql.ResultSetMetaData メソッドによって返される **getMetaData** インスタンスを使用することによっても、テーブル名にアクセスできます。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

handle_UploadData 接続イベントの詳細については、「handle_UploadData 接続イベント」383 ページを参照してください。

参照

- UploadedTableData.getMetaData メソッド [Mobile Link サーバ Java]529 ページ

例

次の例は、単一アップロードトランザクションのアップロードされた各テーブルの名前を取得します。

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ud) {
   throws SQLException, IOException
{
   int i;

   // Get UploadedTableData instances.
   UploadedTableData tables[] = ud.getUploadedTables();

   for (i=0; i<tables.length; i+=1) {
      // Get the table name.
      String table_name = tables[i].getName();

      // ...
   }
}
```

getUpdates メソッド

Mobile Link クライアントによってアップロードされた更新操作を表す **UpdateResultSet** オブジェクトを返します。

構文

```
UpdateResultSet UploadedTableData.getUpdates()
```

戻り値

Mobile Link クライアントによってアップロードされた更新操作を表す **UpdateResultSet** オブジェクト

備考

各更新は、すべてのカラム値を含む 1 つのローで表されています。**UpdateResultSet** は **java.sql.ResultSet** を拡張して、Mobile Link での競合検出用の特別なメソッドを追加しています。

ダイレクトローハンドリングの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

ダイレクトアップロードでの競合の処理の詳細については、「ダイレクトアップロードでの競合」629 ページを参照してください。

handle_UploadData 接続イベントの詳細については、「handle_UploadData 接続イベント」383 ページを参照してください。

参照

- UpdateResultSet インタフェース [Mobile Link サーバ Java]522 ページ

例

リモートクライアントには **remoteOrders** というテーブルがあるものとします。次の例は **getUpdates** メソッドを使用して、更新されたローの結果セットを取得します。このコードは各ローの発注額を取得します。

```
import ianywhere.ml.script.*;
import java.sql.*;

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // Get inserts uploaded by the MobiLink client.
    java.sql.ResultSet rs = remoteOrdersTable.getUpdates();
    while (rs.next()) {
        // Get the uploaded order_amount.
        double order_amount = rs.getDouble("order_amount");

        // ...
    }
    rs.close();
}
```

.NET の同期スクリプト

Mobile Link は、同期スクリプトを作成するためのプログラミング言語として Visual Studio をサポートしています。Mobile Link スクリプトを .NET で作成する場合は、有効な .NET アセンブリを作成できる言語であれば、どれでも使用できます。特に、以下の言語がテスト済みで、文書化されています。

- C#
- Visual Basic .NET
- C++

.NET の同期論理は、SQL 論理と同様に機能します。Mobile Link サーバは、Mobile Link イベントの発生時に SQL スクリプトにアクセスするのと同様に、.NET メソッドを呼び出すことができます。.NET メソッドは、SQL 文字列を Mobile Link に返すことができます。

この項では、C#、Visual Basic .NET、C++ 用の .NET 同期論理を設定、開発、実行する方法について説明します。また、サンプルアプリケーションと .NET 用 Mobile Link サーバ API リファレンスも含まれています。

参照

- 「サーバ側の同期論理の作成オプション」『Mobile Link クイックスタート』
- 「同期スクリプトの作成」229 ページ

.NET への同期スクリプトの実装

同期スクリプトを .NET で実装するときには、アセンブリに含まれるパッケージ、クラス、メソッドの場所を Mobile Link に指示する必要があります。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. 1 つまたは複数の独自クラスを作成します。必要な同期イベントごとに、メソッドを作成します。これらのメソッドは、パブリックにしてください。

   メソッドの詳細については、「メソッド」538 ページを参照してください。

   非静的メソッドを持つ各クラスには、パブリックコンストラクタが必要です。Mobile Link サーバは、各クラスのメソッドが接続のために初めて呼び出されるときに、そのクラスを自動的にインスタンス化します。

   「コンストラクタ」537 ページを参照してください。

2. 1 つまたは複数のアセンブリを作成します。コンパイル中、独自の .NET メソッドで利用するために、Mobile Link サーバ API クラスのレポジトリを含む *iAnywhere.MobiLink.Script.dll* を参照します。*iAnywhere.MobiLink.Script.dll* は、*%SQLANY16%¥Assembly¥V2* にあります。

   クラスをコンパイルするには、コマンドラインを使用するか、Visual Studio や他の .NET 開発環境を使用することができます。

   「Mobile Link サーバ .NET API リファレンス」548 ページを参照してください。

3. プロジェクトをコンパイルします。

   たとえば、次のようにして Visual Studio からコンパイルします。

   a. VS.NET の **[プロジェクト]** » **[既存項目の追加]** をクリックします。

   b. *iAnywhere.MobiLink.Script.dll* を探します。

      [開く] リストで **[リンクファイル]** をクリックします。

      > **注意**
      > Visual Studio の場合は、常に上記の [リンク ファイル] を使用します。*iAnywhere.MobiLink.Script.dll* を参照するために、[参照の追加] オプションを使用しないでください。このオプションは、クラスアセンブリとして同じ物理ディレクトリに *iAnywhere.MobiLink.Script.dll* を複製するため、Mobile Link サーバにとって問題となります。

   c. アセンブリを構築するには、**[ビルド]** メニューを使用します。

   次のように入力して、コマンドラインからコンパイルすることもできます。

   dll-path を *iAnywhere.MobiLink.Script.dll* へのパスに置き換えます。たとえば、C# の場合は次のようになります。

   ```
   csc /out:dll-pathout.dll /target:library /reference:dll-pathiAnywhere.MobiLink.Script.dll sync_v1.cs
   ```

4. 統合データベース内の Mobile Link システムテーブルで、各同期スクリプトについて、呼び出すパッケージ、クラス、メソッドの名前を指定します。スクリプトの各バージョンにつき使用できるクラスは 1 つだけです。

   たとえば、ml_add_dnet_connection_script ストアドプロシージャまたは ml_add_dnet_table_script ストアドプロシージャを使用して、この情報を Mobile Link システムテーブルに追加できます。次の SQL 文は、SQL Anywhere データベース内で実行すると、authenticate_user 接続レベルイベントが発生するたびに myNamespace.myClass.myMethod を実行するように指定します。

   ```
   CALL ml_add_dnet_connection_script(
      'version1',
      'authenticate_user',
      'myNamespace.myClass.myMethod'
   )
   ```

   > **注意**
   > 完全に修飾されたメソッド名では、大文字と小文字が区別されます。

このプロシージャコールの結果として、ml_script システムテーブルの script_language カラムに、**dnet** という語が含まれます。script カラムにはパブリックな .NET メソッドの修飾名が含まれます。

「ml_add_dnet_connection_script システムプロシージャ」649 ページと「ml_add_dnet_table_script システムプロシージャ」650 ページを参照してください。

この情報を追加するには Sybase Central を使用する方法もあります。

「スクリプトの追加と削除」256 ページを参照してください。

5. アセンブリをロードするよう Mobile Link サーバに指示し、CLR を起動します。mlsrv16 コマンドラインでオプションを使用して、これらのアセンブリの場所を Mobile Link に指示します。2 つのオプションから選択できます。

   - **-sl dnet (-MLAutoLoadPath) を使用**　このオプションは、アプリケーションのベースディレクトリへの特定のパスを設定し、そのディレクトリ内のすべてのプライベートアセンブリをロードします。通常、この方法を推奨します。たとえば、*dll-path* に保存されたすべてのアセンブリをロードするには、次を入力します。

     ```
     mlsrv16 -c "DSN=consolidated1" -sl dnet(-MLAutoLoadPath=dll-path)
     ```

     -MLAutoLoadPath オプションを使用すると、イベントスクリプトの完全に修飾されたメソッド名を入力するときに、ドメインを指定できません。

     「.NET アセンブリのロード」544 ページと「-sl dnet mlsrv16 オプション」76 ページを参照してください。

   - **-sl dnet (-MLDomConfigFile) を使用**　このオプションを使用するには、ドメインとアセンブリの設定を含む設定ファイルが必要です。このオプションを使用するのは、共有アセンブリがある場合、ディレクトリのすべてのアセンブリをロードする必要がない場合、またはその他の理由で設定ファイルを使用する必要がある場合です。

     共有アセンブリのロードの詳細については、「.NET アセンブリのロード」544 ページを参照してください。

     mlsrv16 オプションの -sl dnet の詳細については、「-sl dnet mlsrv16 オプション」76 ページを参照してください。

     注意
     -MLAutoLoadPath オプションまたは -MLDomConfigFile オプションを使用できますが、両方は使用できません。

結果

.NET 同期論理が設定されます。

.NET 同期論理

.NET 同期論理を作成するには、Mobile Link イベントの知識、.NET に関する若干の知識、.NET 用 Mobile Link サーバ API に関する基本的な知識が必要です。

API の完全な説明については、「Mobile Link サーバ .NET API リファレンス」548 ページを参照してください。

.NET 同期論理は、ステータス情報の管理と、アップロードイベントとダウンロードイベント関連の論理の実装に使用できます。たとえば、.NET で作成された begin_synchronization スクリプトを使用すると、Mobile Link ユーザ名を変数に格納できます。同期処理中に後で呼び出されるスクリプトは、この変数にアクセスできます。また、.NET は、コミットの実行前または実行後に統合データベースのローにアクセスするために使用できます。

.NET を使用すると、統合データベースへの依存度も減少します。また、統合データベースを新バージョンにアップグレードしたり、別のデータベース管理システムに切り替えたりする場合も、動作に与える影響が少なくて済みます。

ダイレクトローハンドリング

Mobile Link のダイレクトローハンドリングを使用して、リモートデータと中央のデータソース、アプリケーション、または Web サービスとの通信ができます。ダイレクトローハンドリングでは、Java または .NET 用 Mobile Link サーバ API の特別なクラスを使用して、同期対象のデータに直接アクセスします。

「ダイレクトローハンドリング」623 ページを参照してください。

クラスインスタンス

Mobile Link サーバは、クラスをデータベース接続レベルでインスタンス化します。非静的 .NET メソッドをあるイベントに対して作成した場合、そのイベントに達すると、現在のデータベース接続でクラスがインスタンス化されていなければ、Mobile Link サーバが自動的にクラスをインスタンス化します。

「コンストラクタ」537 ページを参照してください。

注意
1 つのスクリプトバージョンの接続レベルイベントまたはテーブルレベルイベントに直接関連するすべてのメソッドは、同じクラスに属している必要があります。

データベース接続ごとに、インスタンス化されたクラスはその接続が終了するまで持続します。したがって、連続する複数の同期セッションで、同じインスタンスを使用することができます。明示的にクリアされないかぎり、パブリック変数またはプライベート変数内の情報は、同じ接続で発生するすべての同期を通して持続されます。

また、静的なクラスや変数も使用できます。この場合、値は同じドメインのすべての接続を通して使用できます。

統合データベースへの接続が終了した場合にのみ、Mobile Link サーバはクラスインスタンスを自動的に削除します。

トランザクション

.NET メソッドには、トランザクションに関する通常のルールが適用されます。データベーストランザクションの開始と継続期間は、同期処理に重要になります。トランザクションの開始と終了は、Mobile Link サーバのみが行います。.NET メソッド内の同期接続でトランザクションを明示的にコミットまたはロールバックすると、同期処理の整合性違反になり、エラーが発生することがあります。

これらのルールは、Mobile Link サーバによって作成されるデータベース接続、特に、メソッドから返される SQL 文にのみ適用されます。

SQL データ型と .NET データ型

次の表は、Mobile Link スクリプトパラメータの SQL データ型とそれに対応する .NET データ型を示します。

SQL データ型	.NET データ型
VARCHAR	string
CHAR	string
INTEGER	int
BIGINT	long
BINARY	byte [ ]
TIMESTAMP	DateTime
INOUT INTEGER	ref int
INOUT VARCHAR	ref string
INOUT CHAR	ref string
INOUT BYTEARRAY	ref byte [ ]
INOUT TIMESTAMP	ref DateTime

コンストラクタ

クラスのコンストラクタはパラメータなしにするか、1 つの iAnywhere.MobiLink.Script.DBConnectionContext をパラメータにできます。次に例を示します。

```
public ExampleClass(iAnywhere.MobiLink.Script.DBConnectionContext cc)
```

または

```
public ExampleClass()
```

渡される同期コンテキストは、Mobile Link サーバが現在のユーザの同期に使用している接続です。

DBConnectionContext.GetConnection メソッドは、Mobile Link が現在のユーザの同期に使用しているのと同じデータベース接続を返します。この接続で文を実行することはできますが、トランザクションのコミットやロールバックは行わないでください。トランザクションは Mobile Link サーバによって管理されます。

Mobile Link サーバでは、コンストラクタがある場合、iAnywhere.MobiLink.Script.DBConnectionContext パラメータを持つコンストラクタが使用されます。コンストラクタがない場合は、void コンストラクタが使用されます。

DBConnectionContext インタフェース [Mobile Link サーバ .NET]561 ページを参照してください。

メソッド

通常は、同期イベントごとにメソッドを 1 つずつ実装します。これらのメソッドは、パブリックにしてください。プライベートメソッドの場合、Mobile Link サーバでは使用できず、その存在を認識できません。

統合データベース内の ml_script テーブルで指定されている名前と一致していれば、メソッド名は重要ではありません。ただし、このマニュアルの例では、メソッド名は Mobile Link イベント名と同じです。これは、.NET コードを読みやすくするためです。

メソッドのシグネチャは、そのイベント用スクリプトのシグネチャと一致していなければなりません。ただし、パラメータリストの最後にパラメータ値が必要でない場合は、リストをトランケートできます。パラメータを渡すとオーバヘッドが生じる可能性があるため、必要なパラメータのみを受け入れてください。

ただし、メソッドはオーバロードできません。ml_script システムテーブルには、クラスごとにメソッドプロトタイプが 1 つだけ格納されます。

値の登録

SQL ベースのアップロードまたはダウンロードに対して呼び出されるメソッドは、有効な SQL 言語の文を返さなければなりません。これらのメソッドの戻り値は String 型にします。他の戻り値の型は使用できません。

他のすべてのスクリプトの戻り値の型は、string または void にします。他の型は使用できません。戻り値の型が NULL ではなく文字列の場合、Mobile Link サーバはその文字列に有効な SQL 文が含まれていると見なして、この文を通常の SQL 言語による同期スクリプトと同様に統合データベース内で実行します。通常、メソッドは文字列を返しますが、その戻り値によってデータベースに対して SQL 文を実行しない場合は null を返すことができます。

ユーザ定義起動クラス

サーバの起動時に自動的にロードされる起動クラスを定義できます。この機能の目的は、最初の同期の前に Mobile Link サーバが CLR を起動する時点で実行される .NET コードを記述できるようにすることです。つまり、サーバインスタンスで、最初のユーザ同期要求の前に、接続の作成またはデータのキャッシュを実行できます。

この操作を行うには、mlsrv16 -sl dnet オプションの MLStartClasses オプションを使用します。たとえば、次に示すのは mlsrv16 コマンドラインの一部です。mycl1 と mycl2 が起動クラスとしてロードされます。

```
-sl dnet(-MLStartClasses=MyNameSpace.MyClass.mycl1,MyNameSpace.MyClass.mycl2)
```

クラスはリスト内の順序でロードされます。同じクラスがリストに 2 回以上指定されている場合は、複数のインスタンスが作成されます。

すべての起動クラスはパブリックでなければなりません。また、引数を 1 つも受け付けないか、または MobiLink.Script.ServerContext データ型の引数を 1 つ受け付けるパブリックコンストラクタが必要です。

ロードされた起動クラスの名前は、「.NET 起動クラス *classname* がロードされました。」というメッセージとともに Mobile Link ログに出力されます。

.NET CLR の詳細については、「-sl dnet mlsrv16 オプション」76 ページを参照してください。

サーバ起動時に構築される起動クラスを表示する方法については、ServerContext.GetStartClassInstances メソッド [Mobile Link サーバ .NET]602 ページを参照してください。

例

次に示すのは、起動クラスのテンプレートです。これは、イベントを処理してデータベース接続を確立するデーモンスレッドを起動します (すべての起動クラスがスレッドの作成を必要とするわけではありませんが、スレッドを作成する場合はデーモンスレッドでなければなりません)。

```
using System;
using System.IO;
using System.Threading;
using iAnywhere.MobiLink.Script;

namespace TestScripts {
   public class MyStartClass {
      ServerContext    _sc;
      bool             _exit_loop;
      Thread           _thread;
      OdbcConnection   _conn;

      public MyStartClass(ServerContext sc) {

         // Perform setup first so that an exception
         // causes MobiLink startup to fail.
         _sc = sc;

         // Create connection for use later.
         _conn = _sc.makeConnection();
         _exit_loop = false;
```

```
            _thread = new Thread(new ThreadStart(run)) ;
            _thread.IsBackground = true;
            _thread.Start();
        }

        public void run() {
            ShutdownCallback callback = new ShutdownCallback(shutdownPerformed);
            _sc.ShutdownListener += callback;

            // run() can't throw exceptions.
            try {
                handlerLoop();
                _conn.close();
                _conn = null;
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                Console.Error.Write(e.ToString());

                // There is no need to be notified of shutdown.
                _sc.ShutdownListener -= callback;

                // Ask server to shut down so this fatal error can be fixed.
                _sc.Shutdown();
            }

            // Shortly after return, this thread no longer exists.
            return;
        }

        public void shutdownPerformed(ServerContext sc) {
            // Stop the event handler loop.
            try {
                _exit_loop = true;

                // Wait a maximum of 10 seconds for thread to die.
                _thread.Join(10*1000);
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                Console.Error.Write(e.ToString());
            }
        }

        private void handlerLoop() {
            while (!_exit_loop) {
                // Handle events in this loop.
                Thread.Sleep(1*1000);
            }
        }
    }
}
```

.NET から情報を印刷する方法

System.Console を使用して、Mobile Link ログに情報を出力する文を .NET メソッドに追加することもできます。これにより、クラスの進行状況と動作を追跡できます。

> **注意**
> この方法で Mobile Link のログに情報を出力すると、モニタツールとして活用できますが、運用環境ではおすすめしません。

これと同じ方法を利用して、任意の同期情報のログを取ったり、スクリプトの使用方法に関する統計情報を収集したりできます。

.NET での Mobile Link サーバエラーの処理

ログをスキャンするだけでは不十分な場合は、プログラムによってアプリケーションをモニタリングできます。たとえば、特定のタイプのメッセージを電子メールで送信できます。

ログに出力される各エラーメッセージまたは各警告メッセージを表すクラスに渡されるメソッドを作成することも可能です。この方法は、Mobile Link サーバをモニタおよび監査するのに役立ちます。

次のコードは、すべてのエラーメッセージ用にリスナをインストールし、その情報を StreamWriter に出力します。

```
class TestLogListener {
   public TestLogListener(StreamWriter output_file) {
      _output_file    = output_file;
   }

   public void errCallback(ServerContext sc, LogMessage lm) {
      string type;
      string user;

      if (lm.Type == LogMessage.MessageType.ERROR) {
         type = "ERROR";
      } else if (lm.Type==LogMessage.MessageType.WARNING) {
         type = "WARNING";
      }
      else {
         type = "INVALID TYPE!!";
      }
      if (lm.User == null) {
         user = "null";
      }
      else {
         user = lm.User;
      }

      _output_file.WriteLine("Caught msg type=" + type
         + " user=" + user
         + " text=" + lm.Text);
      _output_file.Flush();
    }
    StreamWriter _output_file;
}
```

次のコードは、TestLogListener を登録します。クラスコンストラクタや同期スクリプトなど、ServerContext にアクセスできる場所からこのコードを呼び出してください。

```
// ServerContext serv_context;
TestLogListener errtll = new TestLogListener(log_listener_file);
serv_context.ErrorListener += new LogCallback(errtll.errCallback);
```

参照

- LogCallback デリゲート [Mobile Link サーバ .NET]619 ページ
- ServerContext.ErrorListener イベント [Mobile Link サーバ .NET]603 ページ
- ServerContext.WarningListener イベント [Mobile Link サーバ .NET]604 ページ
- LogMessage クラス [Mobile Link サーバ .NET]590 ページ
- LogMessage.MessageType 列挙体 [Mobile Link サーバ .NET]591 ページ

.NET 同期論理のデバッグ

Visual Studio を使用して .NET スクリプトをデバッグするには、次の方法を使用できます。

前提条件

この作業を実行するための前提条件はありません。

♦ タスク

1. Visual Studio を起動します。
2. **[ツール]** » **[プロセスにアタッチ]** を選択します。
3. **[選択可能なプロセス]** コントロールから **mlsrv16.exe** を選択し、**[アタッチ]** を押します。
4. ブレークポイントを設定します。
5. 同期を開始します。

結果

スクリプトはデバッグ可能です。

Visual Studio を使用して .NET スクリプトをデバッグするには、次の方法を使用できます。

前提条件

この作業を実行するための前提条件はありません。

♦ .NET スクリプトのデバッグ

1. 次のいずれかの方法を使用して、デバッグ情報をオンにした状態でコードをコンパイルします。
   - csc コマンドラインで、/debug+ オプションを設定します。
   - Microsoft Visual Studio の設定を使用してデバッグ出力を設定します。
     - **[ビルド]** » **[構成マネージャ]** をクリックします。
       [アクティブソリューション構成] リストで **[デバッグ]** をクリックします。
     - アセンブリを構築します。

2. ソースファイルを含む Visual Studio の実行中のインスタンスを閉じます。

3. 次の手順では、新しい Visual Studio インスタンスを起動して、Mobile Link サーバと使用している .NET 同期スクリプトをデバッグします。コマンドラインオプションを使用して Visual Studio を起動し、Mobile Link サーバをデバッグします。

   ●コマンドプロンプトで、Visual Studio インストール環境の *Common7¥IDE* サブフォルダに移動します。

   ●/debugexe オプションを使用して、devenv (Visual Studio IDE) を起動します。

   たとえば、次のコマンドを実行して、Mobile Link サーバをデバッグします。接続文字列と .NET アセンブリをロードするオプションを含めて、mlsrv16 オプションを指定してください。

   32 ビット Windows 環境の場合：

   ```
   devenv /debugexe %sqlany16%¥bin32¥mlsrv16.exe -c ...
   ```

   64 ビット Windows 環境の場合：

   ```
   devenv /debugexe %sqlany16%¥bin64¥mlsrv16.exe -c ...
   ```

   Visual Studio が起動し、[ソリューション エクスプローラー] ウィンドウに *mlsrv16.exe* が表示されます。

4. .NET コードをデバッグするために Microsoft Visual Studio を次のように設定します。

   ●Visual Studio の [ソリューション エクスプローラー] ウィンドウで *mlsrv16.exe* を右クリックし、[プロパティ] を選択します。

   ●[デバッガーの種類] を [自動] から [混合] または [ネイティブのみ] に変更して、Visual Studio が .NET 同期スクリプトのみをデバッグするようにします。Visual Studio 2010 では、Mobile Link サーバによって使用されるアセンブリのバージョンに応じて、[デバッガーの種類] を [マネージ (v2.0、v1.1、v1.0)] または [マネージ v4.0] に変更します。

   注意
   v4.0 のアセンブリを使用するには、Mobile Link サーバをロードするときに -clrVersion オプションを明示的にインクルードする必要があります。 -clrVersion オプションの詳細については、「-sl dnet mlsrv16 オプション」76 ページを参照してください。

5. 関連する .NET ソースファイルを開き、ブレークポイントを設定します。

   mlsrv16 ソリューションでソースファイルを個別に開きます。元のソリューションやプロジェクトファイルは開かないでください。

6. [デバッグ] メニューまたは [F5] キーを押して Mobile Link を起動します。

   プロンプトが表示されたら、*mlsrv16.sln* を保存します。

   [シンボル情報なし] ウィンドウが表示された場合は、**[OK]** をクリックしてデバッグを続けます。デバッグしているのは、Mobile Link が呼び出す管理対象の .NET 同期スクリプトであり、Mobile Link サーバ本体ではありません。

7. 同期を実行します。この結果、ブレークポイントのあるコードが Mobile Link によって実行されます。

結果

スクリプトはデバッグ可能です。

.NET の同期の方法

この項では、一般的な .NET の同期タスクに取り組む場合に使用できる方法について説明します。

ローのアップロードとダウンロード

.NET を介したローのアップロードまたはダウンロードの詳細については、「ダイレクトローハンドリング」623 ページを参照してください。

共有アセンブリをロードする方法

この項では、.NET アセンブリをロードするためのオプションと共有アセンブリをロードする処理について説明します。

.NET アセンブリのロード

.NET アセンブリは、タイプ、メタデータ、プログラムコードのパッケージです。.NET アプリケーションでは、すべてのコードがアセンブリになければなりません。アセンブリファイルの拡張子は *.dll* または *.exe* です。

アセンブリには、次の種類があります。

- **プライベートアセンブリ**　ファイルシステム内のファイル。
- **共有アセンブリ**　グローバルアセンブリキャッシュにインストールされるアセンブリ。

Mobile Link では、クラスを含むアセンブリが指定されないと、クラスをロードしてそのクラスのメソッドを呼び出すことができません。指定する必要があるのは、Mobile Link が直接呼び出すアセンブリのみです。このアセンブリによって、その他の必要なアセンブリが呼び出されます。

たとえば、Mobile Link が MyAssembly を呼び出すと、MyAssembly が UtilityAssembly と NetworkingUtilsAssembly を呼び出します。この場合は、Mobile Link が MyAssembly だけを探すように設定します。

Mobile Link では、次の方法でアセンブリをロードできます。

- **-sl dnet (-MLAutoLoadPath) を使用**　このオプションは、プライベートアセンブリに対してのみ有効です。このオプションは、アプリケーションのベースディレクトリへのパスを設定し、そのディレクトリ内のすべてのアセンブリをロードします。

  -MLAutoLoadPath オプションを使用すると、イベントスクリプトの完全に修飾されたメソッド名を入力するときに、ドメインを指定できません。

  -MLAutoLoadPath でパスとディレクトリを指定すると、次のアクションが実行されます。

  ○このパスがアプリケーションベースパスとして設定される

  ○指定したディレクトリで *.dll* または *.exe* の付くすべてのファイルのすべてのクラスがロードされる

  ○1 つのアプリケーションドメインが作成され、ドメインが指定されていないすべてのユーザクラスがそのドメインにロードされる

  このオプションでは、グローバルアセンブリキャッシュ内のアセンブリを直接呼び出すことはできません。これらの共有アセンブリを呼び出すには、-MLDomConfigFile を使用します。

- **-sl dnet (-MLDomConfigFile) を使用**　このオプションは、プライベートアセンブリと共有アセンブリの両方に使用できます。このオプションを使用するには、ドメインとアセンブリの設定を含む設定ファイルが必要です。このオプションは、共有アセンブリがある場合、アプリケーションベースパス内のすべてのアセンブリをロードする必要がない場合、またはその他の理由で設定ファイルを使用する必要がある場合に使用してください。

  このオプションを使用すると、Mobile Link は指定されたドメイン設定ファイル内の設定を読み込みます。ドメイン設定ファイルには、1 つまたは複数の .NET ドメインの設定が入っています。このファイルに複数のドメインが記述されている場合は、1 番目に指定されているドメインがデフォルトドメインとして使用されます (デフォルトドメインは、指定されたドメインがスクリプトにない場合に使用されます)。

  Mobile Link はアセンブリをロードするときに、まずプライベートアセンブリとしてロードし、次にグローバルアセンブリキャッシュからアセンブリをロードしようとします。プライベートアセンブリは、アプリケーションベースフォルダにあります。共有アセンブリはグローバルアセンブリキャッシュからロードされます。

-MLDomConfigFile オプションでは、ドメイン設定ファイルで指定されているアセンブリのみが、イベントスクリプトから直接呼び出せます。

サンプルのドメイン設定ファイル

mlDomConfig.xml というサンプルのドメイン設定ファイルが、Mobile Link とともにインストールされます。独自のファイルを最初から作成するか、要件に合わせてサンプルを編集できます。サンプルファイルは、SQL Anywhere の次のパスにあります。

MobiLink¥setup¥dnet¥mlDomConfig.xml

サンプルのドメイン設定ファイル *mlDomConfig.xml* の内容は次のとおりです。

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="iAnywhere.MobiLink.mlDomConfig"
```

```
xsi:schemaLocation='iAnywhere.MobiLink.mlDomConfig mlDomConfig.xsd' xmlns:xsi='http://
www.w3.org/2001/XMLSchema-instance' >
  <domain>
    <name>SampleDomain1</name>
    <appBase>C:¥scriptsDir</appBase>
    <configFile></configFile>
    <assembly name="Assembly1" />
    <assembly name="Assembly2" />
  </domain>

  <domain>
    <name>SampleDomain2</name>
    <appBase>¥Dom2assembly</appBase>
    <configFile>¥Dom2assembly¥AssemblyRedirects.config</configFile>
    <assembly name="Assembly3" />
    <assembly name="Assembly4" />
  </domain>
</config>
```

次に、*mlDomConfig.xml* の内容について説明します。

- **name**　イベントスクリプトでドメインを指定するときに使用するドメイン名です。"DomainName:Namespace.Class.Method" というフォーマットのイベントスクリプトを使用するには、ドメイン設定ファイル内に DomainName ドメインが必要です。

  ドメイン名は 1 つ以上指定してください。

- **appBase**　ドメインがアプリケーションベースディレクトリとして使用するディレクトリです。すべてのプライベートアセンブリは、このディレクトリに基づいて .NET CLR によってロードされます。appBase は必ず指定してください。

- **configFile**　ドメインに使用する .NET アプリケーション設定ファイルです。ここは空白でもかまいません。通常は、この部分を使用してアセンブリのバインドとロードのデフォルトの動作を変更します。アプリケーション設定ファイルの詳細については、.NET のマニュアルを参照してください。

- **assembly**　イベントスクリプト内の型の参照を解決するときに Mobile Link がロードして検索するアセンブリの名前です。アセンブリは 1 つ以上指定してください。アセンブリが複数のドメインで使用されている場合は、ドメインごとに 1 つのアセンブリを指定してください。プライベートアセンブリの場合は、ドメインのアプリケーションベースディレクトリになければなりません。

mlsrv16 オプションの -sl dnet の詳細については、「-sl dnet mlsrv16 オプション」76 ページを参照してください。

.NET 同期のサンプル

このサンプルは、.NET 同期論理を使用して authenticate_user イベントを処理する方法を表示するように既存のアプリケーションを変更します。このサンプルは、*AuthUser.cs* という名前の authenticate_user 用の C# スクリプトを作成します。このスクリプトは、user_pwd_table というテーブル内でユーザのパスワードを検索し、そのパスワードに基づいてユーザを認証します。

1. テーブル user_pwd_table をデータベースに追加します。Interactive SQL で次の SQL 文を実行します。

```
CREATE TABLE user_pwd_table (
  user_name  varchar(128) PRIMARY KEY NOT NULL,
  pwd        varchar(128)
)
```

2. ユーザとパスワードをテーブルに追加します。

```
INSERT INTO user_pwd_table VALUES('user1', 'myPwd')
```

3. .NET アセンブリ用のディレクトリを作成します例：*c:¥mlexample*。

4. 次の内容の *AuthUser.cs* というファイルを作成します。

「authenticate_user 接続イベント」291 ページを参照してください。

```
using System;
using iAnywhere.MobiLink.Script;
namespace MLExample {

public class AuthClass {
   private DBConnection  _conn;

   /// AuthClass constructor.
   public AuthClass(DBConnectionContext cc) {
      _conn   = cc.GetConnection();
   }

   /// The DoAuthenticate method handles the 'authenticate_user'
   /// event.
   /// Note: This method does not handle password changes for
   /// advanced authorization status codes.
   public void DoAuthenticate(
      ref int authStatus,
      string user,
      string pwd,
      string newPwd)
   {
      DBCommand   pwd_command = _conn.CreateCommand();
      pwd_command.CommandText = "select pwd from user_pwd_table"
         + " where user_name = ? ";
      pwd_command.Prepare();

      // Add a parameter for the user name.
      DBParameter user_param  = new DBParameter();
      user_param.DbType       = SQLType.SQL_CHAR;

      // Set the size for SQL_VARCHAR.
      user_param.Size     = (uint) user.Length;
      user_param.Value    = user;
      pwd_command.Parameters.Add(user_param);

      // Fetch the password for this user.
      DBRowReader rr      = pwd_command.ExecuteReader();
      object[] pwd_row    = rr.NextRow();

      if (pwd_row == null) {
         // User is unknown.
         authStatus  = 4000;
      }
```

```
            else {
                if (((string) pwd_row[0]) == pwd) {
                    // Password matched.
                    authStatus = 1000;
                }
                else {
                    // Password did not match.
                    authStatus = 4000;
                }
            }
            pwd_command.Close();
            rr.Close();
            return;
        }
    }
```

MLExample.AuthClass.DoAuthenticate メソッドは、authenticate_user イベントを処理します。これはユーザ名とパスワードを受け入れ、検証の成功または失敗を示す認証ステータスコードを返します。

5. ファイル *AuthUser.cs* をコンパイルします。コンパイルは、コマンドラインまたは Visual Studio で実行できます。

   たとえば、次のコマンドラインは *AuthUser.cs* をコンパイルし、*example.dll* という名前のアセンブリを *c:¥mlexample* に生成します。

   ```
   csc /out:c:¥mlexample¥example.dll /target:library /reference:"%SQLANY16%¥Assembly
   ¥V2¥iAnywhere.MobiLink.Script.dll" AuthUser.cs
   ```

6. authenticate_user イベント用の .NET コードを登録します。実行する必要があるメソッドは、ネームスペース MLExample とクラス AuthClass にあります。次の SQL を実行します。

   ```
   CALL ml_add_dnet_connection_script('ex_version', 'authenticate_user',
   'MLExample.AuthClass.DoAuthenticate');
   COMMIT
   ```

7. 次のオプションで Mobile Link サーバを実行します。このオプションによって、Mobile Link が *c:¥myexample* 内のすべてのアセンブリをロードします。

   ```
   -sl dnet (-MLAutoLoadPath=c:¥mlexample)
   ```

これで、ユーザがバージョン ex_version と同期するときに、テーブル user_pwd_table のパスワードで認証されるようになります。

Mobile Link サーバ .NET API リファレンス

この項では、Mobile Link .NET のインタフェースとクラス、これらに関連するメソッド、プロパティ、コンストラクタについて説明します。これらのクラスを使用するには、*%SQLANY16%¥Assembly¥V2* にある *iAnywhere.MobiLink.Script.dll* アセンブリを参照してください。

ネームスペース

`iAnywhere.MobiLink.Script`

DateTimeWithTimeZone クラス

タイムゾーンオフセット付きの DateTime を表します。

Visual Basic の構文

```
Public NotInheritable Class DateTimeWithTimeZone
```

C# の構文

```
public sealed class DateTimeWithTimeZone
```

メンバー

継承されたメンバーを含む DateTimeWithTimeZone クラスのすべてのメンバー。

名前	説明
DateTimeWithTimeZone コンストラクタ	指定された年、月、日、時間、分、秒、ミリ秒、タイムゾーンの時間、タイムゾーンの分を含む DateTimeWithTimeZone を構成します。
Parse メソッド	指定された文字列を解析して、新しい DateTimeWithTimeZone を返します。
ToString メソッド	タイムゾーンオフセットが最後に追加された DateTime.ToString() と同じです。
DateTime プロパティ	タイムゾーンオフセットなしでこれに対応する DateTime を取得します。
Day プロパティ	DateTime.Day と同じです。
Hour プロパティ	DateTime.Hour と同じです。
Millisecond プロパティ	DateTime.Millisecond と同じです。
Minute プロパティ	DateTime.Minute と同じです。
Month プロパティ	DateTime.Month と同じです。
Second プロパティ	DateTime.Second と同じです。
TimeZoneHour プロパティ	タイムゾーンオフセットの時間の部分を取得します。
TimeZoneMinute プロパティ	タイムゾーンオフセットの分の部分を取得します。
Year プロパティ	DateTime.Year と同じです。

DateTimeWithTimeZone コンストラクタ

指定された年、月、日、時間、分、秒、ミリ秒、タイムゾーンの時間、タイムゾーンの分を含む DateTimeWithTimeZone を構成します。

オーバロードリスト

名前	説明
DateTimeWithTimeZone(DateTime) コンストラクタ	タイムゾーンオフセットが 0 で指定された DateTime と同じ日付と時刻を含む DateTimeWithTimeZone を構成します。
DateTimeWithTimeZone(DateTime, int, int) コンストラクタ	指定のタイムゾーンオフセットで指定された DateTime と同じ日付と時刻を含む DateTimeWithTimeZone を構成します。
DateTimeWithTimeZone(int, int, int, int, int, int, int, int, int) コンストラクタ	指定された年、月、日、時間、分、秒、ミリ秒、タイムゾーンの時間、タイムゾーンの分を含む DateTimeWithTimeZone を構成します。

DateTimeWithTimeZone(DateTime) コンストラクタ

タイムゾーンオフセットが 0 で指定された DateTime と同じ日付と時刻を含む DateTimeWithTimeZone を構成します。

Visual Basic の構文

```
Public Sub New(ByVal dt As Date)
```

C# の構文

```
public DateTimeWithTimeZone(DateTime dt)
```

DateTimeWithTimeZone(DateTime, int, int) コンストラクタ

指定のタイムゾーンオフセットで指定された DateTime と同じ日付と時刻を含む DateTimeWithTimeZone を構成します。

Visual Basic の構文

```
Public Sub New(
    ByVal dt As Date,
    ByVal tz_hour As Integer,
    ByVal tz_minute As Integer
)
```

C# の構文

```
public DateTimeWithTimeZone(DateTime dt, int tz_hour, int tz_minute)
```

 Copyright © 2013, SAP AG or an SAP affiliate company.

パラメータ

- **dt**　日付と時刻の部分。
- **tz_hour**　タイムゾーンオフセットの時間 (-12 ～ 14)
- **tz_minute**　タイムゾーンオフセットの分 (-59 ～ 59)。*tz_hour* が正でない場合にのみ、負の値を指定できます。

例外

- **System.ArgumentOutOfRangeException**　tz_hour または tz_minute が範囲外の場合にスローされます。

DateTimeWithTimeZone(int, int, int, int, int, int, int, int, int) コンストラクタ

指定された年、月、日、時間、分、秒、ミリ秒、タイムゾーンの時間、タイムゾーンの分を含む DateTimeWithTimeZone を構成します。

Visual Basic の構文

```
Public Sub New(
    ByVal year As Integer,
    ByVal month As Integer,
    ByVal day As Integer,
    ByVal hour As Integer,
    ByVal minute As Integer,
    ByVal second As Integer,
    ByVal millisecond As Integer,
    ByVal tz_hour As Integer,
    ByVal tz_minute As Integer
)
```

C# の構文

```
public DateTimeWithTimeZone(
    int year,
    int month,
    int day,
    int hour,
    int minute,
    int second,
    int millisecond,
    int tz_hour,
    int tz_minute
)
```

パラメータ

- **year**　年 (1 ～ 9999)
- **month**　月 (1 ～ 12)
- **day**　日 (1 ～ *month* の日数)

- **hour**　時間 (0 ～ 23)
- **minute**　分 (0 ～ 59)
- **second**　秒 (0 ～ 59)
- **millisecond**　ミリ秒 (1 ～ 999)
- **tz_hour**　タイムゾーンオフセットの時間 (-12 ～ 14)
- **tz_minute**　タイムゾーンオフセットの分 (-59 ～ 59)。*tz_hour* が正でない場合にのみ、負の値を指定できます。

例外

- **System.ArgumentOutOfRangeException**　いずれかのパラメータが範囲外の場合にスローされます。

Parse メソッド

指定された文字列を解析して、新しい DateTimeWithTimeZone を返します。

Visual Basic の構文

```
Public Shared Function Parse(
    ByVal val As String
) As DateTimeWithTimeZone
```

C# の構文

```
public static DateTimeWithTimeZone Parse(string val)
```

パラメータ

- **val**　"yyyy-MM-dd HH:mm:ss.ffffff SHH:mm" 形式の文字列で、S はタイムゾーンの時間オフセットの符号を表し、2 つ目の HH:mm はタイムゾーンオフセットを表します。時刻の小数部分は省略できます。タイムゾーンオフセットは省略できます。タイムゾーンオフセットを指定する場合、符号は省略できます。

例外

- **System.FormatException**　指定された文字列がそのフォーマットと一致しない場合にスローされます。

ToString メソッド

タイムゾーンオフセットが最後に追加された DateTime.ToString() と同じです。

オーバロードリスト

名前	説明
ToString() メソッド	タイムゾーンオフセットが最後に追加された DateTime.ToString() と同じです。
ToString(IFormatProvider) メソッド	タイムゾーンオフセットが最後に追加された DateTime.ToString(provider) と同じです。
ToString(string) メソッド	タイムゾーンオフセットが最後に追加された DateTime.ToString(format) と同じです。
ToString(string, IFormatProvider) メソッド	タイムゾーンオフセットが最後に追加された DateTime.ToString(format, provider) と同じです。

ToString() メソッド

タイムゾーンオフセットが最後に追加された DateTime.ToString() と同じです。

Visual Basic の構文

```
Public Overrides Function ToString() As String
```

C# の構文

```
public override string ToString()
```

ToString(IFormatProvider) メソッド

タイムゾーンオフセットが最後に追加された DateTime.ToString(provider) と同じです。

Visual Basic の構文

```
Public Function ToString(ByVal provider As IFormatProvider) As String
```

C# の構文

```
public string ToString(IFormatProvider provider)
```

ToString(string) メソッド

タイムゾーンオフセットが最後に追加された DateTime.ToString(format) と同じです。

Visual Basic の構文

```
Public Function ToString(ByVal format As String) As String
```

C# の構文

```
public string ToString(string format)
```

ToString(string, IFormatProvider) メソッド

タイムゾーンオフセットが最後に追加された DateTime.ToString(format, provider) と同じです。

Visual Basic の構文

```
Public Function ToString(
    ByVal format As String,
    ByVal provider As IFormatProvider
) As String
```

C# の構文

```
public string ToString(string format, IFormatProvider provider)
```

DateTime プロパティ

タイムゾーンオフセットなしでこれに対応する DateTime を取得します。

Visual Basic の構文

```
Public Property DateTime As Date
```

C# の構文

```
public DateTime DateTime {get;set;}
```

Day プロパティ

DateTime.Day と同じです。

Visual Basic の構文

```
Public ReadOnly Property Day As Integer
```

C# の構文

```
public int Day {get;}
```

Hour プロパティ

DateTime.Hour と同じです。

Visual Basic の構文

```
Public ReadOnly Property Hour As Integer
```

C# の構文

```
public int Hour {get;}
```

Millisecond プロパティ

DateTime.Millisecond と同じです。

Visual Basic の構文

```
Public ReadOnly Property Millisecond As Integer
```

C# の構文

```
public int Millisecond {get;}
```

Minute プロパティ

DateTime.Minute と同じです。

Visual Basic の構文

```
Public ReadOnly Property Minute As Integer
```

C# の構文

```
public int Minute {get;}
```

Month プロパティ

DateTime.Month と同じです。

Visual Basic の構文

```
Public ReadOnly Property Month As Integer
```

C# の構文

```
public int Month {get;}
```

Second プロパティ

DateTime.Second と同じです。

Visual Basic の構文

```
Public ReadOnly Property Second As Integer
```

C# の構文

```
public int Second {get;}
```

TimeZoneHour プロパティ

タイムゾーンオフセットの時間の部分を取得します。

Visual Basic の構文

```
Public Property TimeZoneHour As Integer
```

C# の構文

```
public int TimeZoneHour {get;set;}
```

TimeZoneMinute プロパティ

タイムゾーンオフセットの分の部分を取得します。

Visual Basic の構文

```
Public Property TimeZoneMinute As Integer
```

C# の構文

```
public int TimeZoneMinute {get;set;}
```

Year プロパティ

DateTime.Year と同じです。

Visual Basic の構文

```
Public ReadOnly Property Year As Integer
```

C# の構文

```
public int Year {get;}
```

DBCommand インタフェース

SQL 文またはデータベースコマンドを表します。

Visual Basic の構文

```
Public Interface DBCommand
```

C# の構文

```
public interface DBCommand
```

メンバー

継承されたメンバーを含む DBCommand インタフェースのすべてのメンバー。

名前	説明
Close メソッド	現在の SQL 文またはコマンドを終了します。
ExecuteNonQuery メソッド	non-query の文を実行します。

名前	説明
ExecuteReader メソッド	結果セットを返すクエリ文を実行します。
Prepare メソッド	CommandText に格納されている SQL 文の実行を準備します。
CommandText プロパティ	実行される SQL 文。
Parameters プロパティ	この DBCommand の DBParameterCollection を取得します。

備考

DBCommand は更新またはクエリを表すことができます。

例

次の C# コードは DBCommand インタフェースを使用して次の 2 つのクエリを実行します。

```
DBCommand stmt = conn.CreateCommand();
stmt.CommandText = "SELECT t1a1, t1a2 FROM table1 ";

DBRowReader rs = stmt.ExecuteReader();
printResultSet( rs );
rs.Close();

stmt.CommandText = "SELECT t2a1 FRONT table2 ";
rs = stmt.ExecuteReader();
printResultSet( rs );
rs.Close();
stmt.Close();
```

次の C# コードは DBCommand インタフェースを使用してパラメータを指定して更新を実行します。

```
public void prepare_for_download(DateTime last_download,
    String ml_username)
{
        DBCommand cstmt = conn.CreateCommand();
        cstmt.CommandText = "CALL myProc( ?,?,? )";
        cstmt.Prepare();

        DBParameter param = new DBParameter();
        param.DbType          = SQLType.SQL_CHAR;
        param.Value           = "10000";
        cstmt.Parameters.Add(param);

        param                 = new DBParameter();
        param.DbType          = SQLType.SQL_INTEGER;
        param.Value           = 20000;
        cstmt.Parameters.Add(param);

        param                 = new DBParameter();
        param.DbType          = SQLType.SQL_DECIMAL;
        param.Precision       = 5;
        param.Value           = new Decimal(30000);
        cstmt.Parameters.Add(param);
```

```
        // Execute update
        DBRowReader rset = cstmt.ExecuteNonQuery();
        cstmt.Close();
    }
```

Close メソッド

現在の SQL 文またはコマンドを終了します。

Visual Basic の構文

```
Public Sub Close()
```

C# の構文

```
public void Close()
```

ExecuteNonQuery メソッド

non-query の文を実行します。

Visual Basic の構文

```
Public Function ExecuteNonQuery() As Integer
```

C# の構文

```
public int ExecuteNonQuery()
```

戻り値

データベース内で SQL 文の影響を受けるローの数

ExecuteReader メソッド

結果セットを返すクエリ文を実行します。

Visual Basic の構文

```
Public Function ExecuteReader() As DBRowReader
```

C# の構文

```
public DBRowReader ExecuteReader()
```

戻り値

SQL 文が返す結果を取得するための DBRowReader

Prepare メソッド

CommandText に格納されている SQL 文の実行を準備します。

Visual Basic の構文

```
Public Sub Prepare()
```

C# の構文

```
public void Prepare()
```

CommandText プロパティ

実行される SQL 文。

Visual Basic の構文

```
Public Property CommandText As String
```

C# の構文

```
public string CommandText {get;set;}
```

Parameters プロパティ

この DBCommand の DBParameterCollection を取得します。

Visual Basic の構文

```
Public ReadOnly Property Parameters As DBParameterCollection
```

C# の構文

```
public DBParameterCollection Parameters {get;}
```

戻り値

要求された Parameter コレクション

参照

- DBParameterCollection クラス [Mobile Link サーバ .NET]571 ページ

DBConnection インタフェース

Mobile Link ODBC 接続を表します。

Visual Basic の構文

```
Public Interface DBConnection
```

C# の構文

```
public interface DBConnection
```

メンバー

継承されたメンバーを含む DBConnection インタフェースのすべてのメンバー。

名前	説明
Close メソッド	現在の接続を閉じます。
Commit メソッド	現在のトランザクションをコミットします。
CreateCommand メソッド	この接続で SQL 文またはコマンドを作成します。
Rollback メソッド	現在のトランザクションをロールバックします。

備考

このインタフェースにより、ユーザが作成した同期論理で Mobile Link によって確立された ODBC 接続にアクセスできます。

Close メソッド

現在の接続を閉じます。

Visual Basic の構文

```
Public Sub Close()
```

C# の構文

```
public void Close()
```

Commit メソッド

現在のトランザクションをコミットします。

Visual Basic の構文

```
Public Sub Commit()
```

C# の構文

```
public void Commit()
```

CreateCommand メソッド

この接続で SQL 文またはコマンドを作成します。

Visual Basic の構文

```
Public Function CreateCommand() As DBCommand
```

C# の構文

```
public DBCommand CreateCommand()
```

戻り値

新しく生成された DBCommand

Rollback メソッド

現在のトランザクションをロールバックします。

Visual Basic の構文

```
Public Sub Rollback()
```

C# の構文

```
public void Rollback()
```

DBConnectionContext インタフェース

現在のデータベース接続に関する情報を取得します。

Visual Basic の構文

```
Public Interface DBConnectionContext
```

C# の構文

```
public interface DBConnectionContext
```

メンバー

継承されたメンバーを含む DBConnectionContext インタフェースのすべてのメンバー。

名前	説明
GetConnection メソッド	Mobile Link 統合データベースへの既存の接続を返します。
GetDownloadData メソッド	現在の同期に対する DownloadData を返します。
GetProperties メソッド	この接続のスクリプトバージョンに基づいて、プロパティのコレクションを返します。
GetRemoteID メソッド	この接続で現在同期中のデータベースのリモート ID を返します。
GetServerContext メソッド	現在のサーバコンテキストを返します。

名前	説明
GetVersion メソッド	この接続のバージョン文字列を返します。
NetworkData プロパティ	同期用のネットワークストリームの情報を返します。

備考

このインタフェースは、スクリプトを含むクラスのコンストラクタに渡されます。コンテキストがバックグラウンドスレッドに必要な場合や、接続期間を超えて必要な場合は、ServerContext を使用してください。

コンストラクタの詳細については、「コンストラクタ」537 ページを参照してください。

注意
DBConnectionContext インスタンスは、.NET コードに呼び出すスレッド以外で使用しないでください。

例

次の例は、同期スクリプトで使用するクラスレベルの DBConnectionContext インスタンスを作成する方法を示します。DBConnectionContext の getConnection メソッドは、Mobile Link 統合データベースとの現在の接続を表す DBConnection インスタンスを取得します。

```
using iAnywhere.MobiLink.Script;
using System.Data;

public class OrderProcessor {
    DBConnectionContext _cc;

    public OrderProcessor( DBConnectionContext cc ) {
        _cc = cc;
    }

    // The method used for the handle_DownloadData event.
    public void HandleEvent() {
        DBConnection my_connection = _cc.GetConnection();
        // ...
    }

    // ...
}
```

GetConnection メソッド

Mobile Link 統合データベースへの既存の接続を返します。

Visual Basic の構文

```
Public Function GetConnection() As DBConnection
```

 Copyright © 2013, SAP AG or an SAP affiliate company.

C# の構文

```
public DBConnection GetConnection()
```

戻り値

統合データベースへの現在の接続。この接続は、基本となる Mobile Link 接続の期間中にのみ有効です。

備考

Mobile Link が SQL スクリプトの実行に使用するものと同じ接続です。Mobile Link サーバのこの接続での使用に影響する方法でコミット、終了、または変更しないでください。

接続に対して end_connection イベントが呼び出された後は、その接続を使用しないでください。

フルアクセス権を持つサーバ接続が必要な場合は、MakeConnection メソッドを使用します。

参照

- ServerContext.MakeConnection メソッド [Mobile Link サーバ .NET]603 ページ

GetDownloadData メソッド

現在の同期に対する DownloadData を返します。

Visual Basic の構文

```
Public Function GetDownloadData() As DownloadData
```

C# の構文

```
public DownloadData GetDownloadData()
```

戻り値

現在の同期に対する DownloadData、同期がアップロード専用の場合は NULL

備考

ダイレクトローハンドリング用のダウンロードを作成する場合は、DownloadData インスタンスを使用してください。

例

次の例は、_cc という DBConnectionContext インスタンスが作成済みであることを前提としています。

```
// The method used for the handle_DownloadData event.
public void HandleDownload() {
   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.GetDownloadData();

   // Get an array of tables to set download operations.
      DownloadTableData[] download_tables = my_dd.GetDownloadTables();

      // Get the first table in the DownloadTableData array.
   DownloadTableData my_download_table = download_tables[0];
```

```
    // ...
}
```

GetProperties メソッド

この接続のスクリプトバージョンに基づいて、プロパティのコレクションを返します。

Visual Basic の構文

```
Public Function GetProperties() As NameValueCollection
```

C# の構文

```
public NameValueCollection GetProperties()
```

戻り値

現在のスクリプトバージョンのプロパティ

備考

プロパティは、ml_property テーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

例

次の例は、DBConnectionContext のプロパティを出力する方法を示します。この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used to output the connection properties.
public void OutputProperties() {
    // output the Properties for the current synchronization
    NameValueCollection properties = _cc.GetProperties();
    System.Console.WriteLine(properties.ToString());
}
```

GetRemoteID メソッド

この接続で現在同期中のデータベースのリモート ID を返します。

Visual Basic の構文

```
Public Function GetRemoteID() As String
```

C# の構文

```
public string GetRemoteID()
```

戻り値

リモート ID。

備考

リモート ID の詳細については、「リモート ID」『Mobile Link クライアント管理』を参照してください。

例

次の例は、DBConnectionContext のリモート ID を出力する方法を示します。

```
// The method used to output the remote ID.
public void OutputRemoteID() {
    // output the Remote ID for the current synchronization
    string remoteID = _cc.GetRemoteID();
    System.Console.WriteLine(remoteID);
}
```

GetServerContext メソッド

現在のサーバコンテキストを返します。

Visual Basic の構文

```
Public Function GetServerContext() As ServerContext
```

C# の構文

```
public ServerContext GetServerContext()
```

戻り値

この Mobile Link サーバの ServerContext

備考

このメソッドは、新しい接続を作成したり、ブートクラスと対話したりするために使用できます。

例

次の例は、DBConnectionContext の ServerContext インスタンスを取得して、サーバを停止する方法を示します。

```
// A method that uses an instance of the ServerContext to shut down the server
public void ShutDownServer() {
    ServerContext context = _cc.GetServerContext();
    context.Shutdown();
}
```

GetVersion メソッド

この接続のバージョン文字列を返します。

Visual Basic の構文

```
Public Function GetVersion() As String
```

C# の構文

```
public string GetVersion()
```

戻り値

スクリプトバージョン名

備考

プロパティは、ml_property テーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

例

次の例は、スクリプトバージョンを取得して、決定を行うために使用する方法を示します。

```
public void MyEvent() {
   // ...

   string version = _cc.GetVersion();
   switch( version ) {
   case "My Version 1":
      // ...
      break;
   case "My Version 2":
      // ...
      break;
   }
}
```

NetworkData プロパティ

同期用のネットワークストリームの情報を返します。

Visual Basic の構文

```
Public ReadOnly Property NetworkData As NetworkData
```

C# の構文

```
public NetworkData NetworkData {get;}
```

戻り値

要求に使用されたネットワークストリームの情報、コレクションが有効でない場合は NULL

備考

この方法は、クライアント側証明書と HTTP ヘッダを使用するエンタープライズ内の別のサーバを認証する際に便利です。

ネットワークストリームのデータの収集を有効にするには、-x スイッチに collect_network_data=1 を追加します。このオプションにより、同期ごとのデータの保存時に、メモリのオーバヘッドが増えます。

参照

- NetworkData インタフェース [Mobile Link サーバ .NET]593 ページ

DBParameter クラス

ODBC バウンドパラメータを表します。

Visual Basic の構文

```
Public Class DBParameter
```

C# の構文

```
public class DBParameter
```

メンバー

継承されたメンバーを含む DBParameter クラスのすべてのメンバー。

名前	説明
DbType プロパティ	このパラメータの SQLType。
Direction プロパティ	このパラメータの入出力方向。
IsNullable プロパティ	値に NULL を指定できる場合は true、そうでない場合は false。
ParameterName プロパティ	このパラメータの名前。
Precision プロパティ	このパラメータの 10 進数精度。
Scale プロパティ	このパラメータの解決可能な桁数。
Size プロパティ	このパラメータのサイズ (バイト単位)。
Value プロパティ	このパラメータの値。
HasChanged フィールド	パラメータが作成後に変更されたかどうかを返します。

備考

パラメータを指定してコマンドを実行するには、このクラスが必要です。すべてのパラメータを指定してからコマンドを実行してください。

例

次の C# コードは DBCommand インタフェースを使用してパラメータを指定して更新を実行します。

```
using( DBCommand cstmt = conn.CreateCommand() ) {
    DBCommand cstmt = conn.CreateCommand();

    cstmt.CommandText = "call myProc( ?,?,? )";

    cstmt.Prepare();

    DBParameter param = new DBParameter();
    param.DbType        = SQLType.SQL_CHAR;
    param.Value         = "10000";
    cstmt.Parameters.Add( param );

    param               = new DBParameter();
    param.DbType        = SQLType.SQL_INTEGER;
    param.Value         = 20000;
    cstmt.Parameters.Add( param );

    param               = new DBParameter();
    param.DbType        = SQLType.SQL_DECIMAL;
    param.Precision     = 5;
    param.Value         = new Decimal( 30000 );
    cstmt.Parameters.Add( param );

    // Execute update
    DBRowReader rset = cstmt.ExecuteNonQuery();
    cstmt.Close();
}
```

DbType プロパティ

このパラメータの SQLType。

Visual Basic の構文

```
Public Property DbType As SQLType
```

C# の構文

```
public SQLType DbType {get;set;}
```

備考

デフォルト値は SQLType.SQL_TYPE_NULL です。

Direction プロパティ

このパラメータの入出力方向。

Visual Basic の構文

```
Public Property Direction As ParameterDirection
```

C# の構文

```
public ParameterDirection Direction {get;set;}
```

備考

デフォルト値は ParameterDirection.Input です。

IsNullable プロパティ

値に NULL を指定できる場合は true、そうでない場合は false。

Visual Basic の構文

```
Public Property IsNullable As Boolean
```

C# の構文

```
public bool IsNullable {get;set;}
```

備考

デフォルト値は false です。

ParameterName プロパティ

このパラメータの名前。

Visual Basic の構文

```
Public Property ParameterName As String
```

C# の構文

```
public string ParameterName {get;set;}
```

備考

デフォルト値は NULL です。

Precision プロパティ

このパラメータの 10 進数精度。

Visual Basic の構文

```
Public Property Precision As UInteger
```

C# の構文

```
public uint Precision {get;set;}
```

備考

このプロパティは、SQLType.SQL_NUMERIC パラメータと SQLType.SQL_DECIMAL パラメータにのみ使用します。

デフォルト値は 0 です。

Scale プロパティ

このパラメータの解決可能な桁数。

Visual Basic の構文

```
Public Property Scale As Short
```

C# の構文

```
public short Scale {get;set;}
```

備考

このプロパティは、SQLType.SQL_NUMERIC パラメータと SQLType.SQL_DECIMAL パラメータにのみ使用します。

デフォルト値は 0 です。

Size プロパティ

このパラメータのサイズ (バイト単位)。

Visual Basic の構文

```
Public Property Size As UInteger
```

C# の構文

```
public uint Size {get;set;}
```

備考

デフォルト値は DbType から推定されます。

Value プロパティ

このパラメータの値。

Visual Basic の構文

```
Public Property Value As Object
```

C# の構文

```
public object Value {get;set;}
```

備考

デフォルト値は NULL です。

HasChanged フィールド

パラメータが作成後に変更されたかどうかを返します。

Visual Basic の構文

```
Public HasChanged As Boolean
```

C# の構文

```
public bool HasChanged;
```

DBParameterCollection クラス

DBParameter のコレクション。

Visual Basic の構文

```
Public Class DBParameterCollection
    Implements System.Data.IDataParameterCollection
    Implements System.Collections.IList
    Implements System.Collections.ICollection
    Implements System.Collections.IEnumerable
```

C# の構文

```
public class DBParameterCollection :
    System.Data.IDataParameterCollection,
    System.Collections.IList,
    System.Collections.ICollection,
    System.Collections.IEnumerable
```

基本クラス

- System.Data.IDataParameterCollection
- System.Collections.IList
- System.Collections.ICollection
- System.Collections.IEnumerable

メンバー

継承されたメンバーを含む DBParameterCollection クラスのすべてのメンバー。

名前	説明
DBParameterCollection コンストラクタ	DBParameter の空リストを作成します。
Add メソッド	指定したパラメータをコレクションに追加します。
Clear メソッド	すべてのパラメータをコレクションから削除します。
Contains メソッド	指定した名前のパラメータがコレクションに含まれているかどうかをチェックします。

名前	説明
CopyTo メソッド	コレクションの内容を、指定したインデックスで始まる特定の配列にコピーします。
GetEnumerator メソッド	コレクションの列挙子を返します。
IndexOf メソッド	指定した名前を持つパラメータのコレクション内のインデックスを返します。
Insert メソッド	指定した DBParameter を、コレクション内の指定したインデックス位置に挿入します。
Remove メソッド	指定した DBParameter をコレクションから削除します。
RemoveAt メソッド	指定した名前のパラメータをコレクションから削除します。
Count プロパティ	コレクション内のパラメータ数。
IsFixedSize プロパティ	false を返します。
IsReadOnly プロパティ	false を返します。
IsSynchronized プロパティ	false を返します。
SyncRoot プロパティ	DBParameterCollection へのアクセスを同期するために使用します。
this プロパティ	コレクション内の指定したインデックスを持つ DBParameter を取得または設定します。

備考

DBCommand で DBParameterCollection を作成した時点では、DBParameterCollection は空です。DBCommand を実行する前に、適切なパラメータを指定してください。

DBParameterCollection コンストラクタ

DBParameter の空リストを作成します。

Visual Basic の構文

```
Public Sub New()
```

C# の構文

```
public DBParameterCollection()
```

Add メソッド

指定したパラメータをコレクションに追加します。

Visual Basic の構文

```
Public Function Add(ByVal value As Object) As Integer
```

C# の構文

```
public int Add(object value)
```

パラメータ

- **value** コレクションに追加される DBParameter オブジェクト

戻り値

コレクションに追加されたパラメータのインデックス

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

Clear メソッド

すべてのパラメータをコレクションから削除します。

Visual Basic の構文

```
Public Sub Clear()
```

C# の構文

```
public void Clear()
```

Contains メソッド

指定した名前のパラメータがコレクションに含まれているかどうかをチェックします。

オーバロードリスト

名前	説明
Contains(object) メソッド	指定した DBParameter がコレクションに含まれる場合は true を返します。
Contains(string) メソッド	指定した名前のパラメータがコレクションに含まれているかどうかをチェックします。

Contains(object) メソッド

指定した DBParameter がコレクションに含まれる場合は true を返します。

Visual Basic の構文

```
Public Function Contains(ByVal value As Object) As Boolean
```

C# の構文

```
public bool Contains(object value)
```

パラメータ

- **value**　チェック対象の DBParameter オブジェクト

戻り値

このコレクションに DBParameter が含まれる場合は true、そうでない場合は false

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

Contains(string) メソッド

指定した名前のパラメータがコレクションに含まれているかどうかをチェックします。

Visual Basic の構文

```
Public Function Contains(ByVal parameterName As String) As Boolean
```

C# の構文

```
public bool Contains(string parameterName)
```

パラメータ

- **parameterName**　確認するパラメータの名前

戻り値

このコレクションに指定したパラメータが含まれる場合は true、そうでない場合は false

CopyTo メソッド

コレクションの内容を、指定したインデックスで始まる特定の配列にコピーします。

Visual Basic の構文

```
Public Sub CopyTo(ByVal array As Array, ByVal index As Integer)
```

C# の構文

```
public void CopyTo(Array array, int index)
```

パラメータ

- **array** コレクションの内容のコピー先となる配列
- **index** 配列の内容のコレクションへのコピーを開始する、配列内のインデックス

GetEnumerator メソッド

コレクションの列挙子を返します。

Visual Basic の構文

```
Public Function GetEnumerator() As System.Collections.IEnumerator
```

C# の構文

```
public System.Collections.IEnumerator GetEnumerator()
```

戻り値

コレクションの列挙子

IndexOf メソッド

指定した名前を持つパラメータのコレクション内のインデックスを返します。

オーバロードリスト

名前	説明
IndexOf(object) メソッド	コレクション内の指定された DBParameter のインデックスを返します。
IndexOf(string) メソッド	指定した名前を持つパラメータのコレクション内のインデックスを返します。

IndexOf(object) メソッド

コレクション内の指定された DBParameter のインデックスを返します。

Visual Basic の構文

```
Public Function IndexOf(ByVal value As Object) As Integer
```

C# の構文

```
public int IndexOf(object value)
```

パラメータ

- **value** 検索する DBParameter オブジェクト

戻り値

コレクション内の DBParameter のインデックス

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

IndexOf(string) メソッド

指定した名前を持つパラメータのコレクション内のインデックスを返します。

Visual Basic の構文

```
Public Function IndexOf(ByVal parameterName As String) As Integer
```

C# の構文

```
public int IndexOf(string parameterName)
```

パラメータ

- **parameterName**　検索するパラメータの名前

戻り値

パラメータのインデックス、指定した名前のパラメータがない場合は -1

Insert メソッド

指定した DBParameter を、コレクション内の指定したインデックス位置に挿入します。

Visual Basic の構文

```
Public Sub Insert(ByVal index As Integer, ByVal value As Object)
```

C# の構文

```
public void Insert(int index, object value)
```

パラメータ

- **value**　挿入する DBParameter オブジェクト
- **index**　値を挿入するインデックス位置

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

Remove メソッド

指定した DBParameter をコレクションから削除します。

Visual Basic の構文

```
Public Sub Remove(ByVal value As Object)
```

C# の構文

```
public void Remove(object value)
```

パラメータ

- **value**　削除する DBParameter

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

RemoveAt メソッド

指定した名前のパラメータをコレクションから削除します。

オーバロードリスト

名前	説明
RemoveAt(int) メソッド	コレクション内の指定したインデックス位置の DBParameter を削除します。
RemoveAt(string) メソッド	指定した名前のパラメータをコレクションから削除します。

RemoveAt(int) メソッド

コレクション内の指定したインデックス位置の DBParameter を削除します。

Visual Basic の構文

```
Public Sub RemoveAt(ByVal index As Integer)
```

C# の構文

```
public void RemoveAt(int index)
```

パラメータ

- **index**　削除する DBParameter のインデックス

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

RemoveAt(string) メソッド

指定した名前のパラメータをコレクションから削除します。

Visual Basic の構文

```
Public Sub RemoveAt(ByVal parameterName As String)
```

C# の構文

```
public void RemoveAt(string parameterName)
```

パラメータ

- **parameterName**　削除するパラメータの名前

Count プロパティ

コレクション内のパラメータ数。

Visual Basic の構文

```
Public ReadOnly Property Count As Integer
```

C# の構文

```
public int Count {get;}
```

IsFixedSize プロパティ

false を返します。

Visual Basic の構文

```
Public ReadOnly Property IsFixedSize As Boolean
```

C# の構文

```
public bool IsFixedSize {get;}
```

IsReadOnly プロパティ

false を返します。

Visual Basic の構文

```
Public ReadOnly Property IsReadOnly As Boolean
```

C# の構文

```
public bool IsReadOnly {get;}
```

IsSynchronized プロパティ

false を返します。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

Visual Basic の構文

```
Public ReadOnly Property IsSynchronized As Boolean
```

C# の構文

```
public bool IsSynchronized {get;}
```

SyncRoot プロパティ

DBParameterCollection へのアクセスを同期するために使用します。

Visual Basic の構文

```
Public ReadOnly Property SyncRoot As Object
```

C# の構文

```
public object SyncRoot {get;}
```

this プロパティ

コレクション内の指定した名前を持つ DBParameter を取得または設定します。

オーバロードリスト

名前	説明
this[int] プロパティ	コレクション内の指定したインデックスを持つ DBParameter を取得または設定します。
this[string] プロパティ	コレクション内の指定した名前を持つ DBParameter を取得または設定します。

this[int] プロパティ

コレクション内の指定したインデックスを持つ DBParameter を取得または設定します。

Visual Basic の構文

```
Public Property Item(ByVal index As Integer) As Object
```

C# の構文

```
public object this[int index] {get;set;}
```

パラメータ

- **index**　取得または設定する DBParameter のインデックス

戻り値

コレクション内の指定したインデックスを持つ this

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

this[string] プロパティ

コレクション内の指定した名前を持つ DBParameter を取得または設定します。

Visual Basic の構文

```
Public Property Item(ByVal parameterName As String) As Object
```

C# の構文

```
public object this[string parameterName] {get;set;}
```

パラメータ

- **parameterName**　取得または設定する DBParameter の名前

戻り値

コレクション内の指定した名前を持つ this

参照

- DBParameter クラス [Mobile Link サーバ .NET]567 ページ

DBRowReader インタフェース

データベースから読み込まれるローセットを表します。

Visual Basic の構文

```
Public Interface DBRowReader
```

C# の構文

```
public interface DBRowReader
```

メンバー

継承されたメンバーを含む DBRowReader インタフェースのすべてのメンバー。

名前	説明
Close メソッド	この MLDBRowReader で使用されるすべてのリソースをクリーンアップします。
NextRow メソッド	結果セット内の次のローを取得して返します。
ColumnNames プロパティ	結果セットに含まれるすべてのカラムの名前を取得します。

名前	説明
ColumnTypes プロパティ	結果セットに含まれるすべてのカラムの型を取得します。

備考

ExecuteReader メソッドを実行すると、DBRowReader が作成されます。

参照

- DBCommand.ExecuteReader メソッド [Mobile Link サーバ .NET]558 ページ

例

次の C# コードは、結果セット内で所定の DBRowReader で表されるローの関数を呼び出します。

```
    DBCommand stmt = conn.CreateCommand();
    stmt.CommandText = "select intCol, strCol from table1 ";
    DBRowReader rs = stmt.ExecuteReader();
  object[] values = rset.NextRow();

    while( values != null ) {
       handleRow( (int)values[0], (String)values[1] );
       values = rset.NextRow();
    }
    rset.Close();
    stmt.Close();
```

Close メソッド

この MLDBRowReader で使用されるすべてのリソースをクリーンアップします。

Visual Basic の構文

```
Public Sub Close()
```

C# の構文

```
public void Close()
```

備考

このメソッドを呼び出した後は、この MLDBRowReader を再び使用することはできません。

NextRow メソッド

結果セット内の次のローを取得して返します。

Visual Basic の構文

```
Public Function NextRow() As Object()
```

C# の構文

```
public object[] NextRow()
```

戻り値

結果セット内の次のローの値、この結果セットにローがこれ以上ない場合は NULL

参照

- SQLType 列挙体 [Mobile Link サーバ .NET]620 ページ

ColumnNames プロパティ

結果セットに含まれるすべてのカラムの名前を取得します。

Visual Basic の構文

```
Public ReadOnly Property ColumnNames As String()
```

C# の構文

```
public string[] ColumnNames {get;}
```

備考

値は、結果セット内のカラム名に相当する文字列の配列です。

ColumnTypes プロパティ

結果セットに含まれるすべてのカラムの型を取得します。

Visual Basic の構文

```
Public ReadOnly Property ColumnTypes As SQLType()
```

C# の構文

```
public SQLType[] ColumnTypes {get;}
```

備考

値は、結果セット内のカラムの型に対応する SQLTypes の配列です。

DownloadData インタフェース

ダイレクトローハンドリングで使用するすべてのダウンロードデータ操作をカプセル化します。

Visual Basic の構文

```
Public Interface DownloadData
```

C# の構文

```
public interface DownloadData
```

メンバー

継承されたメンバーを含む DownloadData インタフェースのすべてのメンバー。

名前	説明
GetDownloadTableByName メソッド	現在の同期に使用する名前付きダウンロードテーブルを取得します。
GetDownloadTables メソッド	現在の同期のダウンロードのすべてのテーブルを含む配列を取得します。

備考

DownloadData のインスタンスを取得するには、GetDownloadData メソッドを使用します。DownloadTableData のインスタンスを返すには、GetDownloadTables メソッドと GetDownloadTableByName メソッドを使用します。

ダウンロードデータは DBConnectionContext を使用して取得できます。begin_sync イベントの前にダウンロードデータにアクセスしたり、ダウンロード専用同期で DownloadData にアクセスしたりすることはできません。

ダイレクトローハンドリングの詳細については、「handle_DownloadData 接続イベント」371 ページと「ダイレクトローハンドリング」623 ページを参照してください。

参照

- DownloadTableData インタフェース [Mobile Link サーバ .NET]584 ページ
- DBConnectionContext.GetDownloadData メソッド [Mobile Link サーバ .NET]563 ページ

GetDownloadTableByName メソッド

現在の同期に使用する名前付きダウンロードテーブルを取得します。

Visual Basic の構文

```
Public Function GetDownloadTableByName(
    ByVal table_name As String
) As DownloadTableData
```

C# の構文

```
public DownloadTableData GetDownloadTableByName(string table_name)
```

パラメータ

- **table_name**　ダウンロードデータの取得先テーブルの名前

戻り値

指定したテーブル名のダウンロードデータ、見つからなかった場合は NULL

GetDownloadTables メソッド

現在の同期のダウンロードのすべてのテーブルを含む配列を取得します。

Visual Basic の構文

```
Public Function GetDownloadTables() As DownloadTableData()
```

C# の構文

```
public DownloadTableData[] GetDownloadTables()
```

戻り値

ダウンロードテーブルデータの配列。配列内でのテーブルの順序は、リモートのアップロード順と同じです。

備考

このテーブルに対して実行された操作はリモートデータベースに送信されます。

例

次の例では GetDownloadTables メソッドを使用して、現在の同期の DownloadTableData オブジェクトの配列を取得します。この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void HandleDownload() {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get an array of tables to set download operations.
    DownloadTableData[] download_tables = my_dd.GetDownloadTables();

    // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

    // ...
}
```

DownloadTableData インタフェース

1 つのダウンロードテーブルの情報を同期用にカプセル化します。

Visual Basic の構文

```
Public Interface DownloadTableData
```

C# の構文

```
public interface DownloadTableData
```

メンバー

継承されたメンバーを含む DownloadTableData インタフェースのすべてのメンバー。

名前	説明
GetDeleteCommand メソッド	ユーザが削除操作をダウンロードデータ操作に追加できるようにするコマンドを取得します。
GetLastDownloadTime メソッド	このテーブルの最終ダウンロード時刻を返します。
GetName メソッド	このインスタンスのテーブル名を取得します。
GetSchemaTable メソッド	このダウンロードテーブルのメタデータを記述する DataTable を取得します。
GetUpsertCommand メソッド	ユーザがアップサート (更新または挿入) をダウンロードデータ操作に追加できるようにするコマンドを取得します。

備考

このインタフェースを使用して、同期クライアントサイトにダウンロードされるデータ操作を設定します。

例

次のテーブルがあるとします。

```
CREATE TABLE remoteOrders (
   pk INT NOT NULL,
   col1 VARCHAR(200),
   PRIMARY KEY (pk)
);
```

次の例では GetDownloadTableByName メソッドを使用して、remoteOrders テーブルを表す DownloadTableData インスタンスを返します。

```
// The method used for the handle_DownloadData event
public void HandleDownload() {
   // _cc is a DBConnectionContext instance.

   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.GetDownloadData();

   // Get the DownloadTableData for the remoteOrders table.
   DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");

   // User defined-methods to set download operations.
   SetDownloadUpserts(td);
   SetDownloadDeletes(td);

   // ...
}
```

この例では、SetDownloadInserts メソッドは GetUpsertCommand を使用して、挿入または更新するローのコマンドを取得します。IDbCommand は、リモートデータベースに挿入する値の設定先となるパラメータを保持します。

```
void SetDownloadInserts(DownloadTableData td) {
   IDbCommand  upsert_cmd = td.GetUpsertCommand();
   IDataParameterCollection parameters = upsert_cmd.Parameters;

   // The following method calls are the same as the following SQL statement:
   // INSERT INTO remoteOrders(pk, col1) values(2300, "truck");
   ((IDataParameter) (parameters[0])).Value = (Int32) 2300;
   ((IDataParameter) (parameters[1])).Value = (String) "truck";

   if (upsert_cmd.ExecuteNonQuery() > 0) {
      // Insert was not filtered.
   }
   else {
      // Insert was filtered because it was uploaded
      // in the same synchronization.
   }
}
```

次のメソッドは DownloadTableData.GetDeleteCommand を使用して、削除するローのコマンドを取得します。

```
void SetDownloadDeletes(DownloadTableData td) {
   IDbCommand  delete_cmd = t2_download_dd.GetDeleteCommand();

   // The following method calls are the same as the following SQL statement:
   // DELETE FROM remoteOrders  where pk = 2300;
   IDataParameterCollection parameters = delete_cmd.Parameters;
   ((IDataParameter) (parameters[0])).Value = (Int32) 2300;
   delete_cmd.ExecuteNonQuery();
}
```

GetDeleteCommand メソッド

ユーザが削除操作をダウンロードデータ操作に追加できるようにするコマンドを取得します。

Visual Basic の構文

```
Public Function GetDeleteCommand() As IDbCommand
```

C# の構文

```
public IDbCommand GetDeleteCommand()
```

戻り値

ダウンロードでの削除に使用するコマンド

備考

返されるコマンドには、テーブルのプライマリキーカラムと同じ数のパラメータがあります。削除をダウンロードに含めるには、プライマリキーカラムのカラム値を設定し、文を ExecuteNonQuery を使用して実行する必要があります。コマンドの ExecuteNonQuery は、削除操作がフィルタされた場合は 0 を返し、挿入がフィルタされていない場合は 1 を返します。

ローを削除するには、ダウンロード削除操作対象のすべてのプライマリキー値を設定してください。リモートテーブルをトランケートするには、すべてのプライマリキーカラムを NULL に設定します。

参照

- DownloadTableData インタフェース [Mobile Link サーバ .NET]584 ページ

GetLastDownloadTime メソッド

このテーブルの最終ダウンロード時刻を返します。

Visual Basic の構文

```
Public Function GetLastDownloadTime() As Date
```

C# の構文

```
public DateTime GetLastDownloadTime()
```

戻り値

このテーブルの最後のダウンロード時刻

備考

これは、テーブルごとのダウンロードイベントの多くで渡される最終ダウンロード時刻と同じです。

最終ダウンロード時刻は、特定の同期に対してテーブルダウンロードデータを生成する場合に便利です。

GetName メソッド

このインスタンスのテーブル名を取得します。

Visual Basic の構文

```
Public Function GetName() As String
```

C# の構文

```
public string GetName()
```

戻り値

このインスタンスのテーブル名。

備考

これはユーティリティ関数です。テーブル名には、このインスタンスのスキーマを使用してもアクセスできます。

GetSchemaTable メソッド

このダウンロードテーブルのメタデータを記述する DataTable を取得します。

Visual Basic の構文

```
Public Function GetSchemaTable() As DataTable
```

C# の構文

```
public DataTable GetSchemaTable()
```

戻り値

カラムメタデータが記述された DataTable。

備考

DataTable にカラム名情報を含める場合は、カラム名を送信するためのクライアントオプションを指定してください。送信カラム名は、デフォルトで指定されます。

GetUpsertCommand メソッド

ユーザがアップサート (更新または挿入) をダウンロードデータ操作に追加できるようにするコマンドを取得します。

Visual Basic の構文

```
Public Function GetUpsertCommand() As IDbCommand
```

C# の構文

```
public IDbCommand GetUpsertCommand()
```

戻り値

ダウンロードの挿入／更新に使用するコマンド

備考

返されるコマンドには、このテーブルのカラムと同じ数のパラメータがあります。挿入／更新をダウンロードに含めるには、挿入するカラム値を設定し、文を ExecuteNonQuery を使用して実行する必要があります。コマンドの ExecuteNonQuery は、挿入操作がフィルタされた場合は 0 を返し、挿入がフィルタされていない場合は 1 を返します。

コマンドに対するパラメータの追加や削除はできません。できるのは値の設定だけです。

参照

- DownloadTableData インタフェース [Mobile Link サーバ .NET]584 ページ

FatalException クラス

サーバ側で致命的エラーが内部的に発生したため、直ちに停止する必要があることを Mobile Link に通知します。

Visual Basic の構文

```
Public Class FatalException Inherits System.ApplicationException
```

C# の構文

```
public class FatalException : System.ApplicationException
```

基本クラス

- System.ApplicationException

メンバー

継承されたメンバーを含む FatalException クラスのすべてのメンバー。

名前	説明
FatalException コンストラクタ	デフォルトのメッセージを持つ FatalException を新規に作成します。

FatalException コンストラクタ

デフォルトのメッセージを持つ FatalException を新規に作成します。

オーバロードリスト

名前	説明
FatalException() コンストラクタ	デフォルトのメッセージを持つ FatalException を新規に作成します。
FatalException(string) コンストラクタ	指定されたメッセージを持つ FatalException を新規に作成します。
FatalException(string, Exception) コンストラクタ	指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む FatalException を新規に作成します。

FatalException() コンストラクタ

デフォルトのメッセージを持つ FatalException を新規に作成します。

Visual Basic の構文

```
Public Sub New()
```

C# の構文

```
public FatalException()
```

FatalException(string) コンストラクタ

指定されたメッセージを持つ FatalException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String)
```

C# の構文

```
public FatalException(string message)
```

パラメータ

- **message** この FatalException のメッセージ。

FatalException(string, Exception) コンストラクタ

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む FatalException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# の構文

```
public FatalException(string message, Exception ie)
```

パラメータ

- **message** この FatalException のメッセージ。
- **ie** この FatalException の原因となった例外

LogMessage クラス

ログに出力されたメッセージに関する情報が含まれています。

Visual Basic の構文

```
Public Class LogMessage
```

C# の構文

```
public class LogMessage
```

メンバー

継承されたメンバーを含む LogMessage クラスのすべてのメンバー。

名前	説明
LogMessage コンストラクタ	指定された属性で LogMessage を作成します。
MessageType 列挙体	可能性のあるすべての LogMessage タイプを列挙します。
Text プロパティ	メッセージの本文。
Type プロパティ	このインスタンスが表すログメッセージのタイプ。
User プロパティ	このメッセージがログに記録されるユーザ。

備考

このクラスのインスタンスは、LogCallback に渡されます。

参照

- LogCallback デリゲート [Mobile Link サーバ .NET]619 ページ

LogMessage コンストラクタ

指定された属性で LogMessage を作成します。

Visual Basic の構文

```
Public Sub New(
    ByVal type As MessageType,
    ByVal user As String,
    ByVal text As String
)
```

C# の構文

```
public LogMessage(MessageType type, string user, string text)
```

MessageType 列挙体

可能性のあるすべての LogMessage タイプを列挙します。

Visual Basic の構文

```
Public Enum MessageType
```

C# の構文

```
public enum MessageType
```

メンバー

メンバー名	説明
ERROR	ログメッセージはエラーです。
WARNING	ログメッセージは警告です。
INFO	ログ情報メッセージです。

Text プロパティ

メッセージの本文。

Visual Basic の構文

```
Public ReadOnly Property Text As String
```

C# の構文

```
public string Text {get;}
```

Type プロパティ

このインスタンスが表すログメッセージのタイプ。

Visual Basic の構文

```
Public ReadOnly Property Type As MessageType
```

C# の構文

```
public MessageType Type {get;}
```

User プロパティ

このメッセージがログに記録されるユーザ。

Visual Basic の構文

```
Public ReadOnly Property User As String
```

C# の構文

```
public string User {get;}
```

備考

このプロパティには NULL を指定できます。

NetworkData インタフェース

同期用のネットワークストリームの情報を含みます。

Visual Basic の構文

```
Public Interface NetworkData
```

C# の構文

```
public interface NetworkData
```

メンバー

継承されたメンバーを含む NetworkData インタフェースのすべてのメンバー。

名前	説明
GetHTTPHeaderValue メソッド	指定した名前でサーバが受信した最終ヘッダ値を返します。
GetHTTPHeaderValues メソッド	指定した名前と関連付けられている、サーバが受信したすべてのヘッダ値を返します。
ClientCertificates プロパティ	クライアントから送信されたすべての証明書を含む X509Certificate2Collection を返します。
HTTPHeaders プロパティ	ヘッダ名をヘッダ値のリストとマップする辞書を返します。
IsEndToEndEncrypted プロパティ	同期がエンドツーエンドで暗号化されるか確認します。
IsHTTP プロパティ	同期で HTTP または HTTPS のどちらが使用されるかを確認します。
IsTLS プロパティ	同期で TLS が使用されるかを確認します。

備考

このインタフェースは、クライアント側証明書と HTTP ヘッダを使用するエンタープライズ内の別のサーバを認証する際に便利です。

ネットワークストリームのデータの収集を有効にするには、-x スイッチに collect_network_data=1 を追加します。このオプションにより、同期ごとのデータの保存時に、メモリのオーバヘッドが増えます。TLS または HTTPS をクライアント側証明書で使用する場合に、trusted_certificates=<certificate file> を追加すると、サーバは TLS ハンドシェイクの間に証明書を送信するようクライアントに要求します。これにより、時間とネットワークコストがかかるようになります。

NetworkData オブジェクトは、DBConnectionContext インタフェースの NetworkData メソッドを呼び出して取得できます。HTTP または HTTPS を使用する場合、これには、認証スクリプトが呼び出される前にサーバが受信した、最終の HTTP 要求のヘッダデータが含まれます。

例

次に、DBConnectionContext オブジェクトから NetworkData オブジェクトを取得し、データを出力する例を示します。

```
using iAnywhere.MobiLink.Script;
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;

public class OrderProcessor {
   DBConnectionContext _cc;

   public OrderProcessor( DBConnectionContext cc ) {
      _cc = cc;
   }

   public void AuthUser() {
      NetworkData nd = _cc.NetworkData;
      if( nd != null ) {
         if( nd.IsHTTP ) {
            PrintLn( "http" );
            string user_agent = nd.GetHTTPHeaderValue( "user-agent" );
            PrintLn( "  user-agent: " + user_agent.Substring( 0, user_agent.IndexOf( '/' ) ) );
         } else {
            PrintLn( "no http" );
         }
         if( nd.IsTLS ) {
            PrintLn( "tls" );
            X509Certificate2Collection certs = nd.ClientCertificates;
            if( certs != null ) {
               PrintLn( "  client-side cert:" );
               int n = 1;
               foreach( X509Certificate2 x509 in certs ) {
                  PrintLn( "    cert " + n++ );
                  PrintLn( "      Subject: " + x509.SubjectName.Name );
                  PrintLn( "      Issuer: " + x509.IssuerName.Name );
               }
            } else {
               PrintLn( "  no client cert" );
            }
         } else {
            PrintLn( "no tls" );
         }
         if( nd.IsEndToEndEncrypted ) {
            PrintLn( "e2ee" );
         } else {
            PrintLn( "no e2ee" );
         }
      } else {
         PrintLn( "NULL networkdata" );
      }
   }
}
```

GetHTTPHeaderValue メソッド

指定した名前でサーバが受信した最終ヘッダ値を返します。

Visual Basic の構文

```
Public Function GetHTTPHeaderValue(ByVal name As String) As String
```

C# の構文

```
public string GetHTTPHeaderValue(string name)
```

パラメータ

- **name**　値を返すヘッダの名前

戻り値

指定したヘッダ名に関連付けられている最終ヘッダ値。

参照

- NetworkData.GetHTTPHeaderValues メソッド [Mobile Link サーバ .NET]595 ページ
- NetworkData.HTTPHeaders プロパティ [Mobile Link サーバ .NET]596 ページ

GetHTTPHeaderValues メソッド

指定した名前と関連付けられている、サーバが受信したすべてのヘッダ値を返します。

Visual Basic の構文

```
Public Function GetHTTPHeaderValues(
    ByVal name As String
) As IList< string >
```

C# の構文

```
public IList< string > GetHTTPHeaderValues(string name)
```

パラメータ

- **name**　値を返すヘッダの名前

戻り値

指定したヘッダ名に関連付けられているヘッダ値。

参照

- NetworkData.GetHTTPHeaderValue メソッド [Mobile Link サーバ .NET]595 ページ
- NetworkData.HTTPHeaders プロパティ [Mobile Link サーバ .NET]596 ページ

ClientCertificates プロパティ

クライアントから送信されたすべての証明書を含む X509Certificate2Collection を返します。

Visual Basic の構文

```
Public ReadOnly Property ClientCertificates As
X509Certificate2Collection
```

C# の構文

```
public X509Certificate2Collection ClientCertificates {get;}
```

戻り値

クライアントを識別する X509 証明書を含む X509Certificate2Collection。または、そのような証明書が提供されなかった場合は NULL。

備考

この関数によって NULL 以外の値が返されるのは、isTLS() が true であり、クライアントが「identity」ストリームパラメータを使用して証明書を提供し、サーバ上で trusted_certificates オプションが設定される場合のみです。NULL 以外の CertPath には、自己署名証明書からピアの証明書の順の証明書が含まれます。

参照

- 「.NET 用の trusted_certificates の例」102 ページ

HTTPHeaders プロパティ

ヘッダ名をヘッダ値のリストとマップする辞書を返します。

Visual Basic の構文

```
Public ReadOnly Property HTTPHeaders As IDictionary< string, IList<
string > >
```

C# の構文

```
public IDictionary< string, IList< string > > HTTPHeaders {get;}
```

戻り値

ヘッダ名と値のペアの辞書。

参照

- NetworkData.GetHTTPHeaderValue メソッド [Mobile Link サーバ .NET]595 ページ
- NetworkData.GetHTTPHeaderValues メソッド [Mobile Link サーバ .NET]595 ページ

IsEndToEndEncrypted プロパティ

同期がエンドツーエンドで暗号化されるか確認します。

Visual Basic の構文

```
Public ReadOnly Property IsEndToEndEncrypted As Boolean
```

C# の構文

```
public bool IsEndToEndEncrypted {get;}
```

IsHTTP プロパティ

同期で HTTP または HTTPS のどちらが使用されるかを確認します。

Visual Basic の構文

```
Public ReadOnly Property IsHTTP As Boolean
```

C# の構文

```
public bool IsHTTP {get;}
```

戻り値

同期で HTTP または HTTPS が使用される場合は true、それ以外の場合は false を返します。

IsTLS プロパティ

同期で TLS が使用されるかを確認します。

Visual Basic の構文

```
Public ReadOnly Property IsTLS As Boolean
```

C# の構文

```
public bool IsTLS {get;}
```

戻り値

同期で TLS が使用される場合は true、それ以外の場合は false を返します。

ScriptExecutionException クラス

ユーザスクリプトでエラーが発生したことを通知します。

Visual Basic の構文

```
Public Class ScriptExecutionException
    Inherits System.ApplicationException
```

C# の構文

```
public class ScriptExecutionException : System.ApplicationException
```

基本クラス

- System.ApplicationException

派生クラス

- ServerException クラス [Mobile Link サーバ .NET]604 ページ
- SynchronizationException クラス [Mobile Link サーバ .NET]608 ページ

メンバー

継承されたメンバーを含む ScriptExecutionException クラスのすべてのメンバー。

名前	説明
ScriptExecutionException コンストラクタ	デフォルトのメッセージを持つ ScriptExecutionException を新規に作成します。

備考

この例外または SychronizationException を除く例外の派生が発行されると、Mobile Link サーバは停止します。

ScriptExecutionException コンストラクタ

デフォルトのメッセージを持つ ScriptExecutionException を新規に作成します。

オーバロードリスト

名前	説明
ScriptExecutionException() コンストラクタ	デフォルトのメッセージを持つ ScriptExecutionException を新規に作成します。
ScriptExecutionException(string) コンストラクタ	指定されたメッセージを持つ ScriptExecutionException を新規に作成します。
ScriptExecutionException(string, Exception) コンストラクタ	指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む ScriptExecutionException を新規に作成します。

ScriptExecutionException() コンストラクタ

デフォルトのメッセージを持つ ScriptExecutionException を新規に作成します。

Visual Basic の構文

```
Public Sub New()
```

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

C# の構文

```
public ScriptExecutionException()
```

ScriptExecutionException(string) コンストラクタ

指定されたメッセージを持つ ScriptExecutionException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String)
```

C# の構文

```
public ScriptExecutionException(string message)
```

パラメータ

- **message** この ScriptExecutionException のメッセージ。

ScriptExecutionException(string, Exception) コンストラクタ

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む ScriptExecutionException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# の構文

```
public ScriptExecutionException(string message, Exception ie)
```

パラメータ

- **message** この ScriptExecutionException のメッセージ。
- **ie** この ScriptExecutionException の原因となった例外

ServerContext インタフェース

Mobile Link サーバの継続期間中に存在する、すべてのコンテキストをインスタンス化します。

Visual Basic の構文

```
Public Interface ServerContext
```

C# の構文

```
public interface ServerContext
```

メンバー

継承されたメンバーを含む ServerContext インタフェースのすべてのメンバー。

名前	説明
getProperties メソッド	指定したコンポーネントとセットのプロパティのコレクションを返します。
getPropertiesByVersion メソッド	指定したスクリプトバージョンのプロパティのコレクションを返します。
getPropertySetNames メソッド	指定したコンポーネントのセット名のコレクションを返します。
GetStartClassInstances メソッド	このドメインにロードされたすべての起動クラスを返します。
makeConnection メソッド	新しいデータベース接続を作成します。
Shutdown メソッド	Mobile Link サーバがソフトシャットダウンを実行するようにします。
ErrorListener イベント	Mobile Link サーバがエラーを出力したときにトリガされます。
InfoListener イベント	Mobile Link サーバが情報を出力したときにトリガされます。
ShutdownListener イベント	Mobile Link サーバが停止するときにトリガされます。
WarningListener イベント	Mobile Link サーバが警告を出力したときにトリガされます。

備考

このコンテキストを静的なデータとして保持し、バックグラウンドスレッドで使用できます。Mobile Link で起動される .NET CLR の継続期間中は有効です。

ServerContext インスタンスにアクセスするには、GetServerContext メソッドを使用します。このインスタンスは、起動クラスのコンストラクタに渡されます。

参照

- DBConnectionContext.GetServerContext メソッド [Mobile Link サーバ .NET]565 ページ

getProperties メソッド

指定したコンポーネントとセットのプロパティのコレクションを返します。

Visual Basic の構文

```
Public Function getProperties(
    ByVal component As String,
    ByVal set As String
) As NameValueCollection
```

C# の構文

```
public NameValueCollection getProperties(string component, string set)
```

パラメータ

- **component**　ml_property テーブルの component_name カラムを表します。
- **set**　ml_property テーブルの property_set_name カラムを表します。

戻り値

指定したコンポーネント／セットのプロパティ

備考

プロパティは、ml_property テーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

getPropertiesByVersion メソッド

指定したスクリプトバージョンのプロパティのコレクションを返します。

Visual Basic の構文

```
Public Function getPropertiesByVersion(
    ByVal script_version As String
) As NameValueCollection
```

C# の構文

```
public NameValueCollection getPropertiesByVersion(string script_version)
```

パラメータ

- **script_version**　関連するプロパティを返すスクリプトバージョン

戻り値

指定したスクリプトバージョンのプロパティ

備考

プロパティは、ml_property テーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

getPropertySetNames メソッド

指定したコンポーネントのセット名のコレクションを返します。

Visual Basic の構文

```
Public Function getPropertySetNames(
    ByVal component As String
) As StringCollection
```

C# の構文

```
public StringCollection getPropertySetNames(string component)
```

パラメータ

- **component** ml_property テーブルの component_name カラムを表します。

戻り値

指定したコンポーネントのセット名のコレクション

備考

プロパティは、ml_property テーブルに格納されます。詳細については、「ml_add_property システムプロシージャ」660 ページを参照してください。

GetStartClassInstances メソッド

このドメインにロードされたすべての起動クラスを返します。

Visual Basic の構文

```
Public Function GetStartClassInstances() As Object()
```

C# の構文

```
public object[] GetStartClassInstances()
```

戻り値

サーバ起動時に構築されたすべての起動クラスの配列。起動クラスがない場合、配列の長さは 0 です。

備考

ユーザ定義起動クラスの詳細については、「ユーザ定義起動クラス」539 ページを参照してください。

例

次の例は、起動クラスの検索方法を示します。

```
void FindStartClass( ServerContext sc, string name )
{
    object[] startClasses = sc.GetStartClassInstances();

    foreach( object obj in startClasses ) {
        if( obj is MyClass ) {
            // Execute some code.....
        }
```

```
    }
  }
```

makeConnection メソッド

新しいデータベース接続を作成します。

Visual Basic の構文

```
Public Function MakeConnection() As DBConnection
```

C# の構文

```
public DBConnection MakeConnection()
```

戻り値

新しい接続

Shutdown メソッド

Mobile Link サーバがソフトシャットダウンを実行するようにします。

Visual Basic の構文

```
Public Sub Shutdown()
```

C# の構文

```
public void Shutdown()
```

ErrorListener イベント

Mobile Link サーバがエラーを出力したときにトリガされます。

Visual Basic の構文

```
Public Event ErrorListener As LogCallback
```

C# の構文

```
public event LogCallback ErrorListener;
```

InfoListener イベント

Mobile Link サーバが情報を出力したときにトリガされます。

Visual Basic の構文

```
Public Event InfoListener As LogCallback
```

C# の構文

```
public event LogCallback InfoListener;
```

ShutdownListener イベント

Mobile Link サーバが停止するときにトリガされます。

Visual Basic の構文

```
Public Event ShutdownListener As ShutdownCallback
```

C# の構文

```
public event ShutdownCallback ShutdownListener;
```

WarningListener イベント

Mobile Link サーバが警告を出力したときにトリガされます。

Visual Basic の構文

```
Public Event WarningListener As LogCallback
```

C# の構文

```
public event LogCallback WarningListener;
```

ServerException クラス

サーバでエラーが発生したため直ちに停止する必要があることを Mobile Link に通知します。

Visual Basic の構文

```
Public Class ServerException Inherits ScriptExecutionException
```

C# の構文

```
public class ServerException : ScriptExecutionException
```

基本クラス

- ScriptExecutionException クラス [Mobile Link サーバ .NET]597 ページ

メンバー

継承されたメンバーを含む ServerException クラスのすべてのメンバー。

名前	説明
ScriptExecutionException コンストラクタ	デフォルトのメッセージを持つ ScriptExecutionException を新規に作成します。

名前	説明
ServerException コンストラクタ	デフォルトのメッセージを持つ ServerException を新規に作成します。

ServerException コンストラクタ

デフォルトのメッセージを持つ ServerException を新規に作成します。

オーバロードリスト

名前	説明
ServerException() コンストラクタ	デフォルトのメッセージを持つ ServerException を新規に作成します。
ServerException(string) コンストラクタ	指定されたメッセージを持つ ServerException を新規に作成します。
ServerException(string, Exception) コンストラクタ	指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む ServerException を新規に作成します。

ServerException() コンストラクタ

デフォルトのメッセージを持つ ServerException を新規に作成します。

Visual Basic の構文

```
Public Sub New()
```

C# の構文

```
public ServerException()
```

ServerException(string) コンストラクタ

指定されたメッセージを持つ ServerException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String)
```

C# の構文

```
public ServerException(string message)
```

パラメータ

- **message** この ServerException のメッセージ。

ServerException(string, Exception) コンストラクタ

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む ServerException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# の構文

```
public ServerException(string message, Exception ie)
```

パラメータ

- **message** この ServerException のメッセージ。
- **ie** この ServerException の原因となった例外

SpatialUtilities クラス

空間値を操作する静的メソッドのコレクションを表します。

Visual Basic の構文

```
Public NotInheritable Class SpatialUtilities
```

C# の構文

```
public sealed class SpatialUtilities
```

メンバー

継承されたメンバーを含む SpatialUtilities クラスのすべてのメンバー。

名前	説明
CreateSpatialValue メソッド	ダウンロード用にフォーマットされた空間値を含む新しいバイト配列を返します。最初の 4 バイトにはリトルエンディアンで指定した SRID が含まれ、残りのバイトには指定したバイト配列に渡された空間データが含まれます。
GetBytes メソッド	指定したバイト配列と同じ空間データを含む新しいバイト配列を返しますが、SRID は削除されています。
GetSRID メソッド	指定した空間値の SRID を返します。
SetSRID メソッド	指定した SRID を、指定したバイト配列の最初の 4 バイトに格納します。

CreateSpatialValue メソッド

ダウンロード用にフォーマットされた空間値を含む新しいバイト配列を返します。最初の 4 バイトにはリトルエンディアンで指定した SRID が含まれ、残りのバイトには指定したバイト配列に渡された空間データが含まれます。

Visual Basic の構文

```
Public Shared Function CreateSpatialValue(
    ByVal srid As Integer,
    ByVal spatial_value As Byte()
) As Byte()
```

C# の構文

```
public static byte[] CreateSpatialValue(int srid, byte[] spatial_value)
```

パラメータ

- **srid**　SRID。
- **spatial_value**　空間データ。

戻り値

ダウンロード用にフォーマットされた空間値。

GetBytes メソッド

指定したバイト配列と同じ空間データを含む新しいバイト配列を返しますが、SRID は削除されています。

Visual Basic の構文

```
Public Shared Function GetBytes(ByVal spatial_value As Byte()) As Byte()
```

C# の構文

```
public static byte[] GetBytes(byte[] spatial_value)
```

パラメータ

- **spatial_value**　SRID を削除する必要がある空間値

戻り値

新しいバイト配列

GetSRID メソッド

指定した空間値の SRID を返します。

Visual Basic の構文

```
Public Shared Function GetSRID(ByVal spatial_value As Byte()) As Integer
```

C# の構文

```
public static int GetSRID(byte[] spatial_value)
```

パラメータ

- **spatial_value**　アップロードされた値。最初の 4 バイトには、リトルエンディアンでエンコードされた SRID が含まれている必要があります。

戻り値

SRID。

SetSRID メソッド

指定した SRID を、指定したバイト配列の最初の 4 バイトに格納します。

Visual Basic の構文

```
Public Shared Sub SetSRID(
    ByVal spatial_value As Byte(),
    ByVal srid As Integer
)
```

C# の構文

```
public static void SetSRID(byte[] spatial_value, int srid)
```

パラメータ

- **spatial_value**　SRID を格納する配列。
- **srid**　格納する SRID

SynchronizationException クラス

同期例外が発生したことと、現在の同期をロールバックして再開する必要があることを示します。

Visual Basic の構文

```
Public Class SynchronizationException Inherits ScriptExecutionException
```

C# の構文

```
public class SynchronizationException : ScriptExecutionException
```

基本クラス

- ScriptExecutionException クラス [Mobile Link サーバ .NET]597 ページ

メンバー

継承されたメンバーを含む SynchronizationException クラスのすべてのメンバー。

名前	説明
ScriptExecutionException コンストラクタ	デフォルトのメッセージを持つ ScriptExecutionException を新規に作成します。
SynchronizationException コンストラクタ	デフォルトのメッセージを持つ SynchronizationException を新規に作成します。

SynchronizationException コンストラクタ

デフォルトのメッセージを持つ SynchronizationException を新規に作成します。

オーバロードリスト

名前	説明
SynchronizationException() コンストラクタ	デフォルトのメッセージを持つ SynchronizationException を新規に作成します。
SynchronizationException(string) コンストラクタ	指定されたメッセージを持つ SynchronizationException を新規に作成します。
SynchronizationException(string, Exception) コンストラクタ	指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む SynchronizationException を新規に作成します。

SynchronizationException() コンストラクタ

デフォルトのメッセージを持つ SynchronizationException を新規に作成します。

Visual Basic の構文

```
Public Sub New()
```

C# の構文

```
public SynchronizationException()
```

SynchronizationException(string) コンストラクタ

指定されたメッセージを持つ SynchronizationException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String)
```

C# の構文

```
public SynchronizationException(string message)
```

パラメータ

- **message** この SynchronizationException のメッセージ。

SynchronizationException(string, Exception) コンストラクタ

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む SynchronizationException を新規に作成します。

Visual Basic の構文

```
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

C# の構文

```
public SynchronizationException(string message, Exception ie)
```

パラメータ

- **message** この SynchronizationException のメッセージ。
- **ie** この SynchronizationException の原因となった例外

UpdateDataReader インタフェース

1 つのテーブルの 1 つのアップロードトランザクションのための更新操作を保持します。

Visual Basic の構文

```
Public Interface UpdateDataReader Implements System.Data.IDataReader
```

C# の構文

```
public interface UpdateDataReader : System.Data.IDataReader
```

基本クラス

- System.Data.IDataReader

メンバー

継承されたメンバーを含む UpdateDataReader インタフェースのすべてのメンバー。

名前	説明
Close method (System.Data.IDataReader から継承)	System.Data.IDataReader オブジェクトを閉じます。

名前	説明
GetSchemaTable method (System.Data.IDataReader から継承)	System.Data.IDataReader のカラムメタデータが記述された System.Data.DataTable を返します。
NextResult method (System.Data.IDataReader から継承)	バッチ SQL 文の結果を読み込むときにデータリーダーを次の結果に進めます。
Read method (System.Data.IDataReader から継承)	System.Data.IDataReader を次のレコードに進めます。
SetNewRowValues メソッド	新しいカラム値 (更新後のロー) を返すように、この DataReader のモードを設定します。
SetOldRowValues メソッド	古いカラム値 (更新前のロー) を返すように、この DataReader のモードを設定します。
Depth property (System.Data.IDataReader から継承)	現在のローのネストの深さを示す値を取得します。
IsClosed property (System.Data.IDataReader から継承)	データリーダーが閉じているかどうかを示す値を取得します。
RecordsAffected property (System.Data.IDataReader から継承)	SQL 文の実行によって変更、挿入、または削除されたローの数を取得します。

備考

DataReader のモードを変更することで、新しいローと古いローにアクセスできます。そのほかについては、このインタフェースを通常の DataReader と同様に使用できます。

SetNewRowValues メソッド

新しいカラム値 (更新後のロー) を返すように、この DataReader のモードを設定します。

Visual Basic の構文

```
Public Sub SetNewRowValues()
```

C# の構文

```
public void SetNewRowValues()
```

備考

これがデフォルトモードです。

SetOldRowValues メソッド

古いカラム値 (更新前のロー) を返すように、この DataReader のモードを設定します。

Visual Basic の構文

```
Public Sub SetOldRowValues()
```

C# の構文

```
public void SetOldRowValues()
```

UploadData インタフェース

ダイレクトローハンドリングで使用するアップロード操作をカプセル化します。

Visual Basic の構文

```
Public Interface UploadData
```

C# の構文

```
public interface UploadData
```

メンバー

継承されたメンバーを含む UploadData インタフェースのすべてのメンバー。

名前	説明
GetUploadedTableByName メソッド	アップロードトランザクションの名前付きアップロードテーブルデータを取得します。
GetUploadedTables メソッド	現在のアップロードトランザクションのすべてのアップロードテーブルデータを含む配列を取得します。

備考

アップロードトランザクションには、ロー操作が格納されたテーブルのセットが含まれています。単一のアップロードトランザクションを表す UploadData インスタンスが、handle_UploadData 同期イベントに渡されます。

注意
ダイレクトローハンドリングのアップロード操作は、handle_UploadData イベントに対して登録したメソッドで処理してください。登録されたメソッドを呼び出した後、UploadData は破棄されます。後続のイベントで使用するために新しい UploadData インスタンスを作成しないでください。

UploadedTableData インスタンスを取得するには、UploadData.GetUploadedTables メソッドまたは UploadData.GetUploadedTableByName メソッドを使用します。

リモートデータベースがトランザクションアップロードまたはインクリメンタルアップロードを使用している場合を除き、同期の UploadData は 1 つです。

例

「handle_UploadData 接続イベント」383 ページを参照してください。

GetUploadedTableByName メソッド

アップロードトランザクションの名前付きアップロードテーブルデータを取得します。

Visual Basic の構文

```
Public Function GetUploadedTableByName(
    ByVal table_name As String
) As UploadedTableData
```

C# の構文

```
public UploadedTableData GetUploadedTableByName(string table_name)
```

パラメータ

- **table_name**　アップロードしたデータの取得先テーブルの名前

戻り値

指定したテーブル名のダウンロードしたデータ、見つからなかった場合は NULL

例

handle_UploadData 同期イベントに対して HandleUpload というメソッドを使用すると仮定した場合、次の例は、GetUploadedTableByName メソッドを使用して、remoteOrders テーブルの UploadedTableData インスタンスを返します。

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {
   UploadedTableData uploaded_t1 = ut.GetUploadedTableByName("remoteOrders");
   // ...
}
```

GetUploadedTables メソッド

現在のアップロードトランザクションのすべてのアップロードテーブルデータを含む配列を取得します。

Visual Basic の構文

```
Public Function GetUploadedTables() As UploadedTableData()
```

C# の構文

```
public UploadedTableData[] GetUploadedTables()
```

戻り値

アップロードテーブルデータの配列。配列内でのテーブルの順序は、クライアントのアップロード順と同じです。

備考

テーブルの配列内での順序は、Mobile Link による SQL のローハンドリングでの順序と同じで、参照整合性違反を防ぐ最適な順序になります。データソースがリレーショナルデータベースの場合は、このテーブル順序を使用してください。

参照

- UploadedTableData インタフェース [Mobile Link サーバ .NET]614 ページ

例

handle_UploadData 同期イベントに対して HandleUpload というメソッドを使用すると仮定した場合、次の例は、GetUploadedTables メソッドを使用して、現在のアップロードトランザクションの UploadedTableData インスタンスを返します。

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ud) {
    UploadedTableData[] tables = ud.GetUploadedTables();
    //...
}
```

UploadedTableData インタフェース

1 つのアップロードテーブルの情報を同期用にカプセル化します。

Visual Basic の構文

```
Public Interface UploadedTableData
```

C# の構文

```
public interface UploadedTableData
```

メンバー

継承されたメンバーを含む UploadedTableData インタフェースのすべてのメンバー。

名前	説明
GetDeletes メソッド	このアップロードテーブルデータに対する削除を保持した DataReader を取得します。
GetInserts メソッド	このアップロードテーブルデータに対する挿入を保持した DataReader を取得します。
GetName メソッド	このインスタンスのテーブル名を取得します。

名前	説明
GetSchemaTable メソッド	このダウンロードテーブルのメタデータを記述する DataTable を取得します。
GetUpdates メソッド	このアップロードテーブルデータに対する更新を保持した DataReader を取得します。

備考

挿入、更新、削除の各操作には、標準の ADO.NET IDataReader を使用してアクセスできます。テーブルメタデータへのアクセスには GetSchemaTable 呼び出しまたは挿入および削除データリーダーを使用できます。削除データリーダーには、テーブルのプライマリキーカラムのみが含まれます。

GetDeletes メソッド

このアップロードテーブルデータに対する削除を保持した DataReader を取得します。

Visual Basic の構文

```
Public Function GetDeletes() As IDataReader
```

C# の構文

```
public IDataReader GetDeletes()
```

戻り値

削除するローのプライマリキーカラムを保持した DataReader

備考

個々の削除は、このインスタンステーブルのローを一意に表すプライマリキー値で表されます。

> **注意**
> カラムのインデックスと順序は、このテーブルのスキーマのプロパティ DataTable.PrimaryKey の配列と一致します。

例

リモートクライアントに sparse_pk というテーブルがあるものとします。次の例は GetDeletes メソッドを使用して、削除するローのデータリーダーを取得します。この場合、削除 DataReader には 2 つのプライマリキーカラムが含まれます。各プライマリキーカラムのインデックスに注目してください。

```
CREATE TABLE sparse_pk (
   pcol1  INT NOT NULL,
   col2 VARCHAR(200),
   pcol3 INT NOT NULL,
   PRIMARY KEY (pcol1, pcol3)
);
```

```
using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

   // Get an UploadedTableData for the sparse_pk table.
   UploadedTableData sparse_pk_table = ut.GetUploadedTableByName("sparse_pk");

   // Get deletes uploaded by the MobiLink client.
   using( IDataReader data_reader = sparse_pk_table.GetDeletes() ) {
         while (data_reader.Read()) {
         StringBuilder row_str = new StringBuilder("( ");
         row_str.Append(data_reader.GetString(0)); // pcol1
         row_str.Append(", ");
         row_str.Append(data_reader.GetString(1)); // pcol3
         row_str.Append(" )");
         writer.WriteLine(row_str);
         }
   }
}
```

GetInserts メソッド

このアップロードテーブルデータに対する挿入を保持した DataReader を取得します。

Visual Basic の構文

```
Public Function GetInserts() As IDataReader
```

C# の構文

```
public IDataReader GetInserts()
```

戻り値

このテーブルデータの挿入を保持した DataReader

備考

各挿入は結果セットの 1 つのローで表されています。

例

```
CREATE TABLE sparse_pk (
   pcol1  INT NOT NULL,
   col2 VARCHAR(200),
   pcol3 INT NOT NULL,
   PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
```

```
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

    // Get an UploadedTableData for the sparse_pk table.
    UploadedTableData sparse_pk_table = ut.GetUploadedTableByName("sparse_pk");

    // Get inserts uploaded by the MobiLink client.
    using( IDataReader data_reader = sparse_pk_table.GetInserts() ) {

        while (data_reader.Read()) {
            StringBuilder row_str = new StringBuilder("( ");
            row_str.Append(data_reader.GetString(0)); // pcol1
            row_str.Append(", ");
            if (data_reader.IsDBNull(1)) {
                row_str.Append("&lt;NULL&gt;");
            }
            else {
                row_str.Append(data_reader.GetString(1)); // col2
            }
            row_str.Append(", ");
            row_str.Append(data_reader.GetString(2)); // pcol3
            row_str.Append(" )");
            writer.WriteLine(row_str);
        }
    }
}
```

GetName メソッド

このインスタンスのテーブル名を取得します。

Visual Basic の構文

```
Public Function GetName() As String
```

C# の構文

```
public string GetName()
```

戻り値

このインスタンスのテーブル名。

備考

これはユーティリティ関数です。テーブル名には、このインスタンスのスキーマを使用してもアクセスできます。

GetSchemaTable メソッド

このダウンロードテーブルのメタデータを記述する DataTable を取得します。

Visual Basic の構文

```
Public Function GetSchemaTable() As DataTable
```

C# の構文

```
public DataTable GetSchemaTable()
```

戻り値

カラムメタデータが記述された DataTable。

備考

DataTable にカラム名情報を含める場合は、カラム名を送信するためのクライアント拡張オプションを指定してください。これは、デフォルトの動作です。

GetUpdates メソッド

このアップロードテーブルデータに対する更新を保持した DataReader を取得します。

Visual Basic の構文

```
Public Function GetUpdates() As UpdateDataReader
```

C# の構文

```
public UpdateDataReader GetUpdates()
```

戻り値

このテーブルデータに対する更新を保持した DataReader

備考

結果セット内の各ローは 1 つの更新を表します。結果セットのモードは、新しいカラム値と古いカラム値とに切り替えることができます。

例

次の例に、GetUpdates メソッドの使用方法を示します。

```
CREATE TABLE sparse_pk (
   pcol1  INT NOT NULL,
   col2 VARCHAR(200),
   pcol3 INT NOT NULL,
   PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

   // Get an UploadedTableData for the sparse_pk table.
   UploadedTableData sparse_pk_table = ut.GetUploadedTableByName("sparse_pk");

   // Get updates uploaded by the MobiLink client.
```

```
        using( UpdateDataReader data_reader = sparse_pk_table.GetInserts() ) {

          while (data_reader.Read()) {
            data_reader.SetNewRowValues();
            StringBuilder row_str = new StringBuilder("New values ( ");
            row_str.Append(data_reader.GetString(0)); // pcol1
            row_str.Append(", ");
            if (data_reader.IsDBNull(1)) {
              row_str.Append("&lt;NULL&gt;");
            }
            else {
              row_str.Append(data_reader.GetString(1)); // col2
            }
            row_str.Append(", ");
            row_str.Append(data_reader.GetString(2)); // pcol3
            row_str.Append(" )");
            data_reader.SetOldRowValues();
            row_str.Append(" Old Values (");
            row_str.Append(data_reader.GetString(0)); // pcol1
            row_str.Append(", ");
            if (data_reader.IsDBNull(1)) {
              row_str.Append("&lt;NULL&gt;");
            }
            else {
              row_str.Append(data_reader.GetString(1)); // col2
            }
            row_str.Append(", ");
            row_str.Append(data_reader.GetString(2)); // pcol3
            row_str.Append(" )");
            writer.WriteLine(row_str);
          }
        }
      }
```

LogCallback デリゲート

Mobile Link サーバがメッセージを出力したときに呼び出されます。

Visual Basic の構文

```
Public Delegate Sub LogCallback(
    ByVal sc As ServerContext,
    ByVal message As LogMessage
)
```

C# の構文

```
public delegate void LogCallback(ServerContext sc, LogMessage message);
```

備考

Mobile Link サーバがメッセージを出力したときに呼び出される ServerContext イベントで、このデリゲートの実装を登録できます。

参照

- ServerContext.ErrorListener イベント [Mobile Link サーバ .NET]603 ページ
- ServerContext.InfoListener イベント [Mobile Link サーバ .NET]603 ページ
- ServerContext.WarningListener イベント [Mobile Link サーバ .NET]604 ページ

ShutdownCallback デリゲート

Mobile Link サーバが停止するときに呼び出されます。

Visual Basic の構文

```
Public Delegate Sub ShutdownCallback(ByVal sc As ServerContext)
```

C# の構文

```
public delegate void ShutdownCallback(ServerContext sc);
```

備考

Mobile Link サーバの停止時に呼び出される ShutdownListener イベントで、このデリゲートの実装を登録できます。

参照

- ServerContext.ShutdownListener イベント [Mobile Link サーバ .NET]604 ページ

SQLType 列挙体

可能性のあるすべての ODBC データ型を列挙します。

Visual Basic の構文

```
Public Enum SQLType
```

C# の構文

```
public enum SQLType
```

メンバー

メンバー名	説明
SQL_TYPE_NULL	NULL データ型。
SQL_UNKNOWN_TYPE	不定のデータ型。
SQL_CHAR	設定されたサイズの UTF-8 文字配列。 .NET 型 String を持ちます。
SQL_NUMERIC	設定されたサイズと精度の数値。 .NET 型 decimal を持ちます。
SQL_DECIMAL	設定されたサイズと精度の 10 進数字。 .NET 型 decimal を持ちます。

メンバー名	説明
SQL_INTEGER	32 ビット整数値。 .NET 型 Int32 を持ちます。
SQL_SMALLINT	16 ビット整数値。 .NET 型 Int16 を持ちます。
SQL_FLOAT	ODBC ドライバで精度が定義された浮動小数点数。 .NET 型 Double を持ちます。
SQL_REAL	単精度の浮動小数点数。 .NET 型 Single を持ちます。
SQL_DOUBLE	倍精度浮動小数点数。 .NET 型 Double を持ちます。
SQL_DATE	日付。 .NET 型 DateTime を持ちます。
SQL_DATETIME	日付と時刻。 .NET 型 DateTime を持ちます。
SQL_TIME	時刻。 .NET 型 DateTime を持ちます。
SQL_INTERVAL	時間の間隔。 .NET 型 TimeSpan を持ちます。
SQL_TIMESTAMP	タイムスタンプ。 .NET 型 DateTime を持ちます。
SQL_VARCHAR	ユーザ設定の最大長を持ち、NULL で終了する UTF-8 文字列。 .NET 型 String を持ちます。

メンバー名	説明
SQL_TYPE_DATE	日付。 .NET 型 DateTime を持ちます。
SQL_TYPE_TIME	時刻。 .NET 型 DateTime を持ちます。
SQL_TYPE_TIMESTAMP	タイムスタンプ。 .NET 型 DateTime を持ちます。
SQL_DEFAULT	デフォルト型。 型を持ちません。
SQL_ARD_TYPE	ARD オブジェクト。 型を持ちません。
SQL_BIT	シングルビット。 .NET 型 Boolean を持ちます。
SQL_TINYINT	8 ビット整数値。 .NET SByte 型を持ちます。
SQL_BIGINT	64 ビット整数値。 .NET 型 Int64 を持ちます。
SQL_LONGVARBINARY	ドライバ依存の最大長を持つ可変長バイナリデータ。 .NET 型 byte[] を持ちます。
SQL_VARBINARY	ユーザ指定した最大長を持つ可変長バイナリデータ。 .NET 型 byte[] を持ちます。
SQL_BINARY	固定長のバイナリデータ。 .NET 型 byte[] を持ちます。

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

メンバー名	説明
SQL_LONGVARCHAR	ドライバ依存の最大長を持ち、NULL で終了する UTF-8 文字列。 .NET 型 String を持ちます。
SQL_GUID	GUID。 .NET 型 Guid を持ちます。
SQL_WCHAR	固定サイズの Unicode 文字配列。 .NET 型 String を持ちます。
SQL_WVARCHAR	ユーザ定義の最大長を持ち、NULL で終了するユニコード文字列。.NET 型 String を持ちます。
SQL_WLONGVARCHAR	ドライバ依存の最大長を持ち、NULL で終了するユニコード文字列。.NET 型 String を持ちます。
SQL_SS_TIMESTAMPOFFSET	タイムゾーンオフセットを持つタイムスタンプ。.NET 型 iAnywhere.MobiLink.Script.DateTimeWithTimeZone を持ちます。Microsoft SQL Server と Oracle データベースに対してのみ使用できます。

備考

各 SQLType は、.NET 型に対応しています。

ダイレクトローハンドリング

注意
ダイレクトローハンドリングは、Mobile Link の高度な機能です。この機能を使用するには、Mobile Link アプリケーションの作成方法と Mobile Link API の使用方法を熟知していることが前提です。 次の項を参照してください。

- Mobile Link クイックスタート
- Mobile Link サーバ管理 1 ページ
- Mobile Link クライアント管理

Mobile Link は、SQL とダイレクトの 2 種類のローハンドリングをサポートしています。この 2 つは個別に使用することも併用することもできます。

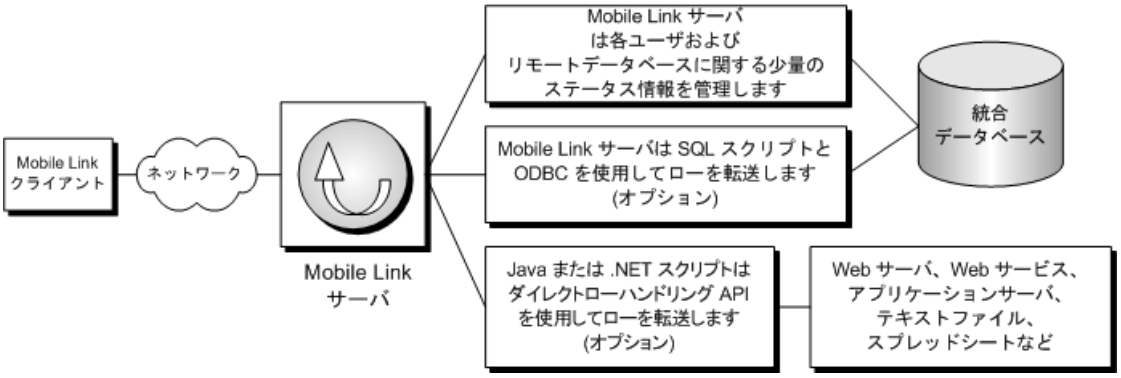
- **SQL ローハンドリング**　リモートデータをサポートされた統合データベースに同期できます。SQL ベースのイベントは、競合の解決などの同期タスクを実行するための堅牢なインタフェースを提供します。SQL を直接使用したり、Java と .NET 用の Mobile Link サーバ API を使用して SQL を返したりできます。

- **ダイレクトローハンドリング**　リモートデータを任意の中央テータソースに同期できます。ダイレクトローハンドリングを使用することで、特別な Mobile Link イベントや Java 用および .NET 用の Mobile Link サーバ API を使用して、同期された未加工のデータにアクセスできます。

  同期可能なデータソースには、アプリケーション、Web サーバ、Web サービス、アプリケーションサーバ、テキストファイル、スプレッドシート、非リレーショナルデータベースなど、ほとんどのものが該当します。また、統合データベースとして使用できない RDBMS も指定できます。ただし、Mobile Link システムテーブルの格納には統合データベースが必要です。また、ダイレクトローハンドリングの数多くの実装が、統合データベースと別のデータソースとの両方に同期します。

  ダイレクトローハンドリングを使用するには、Mobile Link 統合データベースの作成方法、同期スクリプトの追加方法、Mobile Link リモートユーザの作成方法に関する知識が必要です。

次の図は、Mobile Link の基本アーキテクチャを示しています。



ダイレクトローハンドリングのコンポーネント

ダイレクトローハンドリングを実装するには、2 つの同期イベントと、Java と .NET 用の Mobile Link サーバ API のいくつかのインタフェースとメソッドを使用します。

ダイレクト同期イベント

ダイレクトローハンドリングを使用すると、アップロードストリームやダウンロードストリームに直接アクセスできます。これを行うには、handle_UploadData と handle_DownloadData 同期イベントを処理する Java または .NET メソッドを作成します。

- **handle_UploadData**　UploadData パラメータを 1 つ受け取ります。このパラメータは、1 つのアップロードトランザクションに対して Mobile Link クライアントによってアップロードされる操作をカプセル化します。 次の項を参照してください。

  ○「ダイレクトアップロード」627 ページ
  ○「handle_UploadData 接続イベント」383 ページ

- **handle_DownloadData**　DownloadData インタフェースを使用したダウンロード操作を設定します。 次の項を参照してください。

  ○「ダイレクトダウンロード」633 ページ
  ○「handle_DownloadData 接続イベント」371 ページ

ダイレクトローハンドリングに使用する Mobile Link サーバ API のコンポーネント

Java API の場合

- DBConnectionContext.getDownloadData メソッド[Mobile Link サーバ Java]
- DownloadData インタフェース [Mobile Link サーバ Java]
- DownloadTableData インタフェース [Mobile Link サーバ Java]
- UpdateResultSet インタフェース [Mobile Link サーバ Java]
- UploadData インタフェース [Mobile Link サーバ Java]
- UploadedTableData インタフェース [Mobile Link サーバ Java]

.NET API の場合

- DBConnectionContext.GetDownloadData メソッド [Mobile Link サーバ .NET]
- DownloadData インタフェース [Mobile Link サーバ .NET]
- DownloadTableData インタフェース [Mobile Link サーバ .NET]
- UpdateDataReader インタフェース [Mobile Link サーバ .NET]
- UploadedTableData インタフェース [Mobile Link サーバ .NET]
- UploadData インタフェース [Mobile Link サーバ .NET]

ダイレクトローハンドリングの設定

ダイレクトローハンドリングを使用するには、Mobile Link 統合データベースの作成方法、同期スクリプトの追加方法、Mobile Link リモートユーザの作成方法に関する知識が必要です。

次に説明するのは、統合データベース以外のデータソースと同期する方法の概要です。

1. 統合データベースがない場合は、設定します。

   統合データベースと同期するかどうかに関係なく、Mobile Link システムテーブルを保持するためには統合データベースが必要です。

   「Mobile Link 統合データベース」1 ページを参照してください。

2. アップロードを処理するには、UploadData インタフェースを使用してパブリックメソッドを作成し、handle_UploadData 接続イベントに登録します。

「ダイレクトアップロード」627 ページを参照してください。

3. ダウンロードを処理するには、DownloadData インタフェースを使用してパブリックメソッドを作成し、handle_DownloadData 接続イベント (または他のイベント) に登録します。

「ダイレクトダウンロード」633 ページを参照してください。

クイックスタートのためのその他の資料

- 「チュートリアル：ダイレクトローハンドリングの使用」『Mobile Link クイックスタート』
- http://www.sybase.com/detail?id=1058600#319 (このページを表示するには、Sybase.com アカウントが必要です。)
- 「Java 同期論理の設定」
- 「.NET への同期スクリプトの実装」

質問は、次の SQL Anywhere フォーラムに投稿できます。 http://sqlanywhere-forum.sybase.com.

ダイレクトローハンドリングに関する開発のヒント

ユニークなプライマリキー

Mobile Link 同期 (ダイレクトローハンドリングを含む) では、データソースは、更新されないユニークなプライマリキーを必要とします。スプレッドシートやテキストファイルなどの非リレーショナルデータソースの場合、1 つのカラムに、ローを識別するユニークで不変の値が含まれている必要があります。

「ユニークなプライマリキー」117 ページを参照してください。

カラム名

テーブルのカラム名は、常にクライアントから送信され、ダイレクトローハンドリングに使用できます。または、カラムのインデックスを使用すれば、リモートデータベースで設定されたカラム順に基づいて、ロー情報にアクセスできます。

ダウンロードに最終ダウンロード時刻を使用する

可能なかぎり、タイムスタンプベースの SQL アプリケーションのようにダイレクトローハンドリングアプリケーションを設定してください。つまり、last_modified カラムを維持し、そのカラムに基づいてデータをダウンロードします。この方法によって、異なるダウンロード方法を使用した場合に発生する可能性がある予期しない問題を回避できます。

「タイムスタンプベースのダウンロードの実装」108 ページを参照してください。

アップロードのトランザクション管理

Mobile Link 統合データベースを使用してトランザクションをコミットすることはできません。一方、ダイレクトローハンドリングデータソースを使用してトランザクションをコミットすることは可能です。トランザクション管理を設定する場合は、次のヒントを参考にしてください。

- **アップロードは Mobile Link によってコミットされる前にコミットする**　Mobile Link は、アップロードを適用するとき、end_upload イベントの最後で変更をコミットします。保持するすべてのアップロード変更は、end_upload スクリプトが終了する前にコミットしてください。

これを実行しなかった場合、エラーや失敗が発生したときに、アプリケーションはアップロードが適用されたとみなしていても Mobile Link はデータを適用していない状態になり、その結果、データが喪失する可能性があります。

- **冗長なアップロードを処理する**　アップロードされたローをアプリケーションがコミットしてから Mobile Link サーバがコミットするまでにエラーや失敗が発生すると、Mobile Link サーバとデータソースの一貫性が失われる場合があります。この問題を解決するには、冗長なアップロードを許可し、冗長なアップロードが適切に適用されるようにするための論理を用意します。特に、アプリケーションがアップロードをもう一度送信する場合は、アップロードが再度適用されないようにしてください。

エラーの処理

エラーを処理するには、上記のように、適切なトランザクション管理を使用します。また、ローを処理する Java コードおよび .NET コードでは、発生するすべての例外を Mobile Link サーバに送信してください。Mobile Link サーバまたはアプリケーションが変更をコミットする前にエラーが発生した場合、Mobile Link は、トランザクションをロールバックし、アプリケーションとの一貫性がある状態を維持します。

クラスインスタンス

ダイレクトローハンドリングでは、Mobile Link は、データベース接続ごとにクラスインスタンスを 1 つ作成します。クラスインスタンスが破棄されるのは、同期の最後ではなく、データベース接続が閉じられたときです。クラスレベル変数は、以前の同期から値を保持します。

ダイレクトアップロード

ダイレクトアップロードを処理するには

1. handle_UploadData 接続イベント用の Java または .NET メソッドを登録します。
   「handle_UploadData 接続イベント」383 ページを参照してください。

2. handle_UploadData 同期イベント用のメソッドを作成します。このイベントは、UploadData パラメータを 1 つ受け取ります。 次の項を参照してください。

   - UploadData インタフェース [Mobile Link サーバ Java]524 ページ
   - UploadData インタフェース [Mobile Link サーバ .NET]612 ページ

通常、handle_UploadData イベントは、同期のたびに 1 度呼び出されます。ただし、トランザクションレベルアップロードを使用する SQL Anywhere クライアントでは、同期のたびに複数回のアップロードが発生する可能性がありますが、その場合は handle_UploadData をトランザクションごとに 1 回呼び出すことで対応できます。

dbmlsync トランザクションレベルアップロードの詳細については、「-tu dbmlsync オプション」『Mobile Link クライアント管理』を参照してください。

Java または .NET 同期スクリプトの作成に関する一般的な情報については、次の項を参照してください。

● 「Java による同期スクリプトの作成」459 ページ
● 「.NET の同期スクリプト」533 ページ

接続レベルイベントの登録の詳細については、次の項を参照してください。

● 「ml_add_java_connection_script システムプロシージャ」651 ページ
● 「ml_add_dnet_connection_script システムプロシージャ」649 ページ

ダイレクトアップロードに使用するクラス

Java と .NET 用の Mobile Link サーバ API には、ダイレクトアップロードの処理用に次のインタフェースが用意されています。

● **UploadData**　1 つのアップロードトランザクションをカプセル化します。アップロードトランザクションには、ロー操作が格納されたテーブルのセットが含まれています。 次の項を参照してください。

○ UploadData インタフェース [Mobile Link サーバ Java]524 ページ
○ UploadData インタフェース [Mobile Link サーバ .NET]612 ページ

● **UploadedTableData**　Mobile Link クライアントによってアップロードされた、テーブルの挿入、更新、削除操作をカプセル化します。Java の場合、UploadedTableData メソッドは UpdateResultSet のインスタンスを返します。.NET の場合、UploadedTableData メソッドは UpdateDataReader インタフェースのインスタンスを返します。返された結果セット IDataReader を参照して、アップロードされたローに対する操作を処理します。 次の項を参照してください。

○ UploadedTableData インタフェース [Mobile Link サーバ Java]526 ページ
○ UploadedTableData インタフェース [Mobile Link サーバ .NET]614 ページ

● **UpdateResultSet**　Java の場合、このクラスは、UploadedTableData の getUpdates メソッドによって返された更新結果セットを表します。このクラスでは、更新ローの新しいバージョンと古いバージョンを取得するための特別なメソッドが含まれるように java.sql.ResultSet が拡張されています。

UpdateResultSet インタフェース [Mobile Link サーバ Java]522 ページを参照してください。

.NET の場合、UpdateDataReader インタフェースは UploadedTableData GetUpdates メソッドによって返されたローセットを表します。このインタフェースでは、更新ローの新しいバージョンと古いバージョンを取得するための特別なメソッドが含まれるように IDataReader が拡張されています。

UpdateDataReader インタフェース [Mobile Link サーバ .NET]610 ページを参照してください。

例

「handle_UploadData 接続イベント」383 ページを参照してください。

ダイレクトアップロードでの競合

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、更新された値 (更新後または新しいイメージロー) だけでなく、Mobile Link サーバとの最後の同期で得た古いローの値 (更新前または古いイメージロー) のコピーも含まれています。更新前イメージローが中央データソースの現在の値と一致しないと、競合が検出されます。

SQL ベースの競合解決

SQL ベースのアップロードの場合、Mobile Link 統合データベースが中央データソースであり、Mobile Link は競合の検出と解決用に特別なイベントを提供しています。

「競合の解決」124 ページを参照してください。

ダイレクトローハンドリングを使用した競合解決

ダイレクトアップロードの場合、新しいローと古いローにプログラムを使用してアクセスして、競合の検出と解決に使用できます。

UpdateResultSet (UploadedTableData.getUpdates メソッドが返す) は、競合処理に使用する特別なメソッドが含まれるように、Java または .NET 標準の結果セットを拡張したものです。setNewRowValues は、リモートクライアントからの新しい更新された値を返すように UpdateResultSet を設定します (デフォルトモード)。setOldRowValues は、古いロー値を返すように UpdateResultSet を設定します。

ダイレクトローハンドリングを使用した競合検出

UpdateResultSet のメソッド .setOldRowValues を使用することによって、リモートデータベースのローの変更される前の値を取得します。返された値をデータソースの既存のロー値と比較します。比較したローが同じではない場合、競合が存在しています。

ダイレクトローハンドリングを使用した競合解決

アップロード中に競合を検出したら、カスタムビジネス論理を使用して競合を解決できます。競合は、Java コードまたは .NET コードによって処理されます。

例

XML ドキュメント内の在庫を追跡していて、XML ドキュメントを中央データソースとして使用すると仮定します。User1 は、リモートデータベースの 1 つである Remote1 を使用します。User2 は、別のデータベースである Remote2 を使用します。

XML ドキュメント、User1、User2 の最初の在庫数はすべて 10 個です。User1 が、3 個を販売し、Remote1 にある在庫数を 7 個に更新します。User2 は 4 個販売し、Remote2 にある在庫数を 6 個に更新します。Remote1 が同期を実行すると、中央データベースは 7 個に更新されます。Remote2 が同期を実行すると、在庫の値が 10 個ではなくなっているため、競合が検出されます。この競合をプログラムで解決するには、次のような 3 つのロー値が必要となります。

- 中央データソースにある現在の値。
- Remote2 がアップロードした新しいローの値。

●Remote2 が最後の同期中に取得した古いローの値。

この場合、ビジネス論理は新しい在庫数を計算し、競合を解決するために次の式を使用します。

```
current data source - (old remote - new remote)
-> 7 - (10-6) = 3
```

Java と .NET の次のプロシージャでは、ダイレクトアップロードでのこのような競合を解決する方法を、次のテーブルを例に示しています。

```
CREATE TABLE remoteOrders
(
   pk integer primary key not null,
   inventory integer not null
);
```

Java

1. handle_UploadData 接続イベント用の Java メソッドを登録します。

   「handle_UploadData 接続イベント」383 ページを参照してください。

   たとえば、次のストアドプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して HandleUpload という Java メソッドを登録します。Mobile Link 統合データベースに対してこのストアドプロシージャを実行します。

   ```
   call ml_add_java_connection_script( 'ver1',
      'handle_UploadData',
      'OrderProcessor.HandleUpload' )
   ```

   同期イベント用のメソッド登録の詳細については、次の項を参照してください。

   ○「スクリプトの追加と削除」256 ページ
   ○「ml_add_java_connection_script システムプロシージャ」651 ページ

2. アップロード内のテーブルの UpdateResultSet を取得します。

   OrderProcessor.HandleUpload メソッドは、remoteOrders テーブルの UpdateResultSet を取得します。

   ```
   // method for handle_UploadData event
    public void HandleUpload( UploadData u_data )
    {

      // Get UploadedTableData for the remoteOrders table.
        UploadedTableData u_table = u_data.getUploadedTableByName("remoteOrders");

      // Get an UpdateResultSet for the remoteOrders table.
      UpdateResultSet update_rs = u_table.getUpdates();

      // (Continued...)
   ```

3. 更新ごとに、中央データソースの現在の値を取得します。

   次の例では、UpdateResultSet の getInt メソッドが、プライマリキーカラム (先頭カラム) の整数値を返します。getMyCentralData メソッドを実装し、使用することにより、中央データソースからデータを取得できます。

```
while( update_rs.next() )
{
  // Get central data source values.

  // Get the primary key value.
  int pk_value = update_rs.getInt(1);

  // Get central data source values.
  int central_value = getMyCentralData(pk_value);

  // (Continued...)
```

4. 更新ごとに、Mobile Link クライアントによってアップロードされた古い値と新しい値を取得します。

   次の例では、UpdateResultSet の setOldRowValues と setNewRowValues を使用して、それぞれ古い値と新しい値を取得しています。

```
// Set mode for old row values.
update_rs.setOldRowValues();

// Get the _old_ stored value on the remote.
int old_value = update_rs.getInt(2);

// Set mode for new row values.
update_rs.setNewRowValues();

// Get the _new_ updated value on the remote.
int new_value = update_rs.getInt(2);

// (Continued...)
```

5. 更新ごとに、競合がないかどうかを確認します。

   古いロー値が中央データソースの現在の値と一致しない場合、競合になります。競合を解決するため、ビジネス論理を使用して解決後の値を計算します。競合がなかった場合、中央データソースは新しいリモート値で更新されます。setMyCentralData メソッドを実装し、使用することにより、更新を実行します。

```
// Check if there is a conflict.

if(old_value == central_value)
{
  // No conflict.
  setMyCentralData(pk_value, new_value);

}
else
{
  // Handle the conflict.
  int inventory = old_value - new_value;
  int resolved_value = central_value - inventory;

  setMyCentralData(pk_value, resolved_value);

 }
}
```

.NET

1. handle_UploadData 接続イベント用のメソッドを登録します。

   たとえば、次のストアドプロシージャコールは、スクリプトバージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して HandleUpload という .NET メソッドを登録します。Mobile Link 統合データベースに対してこのストアドプロシージャを実行します。

   ```
   call ml_add_dnet_connection_script( 'ver1',
     'handle_UploadData',
     'MyScripts.OrderProcessor.HandleUpload' )
   ```

   同期イベント用のメソッド登録の詳細については、次の項を参照してください。

   ○「スクリプトの追加と削除」256 ページ
   ○「ml_add_dnet_connection_script システムプロシージャ」649 ページ

2. アップロード内のテーブルの UpdateDataReader を取得します。

   MyScripts.OrderProcessor.HandleUpload メソッドは、remoteOrders テーブルの UpdateResultSet を取得します。

   ```
   // method for handle_UploadData event
    public void HandleUpload( UploadData u_data )
    {

      // Get UploadedTableData for the remoteOrders table.
        UploadedTableData u_table = u_data.GetUploadedTableByName("remoteOrders");

      // Get an UpdateDataReader for the remoteOrders table.
      UpdateDataReader update_dr = u_table.GetUpdates();

      // (Continued...)
   ```

3. 更新ごとに、中央データソースの現在の値を取得します。

   次の例では、UpdateDataReader の GetInt32 メソッドが、プライマリキーカラム (先頭カラム) の整数値を返します。getMyCentralData メソッドを実装し、使用することにより、中央データソースからデータを取得できます。

   ```
   while( update_dr.Read() )
   {
     // Get central data source values.

     // Get the primary key value.
     int pk_value = update_dr.GetInt32(0);

     // Get central data source values.
     int central_value = getMyCentralData(pk_value);

     // (Continued...)
   ```

4. 更新ごとに、Mobile Link クライアントによってアップロードされた古い値と新しい値を取得します。

   次の例では、UpdateResultSet の setOldRowValues と setNewRowValues を使用して、それぞれ古い値と新しい値を取得しています。

   ```
     // Set mode for old row values.
     update_dr.SetOldRowValues();
   ```

```
// Get an _old_ value.
int old_value = update_dr.GetInt32(1);

// Set mode for new row values.
update_dr.SetNewRowValues();

// Get the _new_ updated value.
int new_value = update_dr.GetInt32(1);

// (Continued...)
```

5. 更新ごとに、競合がないかどうかを確認します。

   古いロー値が中央データソースの現在の値と一致しない場合、競合になります。競合を解決するため、ビジネス論理を使用して解決後の値を計算します。競合がなかった場合、中央データソースは新しいリモート値で更新されます。setMyCentralData メソッドを実装し、使用することにより、更新を実行します。

```
// Check if there is a conflict.

if(old_value == central_value)
{
  // No conflict.
  setMyCentralData(pk_value, new_value);

}
else
{
  // Handle the conflict.
  int inventory = old_value - new_value;
  int resolved_value = central_value - inventory;

  setMyCentralData(pk_value, resolved_value);

 }
}
```

ダイレクトダウンロード

ダイレクトダウンロードを処理するには

1. handle_DownloadData 接続イベント用の Java または .NET メソッドを登録します。
   「handle_DownloadData 接続イベント」371 ページを参照してください。

2. handle_DownloadData 同期イベント用のメソッドを作成します。このイベントでは、DBConnectionContext のインスタンスを使用して、現在の同期に対する DownloadData インスタンスを取得します。 次の項を参照してください。

   - DBConnectionContext インタフェース [Mobile Link サーバ Java]473 ページ
   - DownloadData インタフェース [Mobile Link サーバ Java]478 ページ
   - DBConnectionContext インタフェース [Mobile Link サーバ .NET]561 ページ
   - DownloadData インタフェース [Mobile Link サーバ .NET]582 ページ

handle_DownloadData 同期イベントでダイレクトダウンロード全体を作成できます。また、他の同期イベントを使用してダイレクトダウンロード操作を設定することもできます。ただし、

handle_DownloadData スクリプトを (そのメソッドが何も処理しない場合でも) 作成する必要があります。ダイレクトダウンロードを handle_DownloadData 以外のイベントで処理する場合、そのイベントは begin_synchronization より前または end_download より後に実装できません。

イベントの順序の詳細については、「Mobile Link の完全なイベントモデル」274 ページを参照してください。

ダイレクトダウンロードに使用するクラス

Java と .NET 用の Mobile Link サーバ API には、ダイレクトダウンロードの作成用に次のクラスが用意されています。

- **DownloadData**　同期中にリモートクライアントへ送信する操作を含むダウンロードテーブルをカプセル化します。 次の項を参照してください。

  - DownloadData インタフェース [Mobile Link サーバ Java]478 ページ
  - DownloadData インタフェース [Mobile Link サーバ .NET]582 ページ

- **DownloadTableData**　Mobile Link クライアントにダウンロードするアップサート (更新と挿入) と削除操作をカプセル化します。

  Java の場合、DownloadTableData メソッドは JDBC PreparedStatement のインスタンスを返します。Java の場合、ダウンロードにローを追加するには、準備文のカラム値を設定してから、準備文を実行します。

  .NET の場合、DownloadTableData メソッドは .NET IDbCommand のインスタンスを返します。.NET の場合、ダウンロードにローを追加するには、コマンドのカラム値を設定してから、コマンドを実行します。

  次の項を参照してください。

  - DownloadTableData インタフェース [Mobile Link サーバ Java]481 ページ
  - DownloadTableData インタフェース [Mobile Link サーバ .NET]584 ページ

例

「handle_DownloadData 接続イベント」371 ページを参照してください。

Mobile Link リファレンス

この項では、Mobile Link リファレンス情報を示します。

Mobile Link Replay C++ コールバック

このセクションでは、mlgenreplayapi ユーティリティよって生成される *mlreplaycallbacks.cpp* ファイルのパブリック構造について説明します。ここでは、mlgenreplay ユーティリティを使用したリプレイセッション中に Mobile Link サーバにアップロードされるデータをカスタマイズするために作成できるコールバックメソッドの完全なリストを示します。

注意
コールバックを使用しない場合には、すべてのシミュレートクライアントがリプレイ可能な状態になり、実際にリプレイを開始するまで待機する必要があります。いずれかのシミュレートクライアントの作成または初期化に失敗した場合には、シミュレートクライアントはリプレイを実行できません。

参照

- 「Mobile Link Replay ユーティリティ (mlreplay)」696 ページ

CreateAndInitMLReplayUploadTransaction コールバック

アップロードトランザクションの作成と初期化に使用します。最初に 1 回呼び出されると、その後はアップロードトランザクション、同期、シミュレートクライアントごとに 1 回呼び出され、さらに 2 回以上のすべての繰り返しに対しても 1 回だけ呼び出されます。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL
CreateAndInitMLReplayUploadTransaction(
    IMLReplayUploadTransaction ** uploadTrans,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **uploadTrans**　mlreplay によってリプレイセッションへのカスタムデータの取り込みに使用される IMLReplayUploadTransaction の実装。
- **mlrAPICallbacks**　mlreplay からの情報を提供するコールバック

戻り値

成功の場合は true、エラーの場合は false (リプレイセッションはキャンセルされます)。

備考

このコールバックは変更しないでください。

DelayCreationOfSimulatedClient コールバック

指定したシミュレートされたクライアントの番号とシミュレートされたクライアントの数に基づいて各シミュレートされたクライアントを作成するタイミングを調整する場合に使用できます。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL DelayCreationOfSimulatedClient(
    asa_uint32 simulatedClientNum,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

指定したシミュレートされたクライアントが作成されていると想定される場合は true を返し、指定したシミュレートされたクライアントが作成されていない場合は false を返します。

備考

シミュレートクライアント *X* は、次の条件の基に作成されます。

●シミュレートクライアント 1, ..., *X* - 1 について返した DelayCreationOfSimulatedClient。

●DelayCreationOfSimulatedClient(*X*, *mlrAPICallbacks*) が true を返しています。

このコールバックが false を返したときはシミュレートクライアントは作成されませんが、他のシミュレートクライアントについては作成されます。このコールバックは mlreplay インスタンスのそれぞれのシミュレートクライアントに対して 1 回だけ呼び出されます。

DelayDestructionOfSimulatedClient コールバック

各シミュレートされたクライアントが破壊されるタイミングを調整するのに使用できます。それぞれの mlreplay インスタンスのシミュレートクライアントごとに 1 回だけ呼び出されます。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL DelayDestructionOfSimulatedClient(
    asa_uint32 simulatedClientNum,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

パラメータ

- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

備考

このコールバックは、同時に呼び出すことができます。

DelayDestructionOfSimulatedClient(*X*, *mlrAPICallbacks*) が返されるまで、シミュレートクライアント *X* は破壊されません。

DelayStartOfReplay コールバック

リプレイが開始されるタイミングを調整するのに使用できます。それぞれの mlreplay インスタンスのシミュレートクライアントごとの繰り返しに対して 1 回だけ呼び出されます。

構文

```
bool DelayStartOfReplay(
    asa_uint32 repetitionNum
    uint32 simulatedClientNum,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **repetitionNum**　現在の繰り返し数。
- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

リプレイを開始する場合は true、生成をスキップする場合は false です。

備考

このコールバックは、同時に呼び出すことができます。

繰り返し *X* で false が返された場合は、その繰り返しはスキップされます。

DestroyMLReplayUploadTransaction コールバック

アップロードトランザクションを解体するために使用します。まず、CreateAndInitMLReplayUploadTransaction の最初の呼び出しの後で 1 回呼び出され、次にアップロードトランザクション、同期、シミュレートクライアント、および繰り返しごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT void _MLREPLAY_CDECL DestroyMLReplayUploadTransaction(
    IMLReplayUploadTransaction * uploadTrans
)
```

パラメータ

- **uploadTrans**　mlreplay ユーティリティによってリプレイセッションへのカスタムデータの取り込みに使用される IMLReplayUploadTransaction の実装。

備考

このコールバックは変更しないでください。

FiniIdentifySimulatedClient コールバック

指定されたシミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって使用されたメモリをクリーンアップするために使用します。シミュレートクライアントごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT void _MLREPLAY_CDECL FiniIdentifySimulatedClient(
    asa_uint32 simulatedClientNum,
    char * remoteID,
    char * username,
    char * password,
    char * scriptVersion,
    char ** authenticationParameters,
    asa_uint16 numAuthenticationParameters,
    char * ldt,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。

- **remoteID**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定されたリモート ID。

- **username**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定されたユーザ名。

- **password**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定されたパスワード。
- **scriptVersion**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定されたスクリプトバージョン。
- **authenticationParameters**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定された認証パラメータ。
- **numAuthenticationParameters**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定された認証パラメータの数。
- **ldt**　指定された、シミュレートクライアントに対する IdentifySimulatedClient 呼び出しによって指定されたダウンロード時間。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

備考

リプレイセッションに関する一般ユーザ名、パスワード、およびリモート ID を使用するときに実装できるように、このコールバックには生成時にコメントアウトされたコードが残されています。使用されたメモリがあれば、コードによって解放されます。

GetDownloadApplyTime コールバック

低速のデバイスをシミュレートするために使用します。ダウンロード、シミュレートクライアント、繰り返しごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT asa_uint32 _MLREPLAY_CDECL GetDownloadApplyTime(
    asa_uint32 repetitionNum,
    asa_uint32 simulatedClientNum,
    asa_uint32 recordedSyncNum,
    asa_uint32 recordedDownloadApplyTime,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **repetitionNum**　現在の繰り返し数。
- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。
- **recordedSyncNum**　記録されたプロトコル内での同期番号 (序数 1)。
- **recordedDownloadApplyTime**　記録されたダウンロード適用時間 (ミリ秒単位)。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

ダウンロードの適用に要するミリ秒数。

備考

このコールバックは、同時に呼び出すことができます。

mlreplay ユーティリティは、永続的接続では発生しない同期のダウンロード適用時間を見積るのに非常に効果的です。永続的接続で発生する同期では、ダウンロード確認を使用しないかぎり、ダウンロード適用時間を mlreplay で正確に見積ることはできません。

GetMLReplayAPIVersion コールバック

リプレイ API バージョンを返すために使用されます。

構文

```
_MLREPLAY_EXPORT asa_uint32 _MLREPLAY_CDECL GetMLReplayAPIVersion( void )
```

備考

このコールバックは mlreplay インスタンスごとに 1 回呼び出され、変更することはできません。

GetUploadTransaction コールバック

リプレイセッション中に Mobile Link サーバにアップロードされるローをカスタマイズするために使用します。アップロードトランザクション、同期、シミュレートクライアント、繰り返しごとに 1 回呼び出されます。

> **注意**
> mlreplay ユーティリティは、GetUploadTransaction によって指定された新しいアップロードのサイズと記録済みのプロトコルファイルのアップロードに基づいて、タイミング情報を調整しようとします。ただし、タイミングを重視する場合には、GetUploadTransaction を使用して追加されたローのサイズの合計と最初の記録済み同期のアップロードされたローのサイズとがほぼ一致する場合に、最良の結果が得られます。サイズがほぼ同じであることを確認するための最も簡単な方法は、類似するデータを使用して同じロー数をアップロードした同期を記録しておくことです。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL GetUploadTransaction(
    asa_uint32 repetitionNum,
    asa_uint32 simulatedClientNum,
    asa_uint32 recordedSyncNum,
    asa_uint32 uploadTransNum,
    asa_uint32 numUploadedTrans,
    IMLReplayUploadTransaction * uploadTrans,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **repetitionNum**　現在の繰り返し数。
- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。
- **recordedSyncNum**　記録されたプロトコル内での同期番号 (序数 1)。
- **uploadTransNum**　指定された同期内でのトランザクション番号 (序数 1)。
- **numUploadedTrans**　指定された同期内のアップロードトランザクションの総数。
- **uploadTrans**　現在のトランザクションに対してアップロードトランザクションで設定する必要のある出力パラメータ。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

成功の場合は true、エラーの場合は false を返します。最初の繰り返しより前に GetUploadTransaction がエラーになった場合、リプレイセッションはキャンセルされます。最初の繰り返し以外の繰り返しより前に GetUploadTransaction がエラーになった場合は、エラーになったシミュレートクライアントのみが終了します。

備考

記録済みのプロトコルファイルがリプレイされたときに表示される同期およびアップロードトランザクションの数に基づいて、複数回呼び出される場合があります。

このコールバックは同時に呼び出すことができますが、複数の同時呼び出しで、同じ uploadTrans オブジェクトのポインタを共有しているわけではありません。

GlobalFini コールバック

他のコールバックで使用されたグローバル変数をクリーンアップするために使用します。mlreplay インスタンスごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT void _MLREPLAY_CDECL GlobalFini(
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

GlobalInit コールバック

他のコールバックで使用されたグローバル変数を初期化するために使用します。mlreplay インスタンスごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL GlobalInit(
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

成功の場合は true、エラーの場合は false (リプレイセッションはキャンセルされます)。

IdentifySimulatedClient コールバック

シミュレートクライアントの情報を指定するために使用します。シミュレートクライアントごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL IdentifySimulatedClient(
    asa_uint32 simulatedClientNum,
    char ** remoteID,
    char ** username,
    char ** password,
    char ** scriptVersion,
    char *** authenticationParameters,
    asa_uint16 * numAuthenticationParameters,
    char ** ldt,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。

- **remoteID**　シミュレートされたこのクライアントのリモート ID に設定する必要のある出力パラメータ (すべての mlreplay インスタンスでユニークな値にする必要があります)。

- **username**　シミュレートされたこのクライアントの Mobile Link ユーザ名に設定する必要のある出力パラメータ。

- **password**　Mobile Link ユーザのパスワードに設定する必要のある出力パラメータ。

- **scriptVersion**　Mobile Link ユーザが使用するスクリプトバージョンに設定する必要のある出力パラメータ。
- **authenticationParameters**　シミュレートされたこのクライアントの認証パラメータの配列に設定する必要のある出力パラメータ。
- **numAuthenticationParameters**　authenticationParameters で返される認証パラメータの数に設定される出力パラメータ。
- **ldt**　ユーザの最終ダウンロード時刻 (LDT) に設定される出力パラメータ。ldt のフォーマットは yyyy-MM-dd hh:mm:ss.SSS であることが必要です。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

成功の場合は true、エラーの場合は false (リプレイセッションはキャンセルされます)。

備考

ユーザ名、パスワード、authenticationParameters、scriptVersion、または ldt が null の場合は、記録されたプロトコルで対応する値が mlreplay で使用されます。remoteID が null の場合、シミュレートクライアントの GUID に remote ID が置き換えられます。

リプレイセッションに関する一般ユーザ名、パスワード、およびリモート ID を使用するときに実装できるように、このコールバックには生成時にコメントアウトされたコードが記述されています。コードによって、**user_** *シミュレートクライアント番号*、, **pwd_** *シミュレートクライアント番号*、および **rid_** *シミュレートクライアント番号*のそれぞれユーザ名、パスワード、リモート ID が作成されます。

ReportEndOfReplay コールバック

シミュレートクライアントがリプレイの完了に必要とするアクションを実行するために使用します。シミュレートクライアント、繰り返しごとに 1 回呼び出されます。

構文

```
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL ReportEndOfReplay(
    asa_uint32 repetitionNum,
    asa_uint32 simulatedClientNum,
    bool success,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

パラメータ

- **repetitionNum**　現在の繰り返し数。
- **simulatedClientNum**　このシミュレートクライアントを同じ mlreplay インスタンス内の他のシミュレートクライアントと区別するために使用される、シミュレートクライアント番号 (序数 1)。

- **success**　シミュレートクライアントが正常に完了した場合は true、それ以外の場合は false。
- **mlrAPICallbacks**　リプレイ動作のカスタマイズに使用できる mlreplay からの情報を提供するコールバック。

戻り値

成功の場合は true、エラーの場合は false。

備考

このコールバックを使用して、*X* 回目の繰り返しでデータが正しくアップロードされたかどうかを確認できます。

このコールバックでは、指定したリプレイの結果を判定します。success が false の場合は、このコールバックはリプレイの成功に影響を与えません。success が true であるのに、このコールバックが false を返した場合には、mlreplay では、シミュレートクライアントが指定した繰り返しでエラーになったものと見なします。このコールバックは、同時に呼び出すことができます。

Mobile Link サーバシステムプロシージャ

Mobile Link には、アプリケーションの作成に役立つ、以下のストアドプロシージャが用意されています。

スクリプトを追加または削除するためのシステムプロシージャ

同期スクリプトは、統合データベース内のシステムテーブルに追加しないと使用できません。次のシステムプロシージャを使用して、同期スクリプトを統合データベースに追加するか、または統合データベースから削除します。

- 「ml_add_connection_script システムプロシージャ」
- 「ml_add_table_script システムプロシージャ」
- 「ml_add_dnet_connection_script システムプロシージャ」
- 「ml_add_dnet_table_script システムプロシージャ」
- 「ml_add_java_connection_script システムプロシージャ」
- 「ml_add_java_table_script システムプロシージャ」

Java または .NET 用 Mobile Link サーバ API を使用する場合は、これらのシステムプロシージャを使用して、イベント用のスクリプトとしてメソッドを登録して、イベントが発生したときにメソッドが実行されるようにします。これらのシステムプロシージャを使用して、メソッドを登録解除することもできます。

システムプロシージャを使用してスクリプトを追加する場合、スクリプトは 1 つの文字列になります。スクリプト内の文字列はすべて、エスケープする必要があります。SQL Anywhere の場合、文字列が切れないように各引用符 (') を 2 つ重ねる必要があります。

Adaptive Server Enterprise 11.5 以前のバージョンに 255 バイトを超えるスクリプトを追加する場合は、システムプロシージャを使用できません。長いスクリプトを定義する場合、Sybase Central を使用するか、直接挿入します。

バージョン 6 より前の IBM DB2 LUW では、カラム名とその他の識別子は 18 文字までしかサポートされないため、名前がトランケートされます。たとえば、ml_add_connection_script は ml_add_connection_ に短縮されます。

リモートタスクを管理するためのシステムプロシージャ

次のストアドプロシージャを使用して、リモートタスクを管理できます。

- 「ml_ra_add_agent_id システムプロシージャ」
- 「ml_ra_assign_task システムプロシージャ」
- 「ml_ra_cancel_notification システムプロシージャ」
- 「ml_ra_cancel_task_instance システムプロシージャ」
- 「ml_ra_clone_agent_properties システムプロシージャ」
- 「ml_ra_delete_agent_id システムプロシージャ」
- 「ml_ra_delete_events_before システムプロシージャ」
- 「ml_ra_delete_remote_id システムプロシージャ」
- 「ml_ra_delete_task システムプロシージャ」
- 「ml_ra_get_agent_events システムプロシージャ」
- 「ml_ra_get_agent_ids システムプロシージャ」
- 「ml_ra_get_agent_properties システムプロシージャ」
- 「ml_ra_get_latest_event_id システムプロシージャ」
- 「ml_ra_get_orphan_taskdbs システムプロシージャ」
- 「ml_ra_reassign_taskdb システムプロシージャ」
- 「ml_ra_get_remote_ids システムプロシージャ」
- 「ml_ra_get_task_results システムプロシージャ」
- 「ml_ra_get_task_status システムプロシージャ」
- 「ml_ra_manage_remote_db システムプロシージャ」
- 「ml_ra_notify_agent_sync システムプロシージャ」
- 「ml_ra_set_agent_property システムプロシージャ」
- 「ml_ra_unmanage_remote_db システムプロシージャ」

LDAP システムプロシージャ

次のストアドプロシージャを使用して LDAP 認証を設定および管理することができます。

- 「ml_add_certificates_file システムプロシージャ」646 ページ
- 「ml_add_ldap_server システムプロシージャ」654 ページ
- 「ml_add_user_auth_policy システムプロシージャ」665 ページ

同期モデルシステムプロシージャ

次のストアドプロシージャを使用して、スキーマのアップグレードを管理できます。

- 「ml_model_drop システムプロシージャ」671 ページ
- 「ml_model_check_all_schema システムプロシージャ」671 ページ
- 「ml_model_check_version_schema システムプロシージャ」673 ページ

その他のシステムプロシージャ

- 「ml_add_property システムプロシージャ」
- 「ml_delete_sync_state_before システムプロシージャ」
- 「ml_reset_sync_state システムプロシージャ」

ml_add_certificates_file システムプロシージャ

LDAP 認証で TLS を使用する場合に信頼できる証明書を設定します。

構文

ml_add_certificates_file (
'*file_name*',
)

パラメータ

構文	説明
file_name	VARCHAR(1024)。信頼できる証明書ファイルの名前。

備考

このプロシージャは、指定した信頼できる証明書ファイルに関する情報を ml_trusted_certificates_file テーブルに移植します。

サーバファームに必要な信頼できる証明書ファイルは 1 つだけなので、新しい信頼できる証明書ファイルを挿入する場合は ml_trusted_certificates_file テーブルの既存のエントリを事前に削除します。

参照

- 「ml_add_ldap_server システムプロシージャ」 654 ページ
- 「ml_add_user_auth_policy システムプロシージャ」 665 ページ

ml_add_column システムプロシージャ (廃止予定)

リモートデータベースのカラムに関する情報を登録します。この情報は名前付きカラムパラメータで使用されます。

構文

ml_add_column (
'*version*',
'*table*',
'*column*',
'*type*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
column	VARCHAR(128)。カラムの名前。
type	VARCHAR(128)。今後の使用のために予約されています。NULL に設定されます。

備考

このプロシージャは、リモートデータベースのカラムに関する情報を Mobile Link システムテーブル ml_column に設定します。この情報は名前付きローパラメータで使用されます。

警告
ml_add_column の呼び出しは、リモートデータベーステーブルでのカラム順と同じ順序で実行される必要があります。そうしないと、データが正しくなくなることがあります。

同期クライアントがカラム名を「送信しない」場合に、このシステムプロシージャを実行する必要があります。デフォルトでは、バージョン 12 以降のクライアントはカラム名を「送信する」ため、ほとんどの展開では ml_add_column は不要です。ml_add_column の名前は、常にクライアントから送信された名前を上書きします。

特定のスクリプトバージョン内のテーブル名のエントリをすべて削除するには、カラム名を NULL に設定します。

参照

● 「スクリプトのパラメータ」234 ページ

例

次のストアドプロシージャコールは、スクリプトバージョン Version1 で、MyTable の col1 に関する情報を Mobile Link システムテーブル ml_column に設定します。同期クライアントがカラム名を送信しない場合は (バージョン 12 より前のクライアントのデフォルト動作)、この呼び出しによって、Version1 スクリプトバージョンで、MyTable1 のテーブルスクリプトに名前付きローパラメータ r.col1 と o.col1 を使用できます。

```
CALL ml_add_column( 'Version1', 'MyTable1', 'col1', NULL )
```

次のストアドプロシージャコールは、スクリプトバージョン Version1 で、Mobile Link システムテーブル ml_column 内の MyTable1 のエントリをすべて削除します。

```
CALL ml_add_column( 'Version1', 'MyTable1', NULL, NULL )
```

ml_add_connection_script システムプロシージャ

このシステムプロシージャを使用して、SQL 接続スクリプトを統合データベースに追加するか、または統合データベースから削除します。

構文

ml_add_connection_script (
'*version*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。SAP HANA および Oracle の場合は CLOB。

備考

接続スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトを追加すると、スクリプトが ml_script テーブルに挿入され、このスクリプトを指定のイベントとスクリプトバージョンに関連付ける適切な参照が定義されます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ

例

次の文は、begin_synchronization イベントと関連付けられた接続スクリプトを SQL Anywhere 統合データベースのスクリプトバージョン custdb に追加します。追加されるスクリプト自体は、@EmployeeID 変数を設定する単一の文です。

```
call ml_add_connection_script( 'custdb',
 'begin_synchronization',
 'set @EmployeeID = {ml s.username}' )
```

ml_add_dnet_connection_script システムプロシージャ

このシステムプロシージャを使用して、.NET メソッドを接続イベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_dnet_connection_script (
'*version*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。SAP HANA および Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトの内容の値は、たとえば MyClass.MyMethod などの .NET アセンブリに含まれるクラス内のパブリックメソッドです。

ml_add_dnet_connection_script を呼び出すと、メソッドが指定のイベントとスクリプトバージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ
- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ
- 「メソッド」538 ページ
- 「.NET の同期スクリプト」533 ページ

例

次の例は、ExampleClass クラスの beginDownloadConnection メソッドを begin_download イベントに対して登録します。

```
call ml_add_dnet_connection_script( 'ver1',
'begin_download',
'ExamplePackage.ExampleClass.beginDownloadConnection' );
```

ml_add_dnet_table_script システムプロシージャ

このシステムプロシージャを使用して、.NET メソッドをテーブルイベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_dnet_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。SAP HANA および Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトの内容の値は、たとえば MyClass.MyMethod などの .NET アセンブリに含まれるクラス内のパブリックメソッドです。

ml_add_dnet_table_script を呼び出すと、メソッドが指定のテーブル、イベント、スクリプトバージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ
- 「メソッド」538 ページ
- 「.NET の同期スクリプト」533 ページ

例

次の例では、EgClass クラスの empDownloadCursor メソッドを emp テーブルの download_cursor イベントに割り当てます。

```
call ml_add_dnet_table_script( 'ver1', 'emp',
'download_cursor','EgPackage.EgClass.empDownloadCursor' )
```

ml_add_java_connection_script システムプロシージャ

このシステムプロシージャを使用して、Java メソッドを接続イベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_java_connection_script (
'*version*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。SAP HANA および Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトの内容の値は、たとえば MyClass.MyMethod などの Mobile Link サーバのクラスパスに含まれるクラス内のパブリックメソッドです。

ml_add_java_connection_script を呼び出すと、メソッドが指定のイベントとスクリプトバージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ
- 「メソッド」464 ページ
- 「Java による同期スクリプトの作成」459 ページ

例

次の例は、CustEmpScripts クラスの endConnection メソッドを end_connection イベントに対して登録します。

```
call ml_add_java_connection_script( 'ver1',
'end_connection',
'CustEmpScripts.endConnection' )
```

ml_add_java_table_script システムプロシージャ

このシステムプロシージャを使用して、Java メソッドをテーブルイベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_java_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。SAP HANA および Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

script 値は、たとえば MyClass.MyMethod などの Mobile Link サーバのクラスパスに含まれるクラス内のパブリックメソッドです。

ml_add_java_table_script を呼び出すと、メソッドが指定のテーブル、イベント、スクリプトバージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

参照

- 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ
- 「メソッド」464 ページ
- 「Java による同期スクリプトの作成」459 ページ

例

次の例は、CustEmpScripts クラスの empDownloadCursor メソッドを emp テーブルの download_cursor イベントに対して登録します。

```
call ml_add_java_table_script( 'ver1', 'emp',
'download_cursor','CustEmpScripts.empDownloadCursor' )
```

ml_add_lang_connection_script システムプロシージャ

このプロシージャは内部でのみ使用されます。

参照

- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ

ml_add_lang_connection_script_chk システムプロシージャ

このプロシージャは内部でのみ使用されます。

参照

- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ

ml_add_lang_table_script システムプロシージャ

このプロシージャは内部でのみ使用されます。

参照

- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ

ml_add_lang_table_script_chk システムプロシージャ

このプロシージャは内部でのみ使用されます。

参照

- 「ml_add_table_script システムプロシージャ」664 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ

ml_add_ldap_server システムプロシージャ

LDAP サーバを作成、削除、または更新します。

構文

ml_add_ldap_server (
'*ldsrv_name*',
'*search_url*',
'*access_dn*',
'*access_dn_pwd*'
'*auth_url*'
'*conn_retries*'
'*conn_timeout*'
'*use_tls*'
)

パラメータ

構文	説明
ldsrv_name	VARCHAR(128)。ユニークな LDAP サーバ名。
search_url	VARCHAR(1024)。指定されたユーザ ID に対して DN (識別名) ルックアップを実行するために、名前、または IP アドレス、ポート番号、および検索文字列によってホストを識別する URL 文字列。
access_dn	VARCHAR(1024)。Mobile Link サーバが LDAP サーバに接続するために使用する LDAP ユーザの識別名。LDAP ユーザは、LDAP サーバで DN を検索するためのパーミッションを持つ必要があります。

構文	説明
access_dn_pwd	VARCHAR(1024)。access_dn パラメータで指定した DN と関連付けられたパスワード。
auth_url	VARCHAR(1024)。名前または IP アドレスでホストを識別、およびユーザ認証に使用する LDAP サーバのポート番号を識別する URL 文字列。
conn_retries	TINYINT。DN の検索と認証のために Mobile Link サーバが LDAP サーバへの接続を試みる回数。有効な範囲は 1 ～ 60 で、デフォルト値は 3 です。
conn_timeout	TINYINT。DN の検索と認証における Mobile Link サーバから LDAP サーバへの接続タイムアウト。この値は、秒単位です。デフォルト値は 10 秒です。
start_tls	TINYINT。DN の検索と認証のための LDAP サーバへの接続で使用される TLS を指定します。

備考

このプロシージャは、指定した LDAP サーバに関する情報を ml_ldap_server テーブルに移植します。

参照

- 「ml_add_certificates_file システムプロシージャ」646 ページ
- 「ml_add_user_auth_policy システムプロシージャ」665 ページ

例

次の例では、**my_primary** という名前の LDAP サーバを ml_ldap_server テーブルに追加します。

```
CALL ml_add_ldap_server(
'my_primary',                                  //server name
'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*',  //search URL
'cn=aseadmin, cn=Users, dc=mycompany, dc=com',       //access DN
'Secret99Password',                              //access DN password
'ldap://voyager:389/',                          //authentication URL
10,                                        //connection retries
5,                                         //connection timeout
0                                          //no TLS
)
```

ml_add_missing_dnld_scripts システムプロシージャ

このシステムプロシージャを使用して、見つからない download_cursor スクリプトと download_delete_cursor スクリプトを無視されるスクリプトとして定義します。

構文

ml_add_missing_dnld_scripts (
'*script_version_name*'**)**

パラメータ

構文	説明
script_version_name	VARCHAR(128)。スクリプトバージョンの名前。

参照

- 「download_cursor テーブルイベント」329 ページ
- 「download_delete_cursor テーブルイベント」332 ページ

ml_add_passthrough システムプロシージャ

このシステムプロシージャを使用して、スクリプトを実行するリモートデータベースを識別します。このプロシージャは、ml_passthrough システムテーブルにエントリを追加します。指定の remote_id と run_order を持つエントリがすでにテーブルに存在している場合、このプロシージャはエントリを更新します。

構文

ml_add_passthrough (
'*remote_id***',**
'*script_name***',**
run_order
)

パラメータ

構文	説明
remote_id	VARCHAR(128)。スクリプトを実行するデータベースのリモート ID。この値は、特定のクライアントに適用する ml_database テーブル内の有効なリモート ID を指定できます。また、ml_database テーブルでリストされたすべてのスクリプトクライアントに適用する場合は NULL を指定できます。 **警告** スクリプトをすべてまたは多数のリモートに適用する場合は、十分注意してください。スクリプトが適切に記述されていないと、ほとんどまたはすべてのリモートが破損したり、無効になったりする可能性があります。
script_name	VARCHAR(128)。サブスクライブされるスクリプトの名前。この値は、ml_passthrough_script テーブルで定義されている有効なスクリプト名にしてください。

構文	説明
run_order	INTEGER。run_order パラメータによって、スクリプトをリモートデータベースに適用する順序が決まります。スクリプトは常に、run_order に従って順序どおりに適用されます。各リモートデータベースには、適用が試行された最終スクリプトの run_order が保存され、その run_order 未満のスクリプトはダウンロードまたは実行しません。 この値は負でない整数にするか、NULL にします。

備考

run_order を NULL と定義した場合、プロシージャによって、remote_id の値に基づいた整数が代入されます。remote_id が NULL の場合、プロシージャによって、ml_passthrough 内の run_order の値に 10 を加えた値が代入されます。remote_id が NULL ではない場合、プロシージャによって、ml_passthrough テーブル内の remote_id のローの run_order の最大値に 10 を加えた値が代入されます。

ml_add_passthrough_repair システムプロシージャ

このシステムプロシージャを使用して、スクリプトのエラーを処理するためのルールを定義します。各ルールは、特定のスクリプトによって指定のエラーコードが生成されたときにクライアントが実行するアクションを定義します。このプロシージャは、ml_passthrough_repair システムテーブルにエントリを追加します。指定の failed_script_name と error_code を持つエントリがすでにテーブルに存在している場合、プロシージャはエントリを更新します。

構文

ml_add_passthrough_repair (
'*failed_script_name*',
error_code,
'*new_script_name*',
'*action*'
)

パラメータ

構文	説明
failed_script_name	VARCHAR(128)。このルールが適用される失敗したスクリプトの名前。この値は、ml_passthrough_script テーブル内の有効なスクリプト名にしてください。
error_code	INTEGER。このルールが処理する SQL Anywhere エラーコード。

構文	説明
new_script_name	VARCHAR(128)。action が R のときは、失敗したスクリプトを置き換えるスクリプトの名前。action が S、P、または H の場合、この値は NULL にすることができます。action が R の場合、この値は ml_passthrough_script テーブル内の有効なスクリプト名にしてください。failed_script_name と同じにすることもできます。
action	CHAR(1)。failed_script_name のために error_code が生成されたときにクライアントが実行するアクション。この値は次のいずれかにしてください。 ● **R** (置換) 失敗したスクリプトを「new_script_name」によって指定されたスクリプトに置き換える必要があること、および新しいスクリプトの実行を試みる必要があることを示します。失敗したスクリプトを再実行するには、「new_script_name」を「failed_script_name」と同じにします。 ● **P** (パージ) リモートデータベースが受信したすべてのスクリプトを破棄し、その後に通常どおりスクリプトの実行を続行する必要があることを示します。 ● **S** (省略) リモートデータベースが失敗したスクリプトを無視し、成功した場合のようにスクリプトの実行を続行する必要があることを示します。 ● **H** (停止) リモートデータベースが追加の指示を受け取るまでそれ以上のスクリプトを実行しないようにすることを示します。

備考

スクリプトを徹底的にテストすることによって、SQL パススルースクリプトが失敗することをできるかぎり回避してください。

ml_add_passthrough_script システムプロシージャ

このシステムプロシージャを使用して、パススルースクリプトを作成します。このプロシージャは、ml_passthrough_script システムテーブルにエントリを追加します。

構文

ml_add_passthrough_script (
'*script_name*',
'*flags*',
'*affected_pubs*',
'*script*',
'*description*'
)

パラメータ

構文	説明
script_name	VARCHAR(128)。スクリプト名。この値は、ユニークにする必要があります。
flags	VARCHAR(256)。クライアントにスクリプトの実行方法を指定する値。この値は、NULL にしたり、セミコロンで区切られたリストによる次のキーワードの組み合わせを含めたりすることができます。 ● **manual**　スクリプトが手動実行モードでのみ実行できることを示します。デフォルトでは、すべてのスクリプトは自動実行モードと手動実行モードのどちらでも実行できます。 ● **exclusive**　スクリプトがすべての同期対象テーブルに対する排他ロックが取得された、同期の最後に自動的に実行できることを示します。affected_pubs 値にパブリケーションが 1 つもリストされていない場合、このオプションは無視されます。このオプションは、SQL Anywhere リモートでのみ有効です。 ● **schema_diff**　スクリプトがスキーマ diff モードでのみ実行する必要があることを示します。このモードでは、スクリプトに記述されているスキーマに合わせてデータベーススキーマが変更されます。たとえば、既存のテーブルに対する create 文は alter 文として扱われます。このフラグは、Ultra Light リモートで実行されるスクリプトにのみ適用されます。 次に例を示します。 'manual;exclusive;schema_diff'
affected_pubs	TEXT。スクリプトの実行前に同期する必要があるパブリケーションのリスト。空の文字列および NULL は、同期の必要がないことを示します。この値は、SQL Anywhere クライアントでのみ有効です。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

構文	説明

script	TEXT。パススルースクリプトの内容。この値は NULL にすることはできません。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。
	
	スクリプトの内容を NULL にすることはできません。Ultra Light リモートの場合、**script** の内容は、単語 **go** で区切られた SQL 文のコレクションである必要があります。単語 **go** は、個別の行に記述してください。SQL Anywhere リモートの場合、**script** の内容は、**begin...end** ブロックで囲まれた場合、有効な任意の SQL 文のコレクションにすることができます。
	
	SQL Anywhere リモートでの **script** の内容の例
	
	DECLARE val INTEGER;
	SELECT c1 INTO val FROM t1 WHERE pk = 5;
	IF val > 100 THEN
	  INSERT INTO t2 VALUES ('c1 is big');
	ENDIF
	
	Ultra Light リモートでの **script** の内容の例
	
	CREATE TABLE myScript (c1 INT NOT NULL PRIMARY KEY)
	GO
	INSERT INTO myScript VALUES (1)
	GO

description	VARCHAR(2000)。スクリプトのコメントまたは説明。この値は NULL にすることもできます。

備考

指定された script_name がすでに ml_passthrough_script に存在する場合、このプロシージャはエラーを生成します。

ml_add_property システムプロシージャ

このシステムプロシージャを使用して、Mobile Link のプロパティを追加または削除します。このシステムプロシージャは、ml_property システムテーブルのローを変更します。

構文

ml_add_property (
'*comp_name*',
'*prop_set_name*',
'*prop_name*',
'*prop_value*'
)

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

パラメータ

構文	説明
comp_name	VARCHAR(128)。コンポーネント名。スクリプトバージョンごとにプロパティを保存するには、ScriptVersion に設定します。Mobile Link サーバプロパティの場合は、MLS に設定します。サーバ起動同期のプロパティの場合は、SIS に設定します。
prop_set_name	VARCHAR(128)。プロパティセット名。 コンポーネント名が ScriptVersion である場合、このパラメータはスクリプトバージョンの名前です。 コンポーネント名が MLS である場合は、このパラメータとして次のいずれかを使用できます。すなわち、Mobile Link ユーザに冗長性を指定する ml_user_log_verbosity か、リモート ID に冗長性を指定する ml_remote_id_log_verbosity か、ロック情報とブロック情報を Mobile Link プロファイラまたは Mobile Link サーバのログファイルにレポートする locking_and_blocking_detection のいずれかです。 コンポーネント名が SIS である場合、このパラメータはプロパティを設定している Notifier、ゲートウェイ、または Carrier の名前です。
prop_name	VARCHAR(128)。プロパティ名。 コンポーネント名が ScriptVersion である場合、このパラメータは定義するプロパティです。このプロパティは、DBConnectionContext の場合は getVersion や getProperties、ServerContext の場合は getPropertiesByVersion、getProperties、getPropertySetNames を使用して参照できます。 コンポーネント名が MLS である場合、このプロパティは、定義された Mobile Link ユーザ名またはリモート ID になるか、またはロック情報とブロック情報を Mobile Link プロファイラまたは Mobile Link サーバのログファイルにレポートする blocking_threshold_in_seconds になります。
prop_value	TEXT。プロパティの値。 prop_set_name が ml_user_log_verbosity または ml_remote_id_log_verbosity の場合は、有効な mlsrv -v オプションである必要があります。 ロック情報とブロック情報を Mobile Link プロファイラまたは Mobile Link サーバのログファイルにレポートする場合、この値は time_in_seconds です。 Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。プロパティを削除するには、NULL に設定します。

ロックとブロックのレポート

Mobile Link サーバは、実行時間が一定時間 (デフォルト値は 60 秒) を超えているユーザ定義のスクリプトを検出し、ロック情報とブロック情報を Mobile Link プロファイラにレポートし (Mobile Link サーバに Mobile Link プロファイラが接続されている場合)、さらにその情報を Mobile Link サーバのログファイルに記録します。

ロック情報およびブロック情報には、次のような情報があります。

- 同期 ID
- 現在ブロックされている Mobile Link サーバ接続 ID
- Mobile Link サーバ接続を現在ブロックしている接続 ID
- ブロックされている時間の合計 (秒単位)
- サーバ接続がブロックされているオブジェクト名または操作名

デフォルト時間を変更するには、統合データベースで次の SQL 文を実行します。

```
call ml_add_property( 'MLS', 'locking_and_blocking_detection', 'blocking_threshold_in_seconds',
'time_in_seconds' );
```

ここで、time_in_seconds は、ブロックのスレッショルド (秒単位) を指定する整数です。time_in_seconds がゼロのとき、この機能は無効です。

これは静的プロパティです。新しい値が有効になるには、Mobile Link サーバを再起動する必要があります。

対象 Mobile Link ユーザとリモート ID に対するログの冗長性

対象とする Mobile Link ユーザまたはリモート ID に対して、Mobile Link サーバが異なるログの冗長性を使用するように設定できます。Mobile Link サーバは、5 分ごとに ml_property テーブルをチェックし、Mobile Link ユーザまたはリモート ID の冗長性設定を調べます。冗長性設定が存在する場合は、指定された Mobile Link ユーザまたはリモート ID に対する出力メッセージを記録するために新しい設定が使用されます。これにより、サーバファームに悪影響を与えるような高い冗長性設定を使用したり、ファーム内の各サーバを再起動したりしなくても、特定のユーザまたはリモート ID に関する詳細を確認できます。

対象 Mobile Link ユーザ (たとえば *ml_user1*) の冗長性を最大にするには、統合データベースにログインして次の SQL 文を実行します。

```
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user1', '-v+' )
```

対象リモート ID (たとえば *rid_1*) の冗長性を最大にするには、統合データベースにログインして次の SQL 文を実行します。

```
call ml_add_property( 'MLS', 'ml_remote_id_log_versity', 'rid_1', '-v+' )
```

verbose_setting は、Mobile Link サーバで有効な -v オプションにする必要があります。たとえばローデータや未定義テーブルスクリプトをログに記録するには、verbose_setting を -vru または vru に設定できます。Mobile Link サーバは 5 分後に ml_user1 または rid_1 に対してこの冗長性設定を使用します。 「-v mlsrv16 オプション」85 ページを参照してください。

Mobile Link ユーザのログの冗長性を無効にするには、統合データベースにログインして次の SQL 文を実行します。

```
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user1', NULL )
```

Mobile Link リモート ID のログの冗長性を無効にするには、統合データベースにログインして次の SQL 文を実行します。

```
call ml_add_property( 'MLS', 'ml_remote_id_log_verbosity', 'rid_1', NULL )
```

Mobile Link サーバは 5 分後に *ml_user1* または *rid_1* に対して先の冗長性設定を使用停止します。

指定された Mobile Link ユーザやリモート ID に対して **ml_user_log_verbosity** と **ml_remote_id_log_verbosity** の「両方」が設定されている場合、および同期における Mobile Link ユーザ名やリモート ID が、指定された対象 Mobile Link ユーザやリモート ID と同一である場合、Mobile Link サーバは出力メッセージを記録するときに **ml_remote_id_log_verbosity** 設定を使用します。

サーバ起動同期

サーバ起動同期では、ml_add_property システムプロシージャを使用すると、Notifier、ゲートウェイ、Carrier のプロパティを設定できます。

たとえば、x という SMTP ゲートウェイのプロパティ server=mailserver1 を追加するには、次の SQL 文を実行します。

```
ml_add_property( 'SIS','SMTP(x)','server','mailserver1' );
```

冗長なプロパティがすべての Notifier とゲートウェイに適用されるため、特定のプロパティセット名を指定することはできません。冗長性の設定を変更するには、次のようにプロパティセット名を空のままにします。次に例を示します。

```
ml_add_property( 'SIS','','verbosity',2 );
```

スクリプトバージョン

通常の Mobile Link 同期では、このシステムプロシージャを使用して、プロパティをスクリプトバージョンに関連付けることができます。この場合、component_name を ScriptVersion に設定します。任意のプロパティを指定し、Java クラスまたは .NET クラスを使用してプロパティにアクセスできます。

たとえば、LDAP サーバを MyVersion というスクリプトバージョンに関連付けるには、次の SQL 文を実行します。

```
ml_add_property( 'ScriptVersion','MyVersion','ldap-server','MyServer' );
```

参照

- 「サーバ起動同期の Mobile Link サーバ設定」『Mobile Link サーバ起動同期』
- DBConnectionContext.getProperties メソッド [Mobile Link サーバ Java]476 ページ
- DBConnectionContext.getVersion メソッド [Mobile Link サーバ Java]478 ページ
- DBConnectionContext.GetProperties メソッド [Mobile Link サーバ .NET]564 ページ および DBConnectionContext.GetVersion メソッド [Mobile Link サーバ .NET]565 ページ
- ServerContext.getPropertiesByVersion メソッド [Mobile Link サーバ Java]506 ページ
- ServerContext.getProperties メソッド [Mobile Link サーバ Java]505 ページ
- ServerContext.getPropertySetNames メソッド [Mobile Link サーバ Java]506 ページ
- ServerContext.getPropertiesByVersion メソッド [Mobile Link サーバ .NET]601 ページ、ServerContext.getProperties メソッド [Mobile Link サーバ .NET]600 ページ、ServerContext.getPropertySetNames メソッド [Mobile Link サーバ .NET]601 ページ

ml_add_table_script システムプロシージャ

このシステムプロシージャを使用して、SQL テーブルスクリプトを統合データベースに追加するか、または統合データベースから削除します。

構文

ml_add_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。IBM DB2 LUW の場合は VARCHAR(4000)。SAP HANA および Oracle の場合は CLOB。

備考

テーブルスクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトを追加すると、スクリプトが ml_script テーブルに挿入され、このスクリプトを指定のテーブル、イベント、スクリプトバージョンに関連付ける適切な参照が定義されます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

Mobile Link サーバが -zf mlsrv16 オプションを使用して起動された場合は除き、特定のスクリプトを有効にするには、Mobile Link サーバを再起動する必要があります。-zf オプションを使用すると、各同期の初めに Mobile Link サーバによってスクリプトの変更がチェックされます。

警告
-zf オプションを使用し Mobile Link サーバを実行すると、Mobile Link サーバのパフォーマンスが低下する場合があるので、可能な限り回避することをおすすめします。

参照

- 「スクリプトを追加または削除するためのシステムプロシージャ」644 ページ
- 「スクリプトの追加と削除」256 ページ
- 「ml_add_connection_script システムプロシージャ」648 ページ
- 「ml_add_dnet_connection_script システムプロシージャ」649 ページ
- 「ml_add_dnet_table_script システムプロシージャ」650 ページ
- 「ml_add_java_connection_script システムプロシージャ」651 ページ
- 「ml_add_java_table_script システムプロシージャ」652 ページ
- 「-zf mlsrv16 オプション」103 ページ

例

次のコマンドは、Customer テーブルの upload_insert イベントに対応するテーブルスクリプトを追加します。

```
call ml_add_table_script( 'default', 'Customer', 'upload_insert',
  'INSERT INTO Customer( cust_id, name, rep_id, active )
    VALUES ( {ml r.cust_id}, {ml r.name}, {ml r.rep_id}, 1 )' )
```

ml_add_user システムプロシージャ

このプロシージャは内部でのみ使用されます。

ml_add_user_auth_policy システムプロシージャ

Mobile Link ユーザ認証ポリシーを追加するには、このシステムプロシージャを使用します。

構文

ml_add_user_auth_policy (
'*policy_name*',
'*primary_ldsrv_name*',
'*secondary_ldsrv_name*',
'*ldap_auto_failback_period*'
'*ldap_failover_to_std*'
)

パラメータ

構文	説明
policy_name	VARCHAR(128)。ユニークなユーザ認証ポリシー名。
primary_ldsrv_name	VARCHAR(128)。このユーザの認証に使用するプライマリ LDAP サーバ名を指定します。ml_ldap_server テーブル内にすでに存在する LDAP サーバ名を指定する必要があります。
secondary_ldsrv_name	VARCHAR(128)。フェイルオーバのためにセカンダリ LDAP サーバ名を指定します。セカンダリ LDAP サーバ名は、ml_ldap_server テーブル内にすでに存在する必要があります。
ldap_auto_failback_period	INTEGER。このパラメータを使用して、ユーザ認証用のプライマリ LDAP サーバにフェイルオーバするタイミングを Mobile Link サーバに通知します。時間は秒単位で指定し、デフォルト値は 900 秒 (15 分) です。 プライマリ LDAP サーバをユーザ認証に使用できない場合、Mobile Link サーバは、問題が検出されたタイミングを記憶し、ユーザ認証用のセカンダリサーバに切り換えます。障害が検出されてから経過した時間が @ldap_auto_failback_period に達すると、Mobile Link サーバは、このユーザ認証ポリシーを現在使用しているのがどんなユーザであれ、ユーザ認証用のプライマリサーバの使用に再び切り換えます。
ldap_failover_to_std	INTEGER。プライマリ LDAP サーバもセカンダリ LDAP サーバも使用できない場合、Mobile Link サーバで標準的な方法 (パスワードスクリプトとユーザ認証スクリプト) を使用してユーザを認証するかどうかを指定します。値は 1 (TRUE) または 0 (FALSE) のいずれかです。デフォルトは 1 です。

備考

このプロシージャは、指定された policy_name がテーブル内に存在しない場合に、ユーザ認証ポリシーを ml_user_auth_policy テーブルに追加します。policy_name がテーブルにすでに存在している場合は、非 NULL パラメータを指定してこのプロシージャを実行すると、対応するすべてのフィールドが、指定した非 NULL パラメータで更新されます。たとえば、プライマリ LDAP サーバもセカンダリ LDAP サーバも使用できない場合、次の SQL 文は、*ldap_server2* をセカンダリ LDAP サーバとして使用するように、ユーザ認証ポリシー *policy_1* を更新してから、フェイルオーバを有効にしてパスワードスクリプトとユーザ認証スクリプトをベースにした標準的な認証を使用します。

```
call ml_add_user_auth_policy( 'policy_1', NULL, 'ldap_server2', NULL, 1 )
```

認証ポリシーを削除するには、policy_name を除くすべてのパラメータを NULL にします。

Mobile Link ユーザ認証ポリシーを追加するとき、パラメータ primary_ldsrv_name を NULL にすることはできませんが、パラメータ secondary_ldsrv_name を NULL にすることは可能です。

参照

- 「ml_add_ldap_server システムプロシージャ」654 ページ
- 「ml_add_certificates_file システムプロシージャ」646 ページ

ml_delete_passthrough システムプロシージャ

このストアドプロシージャは、指定の実行順序を持つ指定のスクリプトを指定のリモートデータベースにダウンロードさせる、ml_passthrough テーブル内のローを削除します。スクリプトは、削除される前にリモートデータベースにダウンロードされた場合、リモートデータベースから削除されずに通常どおり実行されます。

構文

ml_delete_passthrough (
'*remote_id*',
'*script_name*',
'*run_order*'
)

パラメータ

構文	説明
remote_id	VARCHAR(128)。リモート ID。**remote_id** が NULL の場合、指定されたスクリプト名と実行順序に対する、ml_passthrough テーブル内のすべてのローが削除されます。
script_name	VARCHAR(128)。スクリプト名。
run_order	INTEGER。リモートデータベースで適用されるスクリプトの実行順序。**run_order** が NULL の場合、指定された **remote_id** と **script_name** について、実行順序に関係なくすべてのローが ml_passthrough テーブルから削除されます。

備考

Mobile Link サーバは、エントリを ml_passthrough テーブルから自動的には削除しません。古いパススルースクリプトを削除するには、このプロシージャを使用してください。

ml_delete_passthrough_repair システムプロシージャ

このシステムプロシージャを使用して、修復ルールを ml_passthrough_repair システムテーブルから削除します。

構文

ml_delete_passthrough_repair (
'*failed_script_name*',
error_code
)

パラメータ

構文	説明
failed_script_name	VARCHAR(128)。このルールが適用されるスクリプトの名前。
error_code	INTEGER。ルールが適用されるエラーコード。

備考

Mobile Link サーバは、エントリを ml_passthrough_repair テーブルから自動的には削除しません。古いパススルー修復スクリプトを削除するには、このプロシージャを使用してください。

ml_delete_passthrough_script システムプロシージャ

このシステムプロシージャを使用して、パススルースクリプトを ml_passthrough_script システムテーブルから削除します。

構文

ml_delete_passthrough_script (
'*script_name*'
)

パラメータ

構文	説明
script_name	VARCHAR(128)。削除するスクリプトの名前。

備考

スクリプトは、ml_passthrough システムテーブルまたは ml_passthrough_repair システムテーブルで参照されている場合、削除できません。

Mobile Link サーバは、エントリを ml_passthrough_script テーブルから自動的には削除しません。古いパススルースクリプトを削除するには、このプロシージャを使用してください。

ml_delete_sync_state システムプロシージャ

このプロシージャを使用して、未使用または不要な同期ステータスを削除します。

構文

ml_delete_sync_state (
'*user*',
'*remote_id*'
)

 Copyright © 2013, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0

パラメータ

構文	説明
user	VARCHAR(128)。Mobile Link ユーザ名。
remote_id	VARCHAR(128)。リモート ID。

備考

これらのパラメータには NULL を指定できます。すべてのパラメータが NULL の場合、プロシージャは何も処理を実行しません。

このストアドプロシージャは、指定された Mobile Link ユーザ名とリモート ID について、ml_subscription テーブルからすべてのローを削除します。指定されたリモート ID が ml_subscription テーブルのどのローからも参照されなくなった場合は、ml_database テーブルからそのリモート ID も削除します。

リモート ID が NULL で、Mobile Link ユーザ名が NULL でない場合、指定された Mobile Link ユーザ名で参照されるすべてのローを ml_subscription テーブルから削除します。また、ml_subscription テーブル内のどのローからも参照されなくなったすべてのリモート ID を ml_database テーブルから削除します。

Mobile Link ユーザ名が NULL でリモート ID が NULL でない場合、すべてのリモート ID が ml_database テーブルから削除され、この Mobile Link ユーザが ml_subscription テーブル内のどのローからも参照されなくなっても、このユーザはこのストアドプロシージャによって削除されません。この Mobile Link ユーザを削除する必要がある場合は、次のようなコマンドを発行して削除できます。

```
delete from ml_user where name = 'user_name'
```

ここで、*user_name* は、削除する Mobile Link ユーザです。

このストアドプロシージャは、細心の注意を払って使用してください。次回 Mobile Link クライアントがこのリモート ID の同期を要求したとき、Mobile Link サーバは、同期ステータスをチェックしないで、このリモート ID を ml_database テーブルと ml_subscription テーブルに自動的に追加します。前回行われた同期が成功しなかったリモート ID の同期ステータスを削除すると、データの不整合が発生する場合があります。

このストアドプロシージャは、指定されたリモート ID について、ml_subscription テーブルと ml_database テーブルからすべてのローを削除します。

例

次の例は、John という Mobile Link ユーザのリモート ID remote_db_for_John を持つリモートデータベースに関する Mobile Link システムテーブル情報をクリーンアップします。

```
CALL ml_delete_sync_state( 'John', 'remote_db_for_John' )
```

ml_delete_sync_state_before システムプロシージャ

このプロシージャを使用して、リモートデータベースを削除したときに Mobile Link システムテーブルをクリーンアップします。

構文

ml_delete_sync_state_before (
'*ts*'
)

パラメータ

構文	説明
ts	TIMESTAMP。日時は、統合データベースで指定された順序で指定してください。統合データベースの日時の表示形式が "yyyy/mm/dd hh:mm:ss.ssss" に設定されている場合、タイムスタンプは年、月、日、時、分、秒、秒以下の順に指定します。

備考

このストアドプロシージャは Mobile Link システムテーブルから、もう使用されていないリモートデータベースに関連するローを削除します。特に次の処理が行われます。

- ml_subscription システムテーブルから、last_upload_time と last_download_time の両方が指定のタイムスタンプより前の値になっているすべてのローを削除する。
- リモート ID が ml_subscription テーブルのどのローからも参照されていない場合は、ml_database システムテーブルからリモート ID を削除する。

実際には削除されていないリモートデータベースのローを削除する可能性がある場合は、このシステムプロシージャを使用しないでください。もし使用すると、ml_subscription と ml_database テーブル内のローが削除されることで、アップロードが失敗して「不明な状態」になっているリモートデータベースに問題が発生する可能性があります。その不明な状態では、リモートデータベースは Mobile Link システムテーブルに依存してデータを再送します。

プロシージャはパラメータの日付／時間形式を検証しないので、このプロシージャに指定するタイムスタンプには正しい日付／時間形式を使用してください。

例

次の例は、2004 年 1 月 10 日以降同期されていないリモートデータベースに関する Mobile Link システムテーブル情報をクリーンアップします。この例は、日付／時間形式が yyyy/mm/dd hh:mm:ss.ssss である SQL Anywhere 統合データベースで使用できます。

```
CALL ml_delete_sync_state_before( '2004/01/10 00:00:00' )
```

ml_delete_user システムプロシージャ

このプロシージャは内部でのみ使用されます。

ml_model_drop システムプロシージャ

このプロシージャを使用して、Sybase Central の Mobile Link 16 プラグインでインストールされた同期モデルを削除します。

構文

ml_model_drop (
'*script_version*'
)

パラメータ

構文	説明
script_version	VARCHAR(128)。削除する同期モデルと関連付けられたスクリプトバージョンの名前。

備考

このストアドプロシージャは、名前付きスクリプトバージョンに含まれる同期スクリプトを削除するほか、同期モデルが展開されたときに作成されたスキーマ (シャドウテーブル、トラッキングカラム、トリガ、インデックスなど) も削除します。

別の *script_version* と共有されているスキーマは削除されません。

Mobile Link 16 プラグインを使用せずに手動で *script_version* がインストールされている場合、スキーマは削除されません。

参照

- 「ml_model_check_all_schema システムプロシージャ」 671 ページ
- 「ml_model_check_version_schema システムプロシージャ」 673 ページ

ml_model_check_all_schema システムプロシージャ

このプロシージャを使用して、配備された同期モデルに必要な各スキーマオブジェクトのステータスをチェックします。このストアドプロシージャは、配備されたすべての同期モデルの情報を返します。

構文

ml_model_check_all_schema

備考

このプロシージャは、展開されたすべての同期モデルで必要とされる各スキーマオブジェクトが入った結果セットを返します。

Sybase Central 外部にインストールされたスクリプトバージョンまたはバージョン 16 以前に展開された同期モデルに対しては、結果が返されません。

結果セットには次のカラムがあります。

- **schema_owner**　スキーマの所有者を識別します。
- **table_name**　テーブル名を識別します。
- **schema_type**　スキーマタイプを識別します。タイプは次のいずれかです。
  - TABLE
  - INDEX
  - COLUMN
  - TRIGGER
  - PROCEDURE
- **object_name**　オブジェクト名を識別します。
- **locked**　このカラムが 1 に設定されている場合、スキーマがプラグインによって変更されたり削除されたりすることはありません。展開前にすでに存在していた同期モデルが使用しているスキーマは、ロック済みとしてマークが付けられます。
- **used_by**　スキーマオブジェクトを必要とするスクリプトバージョン。
- **status**　ステータスは次のいずれかです。

  INSTALLED　スキーマは正常にインストールされています。

  MISSING　対象のスキーマはインストールされていません。

  MISMATCH　インストールされたスキーマは、必要なスキーマとは異なります。

  UNVERIFIED　スキーマは存在しますが、正しく定義されているかどうかを判断するための情報が十分ではありません。

  UNUSED　このスキーマオブジェクトを使用している同期モデルはありません。
- **overwrite_action**　次のいずれかです。

  REPLACE　モデルを再展開する場合、既存のスキーマが削除されて再作成されます。

  CREATE　モデルを再展開する場合、スキーマが作成されます。

  SKIP　スキーマはすでに正しくインストールされているか、正しいインストールをブロックしている競合スキーマが存在するかのいずれかです。
- **preserve_action**　今後の使用のために予約されています。

参照

- 「ml_model_drop システムプロシージャ」 671 ページ
- 「ml_model_check_version_schema システムプロシージャ」 673 ページ

ml_model_check_version_schema システムプロシージャ

このプロシージャを使用して、配備された同期モデルに必要な各スキーマオブジェクトのステータスをチェックします。このストアドプロシージャは、指定したスクリプトバージョンの情報を返します。

構文

ml_model_check_version_schema (
'*script_version*'
)

パラメータ

構文	説明
script_version	VARCHAR(128)。チェックする同期モデルと関連付けられたスクリプトバージョンの名前。

備考

このプロシージャは、指定されたスクリプトバージョンで必要とされる各スキーマオブジェクトのステータスが入った結果セットを返します。

Sybase Central 外部にインストールされたスクリプトバージョンまたはバージョン 16 以前に展開された同期モデルに対しては、結果が返されません。

結果セットには次のカラムがあります。

- **schema_owner**　スキーマの所有者を識別します。
- **table_name**　テーブル名を識別します。
- **schema_type**　スキーマタイプを識別します。タイプは次のいずれかです。
  - TABLE
  - INDEX
  - COLUMN
  - TRIGGER
  - PROCEDURE
- **object_name**　オブジェクト名を識別します。
- **locked**　このカラムが 1 に設定されている場合、スキーマがプラグインによって変更されたり削除されたりすることはありません。展開前にすでに存在していた同期モデルが使用しているスキーマは、ロック済みとしてマークが付けられます。
- **status**　ステータスは次のいずれかです。

  INSTALLED　スキーマは正常にインストールされています。

  MISSING　対象のスキーマはインストールされていません。

MISMATCH　インストールされたスキーマは、必要なスキーマとは異なります。

UNVERIFIED　スキーマは存在しますが、正しく定義されているかどうかを判断するための情報が十分ではありません。

UNUSED　このスキーマオブジェクトを使用している同期モデルはありません。

- **overwrite_action**　次のいずれかです。

  REPLACE　モデルを再展開する場合、既存のスキーマが削除されて再作成されます。

  CREATE　モデルを再展開する場合、スキーマが作成されます。

  SKIP　スキーマはすでに正しくインストールされているか、正しいインストールをブロックしている競合スキーマが存在するかのいずれかです。

- **preserve_action**　今後の使用のために予約されています。

参照

- 「ml_model_drop システムプロシージャ」671 ページ
- 「ml_model_check_all_schema システムプロシージャ」671 ページ

ml_ra_add_agent_id システムプロシージャ

このプロシージャを使用して、統合データベースの新しいリモートエージェントを定義します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。統合データベースで定義される新しいエージェントの ID を指定する IN パラメータ。

備考

エージェントは、ml_ra_add_agent_id を先に呼び出さないで Mobile Link サーバに接続した場合、自動的に統合データベースに追加されます。ただし、そのエージェントのすべてのプロパティはデフォルト値になります。

参照

- 「ml_ra_manage_remote_db システムプロシージャ」688 ページ
- 「ml_ra_clone_agent_properties システムプロシージャ」676 ページ
- 「ml_ra_set_agent_property システムプロシージャ」690 ページ

ml_ra_assign_task システムプロシージャ

このプロシージャを使用して、タスクを特定のリモートエージェントに割り当てます。

パラメータ

構文	説明
agent_id	VARCHAR(128)。タスクの割り当て先エージェントの ID を指定する IN パラメータ。
task_name	VARCHAR(128)。割り当てる特定のタスクの名前を指定する IN パラメータ。

備考

まず Sybase Central で Mobile Link 16 プラグインを使用してタスクを定義してから、このシステムプロシージャを呼び出します。

一部のタスクは、特定のリモートデータベースを対象とします。この場合、エージェントはそのタイプのリモートデータベースを管理している必要があります。

タスクがすでにエージェントに割り当てられ、続いて完了した場合は、そのタスクを再び割り当てることができます。これにより、タスクは再びアクティブになり、スケジュールに従って実行されます。

まず Sybase Central を使用してタスクを定義してから、task_name パラメータを呼び出します。

参照

- 「ml_ra_cancel_task_instance システムプロシージャ」676 ページ
- 「ml_ra_notify_task システムプロシージャ」689 ページ
- 「ml_ra_cancel_notification システムプロシージャ」675 ページ

ml_ra_cancel_notification システムプロシージャ

このプロシージャを使用して、不要になったサーバ起動リモートタスク (SIRT) 要求をキャンセルします。

パラメータ

構文	説明
agent_id	VARCHAR(128)。キャンセルするタスクを担当するエージェントの ID を指定する IN パラメータ。
task_name	VARCHAR(128)。キャンセルする特定のタスクの名前を指定する IN パラメータ。

参照

- 「ml_ra_notify_task システムプロシージャ」689 ページ
- 「ml_ra_assign_task システムプロシージャ」674 ページ
- 「ml_ra_delete_task システムプロシージャ」678 ページ
- 「ml_ra_cancel_task_instance システムプロシージャ」676 ページ

ml_ra_cancel_task_instance システムプロシージャ

このプロシージャを使用して、リモートタスクインスタンスをキャンセルします。

パラメータ

構文	説明
agent_id	VARCHAR(128)。キャンセルするタスクを担当するエージェントの ID を指定する IN パラメータ。
task_name	VARCHAR(128)。キャンセルする特定のタスクの名前を指定する IN パラメータ。

備考

このシステムプロシージャを使用して、不要になったタスクをキャンセルします。

エージェントがタスクのアクティブな実行を完了し、エージェントデータベースの同期によってキャンセル済み状態を確認するまで、キャンセルされたタスクは **[キャンセル保留]** 状態としてレポートされます。

参照

- 「ml_ra_cancel_notification システムプロシージャ」675 ページ
- 「ml_ra_notify_task システムプロシージャ」689 ページ
- 「ml_ra_assign_task システムプロシージャ」674 ページ
- 「ml_ra_delete_task システムプロシージャ」678 ページ

ml_ra_clone_agent_properties システムプロシージャ

このプロシージャを使用して、すべてのリモートエージェントプロパティを一度に設定します。

パラメータ

構文	説明
dst_agent_id	VARCHAR(128)。作成するコピー先エージェントの ID を指定する IN パラメータ。
src_agent_id	VARCHAR(128)。新しいエージェントを作成するためにクローンを作成するエージェントの ID を指定する IN パラメータ。

備考

このシステムプロシージャを使用して、指定したエージェントのすべてのプロパティを一度に設定します。既存のエージェントのプロパティがすべて新しいエージェントにコピーされます。Sybase Central を使用すると、個々のエージェントプロパティを簡単に設定できます。

割り当てられているタスクと管理されているリモートは新しいエージェントにコピーされません。

参照

- 「ml_ra_set_agent_property システムプロシージャ」690 ページ
- 「ml_ra_add_agent_id システムプロシージャ」674 ページ
- 「ml_ra_manage_remote_db システムプロシージャ」688 ページ

ml_ra_delete_agent_id システムプロシージャ

このプロシージャを使用して、定義済みのエージェントを統合データベースから削除します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。削除するエージェントの ID を指定する IN パラメータ。

備考

リモートデータベースを管理しているエージェントを削除した場合、それらのリモートデータベースは管理されなくなります。

参照

- 「ml_ra_delete_events_before システムプロシージャ」677 ページ
- 「ml_ra_delete_remote_id システムプロシージャ」678 ページ
- 「ml_ra_unmanage_remote_db システムプロシージャ」690 ページ

ml_ra_delete_events_before システムプロシージャ

このプロシージャを使用して、不要になったイベントを統合データベースから削除します。

パラメータ

構文	説明
delete_rows_older_than	TIMESTAMP。指定した値より古いイベントを統合データベースから削除するように指定する IN パラメータ。

備考

リモートタスクが頻繁にステータスを返す場合、大量のイベントが統合データベースに蓄積する可能性があります。

参照

- 「ml_ra_get_latest_event_id システムプロシージャ」683 ページ

ml_ra_delete_remote_id システムプロシージャ

このプロシージャを使用して、不要になったリモートデータベースを統合データベースから削除します。

パラメータ

構文	説明
remote_id	VARCHAR(128)。削除するリモートデータベースに対応するリモート ID を指定する IN パラメータ。

備考

このプロシージャは、指定した remote_id に対してアクティブなタスクがある場合は失敗します。削除を強制的に実行するには、まずリモートを管理している agent_id を削除します。

参照

- 「ml_ra_delete_agent_id システムプロシージャ」677 ページ
- 「ml_ra_delete_task システムプロシージャ」678 ページ

ml_ra_delete_task システムプロシージャ

このプロシージャを使用して、リモートタスクを統合データベースから削除します。

パラメータ

構文	説明
task_name	VARCHAR(128)。削除するリモートタスクの名前を指定する IN パラメータ。

備考

このシステムプロシージャは、アクティブなタスクインスタンスがある場合は失敗します。

参照

- 「ml_ra_delete_agent_id システムプロシージャ」677 ページ
- 「ml_ra_delete_events_before システムプロシージャ」677 ページ
- 「ml_ra_delete_remote_id システムプロシージャ」678 ページ

ml_ra_get_agent_events システムプロシージャ

このプロシージャを使用して、イベントを問い合わせます。

パラメータ

構文	説明
start_at_event_id	BIGINT。クエリの開始点となるイベントの ID を指定する IN パラメータ。
max_events_to_fetch	BIGINT。フェッチするイベントの最大数を指定する IN パラメータ。

戻り値

結果	説明
event_id	BIGINT。各イベントに割り当てられたユニークな ID。値は、新しいイベントごとに 1 ずつ増えます。
event_class	VARCHAR(1)。イベントクラス。情報の場合は **I**、エラーの場合は **E** になります。

結果	説明
event_type	VARCHAR(8)。イベントタイプを以下に示します。 ● **ANEW**　統合データベースに新しいエージェントが定義されました。このイベントは、ml_ra_add_agent を呼び出した場合、またはエージェントが初めて統合データベースに接続するときにエージェントが事前に設定されていない場合に発生する可能性があります。 ● **AFIRST**　エージェントの最初の同期時に発生します。 ● **ADUP**　重複した agent_id が見つかりました。つまり、2 つ以上のエージェントが同じ ID を使用しようとしています。この場合の result_text は、ブロックされたエージェントのエージェントデータベースの remote_id です。 ● **ARESET**　エージェントによってエージェントデータベースが再構築されました。一部のタスクの進行状況と結果が失われた可能性があります。 ● **TB**　タスクが実行を開始しました。 ● **TE**　タスクの実行が終了しました。致命的なエラーは発生しませんでした。 ● **TW**　タスクの実行が処理を続行するまでリトライ間隔を待機しています。 ● **TAC**　コマンドがアボートしたため、タスクの実行が終了しました。 ● **TAT**　実行の最大許容時間を超えたため、タスクの実行が終了しました。 ● **TAR**　最大リトライ回数を超えたため、タスクの実行が終了しました。 ● **TFS**　タスクが完了しました。このタスクは 1 回のみ実行されるものであり、実行に成功したため、再実行されません。 ● **TFF**　タスクが完了しました。このタスクは 1 回のみ実行されるものであり、実行に失敗したため、再実行されません。 ● **TFE**　タスクが完了しました。タスクのスケジュールの期限が切れたため、再実行されません。 ● **TFC**　タスクが完了しました。タスクはサーバによってキャンセルされたため、再実行されません。 ● **CR - コマンド結果**　result_code と result_text に、コマンドのタイプに固有の値が取り込まれます。

結果	説明
	● **CE - コマンドエラー** result_code と result_text に、コマンドのタイプに固有の値が取り込まれます。
agent_id	VARCHAR(128)。このイベントを生成したエージェントの ID。
remote_id	VARCHAR(128)。イベントが適用されるリモートデータベースの ID。これは、特定のリモートデータベースを対象としたタスク関連のイベントに対してのみ設定されます。
task_name	VARCHAR(128)。タスクの名前。これは、タスク関連のイベントに対してのみ設定されます。
command_number	INTEGER。このイベントが適用されるタスク内でのコマンド番号。これは、コマンド固有のイベントに対してのみ設定されます。
run_number	BIGINT。タスクの実行ごとに割り当てられたユニークな番号。これは、タスク固有のイベントに対してのみ設定されます。
duration	INTEGER。イベントの所要時間。これは、コマンド固有のイベントに対してのみ設定されます。
event_time	TIMESTAMP。イベントが発生した日時。ほとんどのイベントでは、日時はエージェントが実行されているコンピュータのクロックに基づきます。
event_received	TIMESTAMP。サーバがイベントを受信した日時。これは、常に統合データベースのクロックに基づいて設定されます。
result_code	BIGINT。イベント固有の BIGINT。たとえば、SQL クエリコマンドの結果では、コードは SQLCODE になります。
result_text	LONG VARCHAR。イベント固有の LONG VARCHAR。たとえば、SQL クエリコマンドの結果では、このカラムに *CSV* フォーマットの結果セットが格納されます。
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

備考

代わりに、ml_ra_get_task_results プロシージャを使用して、タスクの特定の実行に関連するイベントのみをフェッチすることもできます。NULL の @run_number を渡すと、タスクの最後の実行を取得できます。

このプロシージャを使用する 1 つの方法として、ml_ra_get_agent_events を使用してタスク終了イベント (**TE**) を待機し、その後 ml_ra_get_task_results を呼び出して処理が必要になる可能性がある各コマンド結果を取得する方法があります。

参照

- 「ml_ra_get_task_results システムプロシージャ」685 ページ
- 「ml_ra_get_task_results システムプロシージャ」685 ページ

ml_ra_get_agent_ids システムプロシージャ

このプロシージャを使用して、統合データベースのすべてのエージェントを取得します。

戻り値

結果	説明
agent_id	VARCHAR(128)。エージェント ID。
remote_id	VARCHAR(128)。エージェントデータベースのリモート ID。
last_download_time	TIMESTAMP。前回のダウンロードの時刻。
last_upload_time	TIMESTAMP。前回のアップロードの日時。
active_task_count	INTEGER。アクティブタスクの数。
description	VARCHAR(2048)。今後の使用のために予約されています。
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

参照

- 「ml_ra_get_remote_ids システムプロシージャ」684 ページ

ml_ra_get_agent_properties システムプロシージャ

このプロシージャを使用して、エージェントに設定されたすべてのプロパティを表示します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。プロパティを設定しているエージェントの ID を指定する IN パラメータ。

戻り値

結果	説明
property_name	VARCHAR(128)。プロパティ名。

結果	説明
property_value	VARCHAR(2048)。プロパティの値。
last_modified	TIMESTAMP。プロパティが最後に変更された時刻。
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

参照

- 「ml_ra_get_agent_ids システムプロシージャ」682 ページ
- 「ml_ra_clone_agent_properties システムプロシージャ」676 ページ

ml_ra_get_latest_event_id システムプロシージャ

このプロシージャを使用して、新しいイベントの数を確認します。

パラメータ

構文	説明
event_id	BIGINT。最新のイベントの ID を指定する OUT パラメータ。

備考

新しいイベントの数を確認するには、ml_ra_get_latest_event_id システムプロシージャを呼び出して最後に処理した event_id を減算します。

参照

- 「ml_ra_get_agent_events システムプロシージャ」679 ページ

ml_ra_get_orphan_taskdbs システムプロシージャ

このプロシージャを使用して、孤立したエージェントデータベース、つまり有効なエージェント ID がないエージェントデータベースのリストを表示します。

戻り値

結果	説明
remote_id	VARCHAR(128)。リモート ID。
orig_agent_id	VARCHAR(128)。エージェントデータベースが関連付けられた元のエージェントの ID。
last_sync	TIMESTAMP。最後の同期の時刻。

結果	説明
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

備考

同期システムに数多くの問題が発生した場合、データベースが孤立することがあります。たとえば、異なるコンピュータで重複したエージェント ID を作成した場合や、2 つのエージェントデータベースが同じエージェント ID を使用しようとした場合などです。

remote_id フィールドには、問題の診断に役立つコンピュータ名が格納されています。

参照

- 「ml_ra_reassign_taskdb システムプロシージャ」689 ページ

ml_ra_get_remote_ids システムプロシージャ

このプロシージャを使用して、統合データベースのすべてのリモートデータベース (エージェントデータベースを除く) を取得します。

パラメータ

なし

戻り値

結果	説明
remote_id	VARCHAR(128)。リモートデータベースのリモート ID。
schema_name	VARCHAR(128)。リモートデータベースのタイプ。
agent_id	VARCHAR(128)。リモートデータベースのエージェント ID。
agent_conn_str	VARCHAR(2048)。エージェントの接続文字列。
last_download_time	TIMESTAMP。前回のダウンロードの時刻。
last_upload_time	TIMESTAMP。前回のアップロードの日時。
description	VARCHAR(128)。データベースの説明。
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

参照

- 「ml_ra_get_agent_properties システムプロシージャ」682 ページ

ml_ra_get_task_results システムプロシージャ

このプロシージャを使用して、タスクの特定の実行に関連するイベントを取得します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。結果を取得するエージェントの ID を指定する IN パラメータ。
task_name	VARCHAR(128)。結果を取得するタスクの名前を指定する IN パラメータ。
run_number	INTEGER。結果を取得する実行番号を指定する IN パラメータ。

戻り値

結果	説明
event_id	BIGINT。各イベントに割り当てられたユニークな ID。値は、新しいイベントごとに 1 ずつ増えます。
event_class	VARCHAR(1)。イベントクラス。情報の場合は **I**、エラーの場合は **E** になります。
event_type	VARCHAR(8)。イベントタイプ。
agent_id	VARCHAR(128)。このイベントを生成したエージェントの ID。
remote_id	VARCHAR(128)。イベントが適用されるリモートデータベースの ID。これは、特定のリモートデータベースを対象としたタスク関連のイベントに対してのみ設定されます。
task_name	VARCHAR(128)。タスクの名前。これは、タスク関連のイベントに対してのみ設定されます。
command_number	INTEGER。このイベントが適用されるタスク内でのコマンド番号。これは、コマンド固有のイベントに対してのみ設定されます。
run_number	BIGINT。タスクの実行ごとに割り当てられたユニークな番号。これは、タスク固有のイベントに対してのみ設定されます。
duration	INTEGER。イベントの所要時間。これは、コマンド固有のイベントに対してのみ設定されます。
event_time	TIMESTAMP。イベントが発生した日時。ほとんどのイベントでは、日時はエージェントが実行されているコンピュータのクロックに基づきます。

結果	説明
event_received	TIMESTAMP。サーバがイベントを受信した日時。これは、常に統合データベースのクロックに基づいて設定されます。
result_code	BIGINT。イベント固有の BIGINT。たとえば、SQL クエリコマンドの結果では、コードは SQLCODE になります。
result_text	LONG VARCHAR。イベント固有の LONG VARCHAR。たとえば、SQL クエリコマンドの結果では、このカラムに *CSV* フォーマットの結果セットが格納されます。
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

備考

このシステムプロシージャは、タスクの特定の実行に関連するイベントのみをフェッチします。これは、ml_ra_get_agent_events システムプロシージャに代わるものです。

NULL の @run_number を渡すと、タスクの最後の実行を取得できます。

参照

- 「ml_ra_get_agent_events システムプロシージャ」 679 ページ

例

このプロシージャを使用する 1 つの方法として、ml_ra_get_agent_events を使用してタスク終了イベントを待機し、その後 ml_ra_get_task_results を呼び出して処理が必要になる可能性がある各コマンド結果を取得する方法があります。

ml_ra_get_task_status システムプロシージャ

このプロシージャを使用して、タスクのステータスをチェックします。

パラメータ

構文	説明
agent_id	VARCHAR(128)。ステータスを取得するエージェントの ID を指定する IN パラメータ。
task_name	VARCHAR(128)。ステータスを取得するタスクの名前を指定する IN パラメータ。

戻り値

結果	説明
agent_id	VARCHAR(128)。このイベントを生成したエージェントの ID。
remote_id	VARCHAR(128)。イベントが適用されるリモートデータベースの ID。
task_name	VARCHAR(128)。タスクの名前。
task_id	BIGINT。タスク ID。
state	VARCHAR(4)。展開されたタスクのステータス。ステータスは次のいずれかです。 ● **P** 保留。エージェントがタスクを受信したという確認を待機しています。 ● **A** アクティブ。エージェントにはタスクがあり、エージェントはスケジュールに従ってタスクを実行します。 ● **S** 成功。タスクは完了し、再割り当てされるまでは再実行されません。 ● **F** 失敗。タスクは完了し、再割り当てされるまでは再実行されません。 ● **CP** キャンセル保留。エージェントがタスクをキャンセルしたという確認を待機しています。 ● **C** キャンセル済み。タスクは完了し、再割り当てされるまでは再実行されません。 ● **E** 失効。タスクは完了し、再割り当てされるまでは再実行されません。
reported_exec_count	BIGINT。レポートされた実行済みタスクの数。
reported_error_count	BIGINT。レポートされたエラーの数。
reported_attempt_count	BIGINT。レポートされたタスクの実行試行回数。
last_status_update	TIMESTAMP。ステータスが最後に更新された時刻。
last_success	TIMESTAMP。最後に成功したタスクの時刻。
assignment_time	TIMESTAMP。タスクが割り当てられた時刻。

結果	説明
p_crsr	SYS_REF_CURSOR.これは Oracle 専用の OUT パラメータです。

備考

@agent_id パラメータと @task_name パラメータを NULL に設定すると、すべての agent_id、すべての task_name、またはこれらの両方のステータスを取得できます。

reported_attempt_count が reported_exec_count より大きくなる場合があります。これは、試行時にタスクの前提条件が false に評価され、タスクが実行されなかったことを示します。

成功数は、reported_exec_count から reported_error_count を減算することによって計算できます。

参照

- 「ml_ra_get_task_results システムプロシージャ」685 ページ

ml_ra_manage_remote_db システムプロシージャ

このプロシージャを使用して、エージェントが管理するリモートデータベースを追加します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。統合データベースで定義される新しいエージェントの ID を指定する IN パラメータ。
schema_name	VARCHAR(128)。作成するリモートデータベースのタイプを指定する IN パラメータ。このスキーマ名は、Sybase Central を使用して統合データベースに事前に定義されている必要があります。
conn_str	VARCHAR(128)。リモートデータベースに接続するためにエージェントによって使用されるデータベース接続文字列を指定する IN パラメータ。

参照

- 「ml_ra_add_agent_id システムプロシージャ」674 ページ
- 「ml_ra_clone_agent_properties システムプロシージャ」676 ページ

ml_ra_notify_agent_sync システムプロシージャ

このプロシージャを使用して、エージェントにステータスを同期させます。

パラメータ

構文	説明
agent_id	VARCHAR(128)。同期するエージェントの ID を指定する IN パラメータ。

備考

このシステムプロシージャは、新しいタスクを指定されたエージェントに送信し、エージェントにタスクの実行結果を Mobile Link サーバに送信させます。

参照

- 「ml_ra_get_task_results システムプロシージャ」685 ページ

ml_ra_notify_task システムプロシージャ

このプロシージャを使用して、タスクをサーバ起動リモートタスク (SIRT) によって実行します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。タスクを実行するエージェントの ID を指定する IN パラメータ。
task_name	VARCHAR(128)。エージェントで実行するタスクの名前を指定する IN パラメータ。

参照

- 「ml_ra_cancel_notification システムプロシージャ」675 ページ
- 「ml_ra_delete_task システムプロシージャ」678 ページ

ml_ra_reassign_taskdb システムプロシージャ

このプロシージャを使用して、孤立したエージェントデータベースがある状況でエージェントデータベースを再割り当てします。

パラメータ

構文	説明
taskdb_remote_id	VARCHAR(128)。孤立したエージェントデータベースのリモート ID を指定する IN パラメータ。
new_agent_id	VARCHAR(128)。孤立したエージェントデータベースの割り当て先となる新しいエージェントの ID を指定する IN パラメータ。

備考

2 つのエージェントデータベースが存在し、両方が同じ agent_id を使用する必要がある場合、システムでは最初のエージェントデータベースが有効と見なされます。2 つ目のエージェントデータベースは、孤立している、つまり有効な agent_id が関連付けられていないと見なされます。

参照

- 「ml_ra_get_orphan_taskdbs システムプロシージャ」683 ページ

ml_ra_set_agent_property システムプロシージャ

このプロシージャを使用して、リモートエージェントプロパティを設定します。

パラメータ

構文	説明
agent_id	VARCHAR(128)。エージェント ID を指定する IN パラメータ。
property_name	VARCHAR(128)。設定するプロパティ名を指定する IN パラメータ。
property_value	VARCHAR(2048)。設定するプロパティ値を指定する IN パラメータ。

備考

エージェントでサポートされるプロパティは次のとおりです。

- **mlstream**　Mobile Link ストリームパラメータ (たとえば、tcpip(host=localhost))。
- **max_taskdb_sync_interval**　エージェントがエージェントデータベースを同期してから次に同期するまで待機する最長時間 (秒単位)。
- **lwp_freq**　ライトウェイトポーリングの時間間隔。

参照

- 「ml_ra_clone_agent_properties システムプロシージャ」676 ページ

ml_ra_unmanage_remote_db システムプロシージャ

このプロシージャを使用して、リモートデータベースの定義を保持しますが、データベースがエージェントによって管理されないように、リモートデータベースとリモートエージェント間のリンクを切断します。

パラメータ

構文	説明
remote_id	VARCHAR(128)。切断するリモート ID を指定する IN パラメータ。
schema_name	VARCHAR(128)。リモートデータベースのタイプを指定する IN パラメータ。

備考

リモートデータベースにタスクが割り当てられている場合、このプロシージャは失敗します。

リモートデータベースを別のエージェントで管理する必要がある場合は、新しい agent_id を指定して ml_ra_manage_remote_db プロシージャを再度呼び出します。

参照

- 「ml_ra_manage_remote_db システムプロシージャ」688 ページ

ml_reset_sync_state システムプロシージャ

このプロシージャを使用して、Mobile Link システムテーブル内の同期ステータス情報をリセットします。

構文

ml_reset_sync_state (
'*user*',
'*remote_id*'
)

パラメータ

構文	説明
user	VARCHAR(128)。Mobile Link ユーザ名。
remote_id	VARCHAR(128)。リモート ID。

備考

パラメータには NULL を指定できます。両方のパラメータが NULL の場合、このプロシージャは何も処理を実行しません。

このストアドプロシージャは、指定のユーザ名とリモート ID に対する ml_subscription テーブルの progress、last_upload_time、last_download_time カラムに、デフォルト値を設定します。progress のデフォルト値は 0 です。last_upload_time と last_download_time カラムのデフォルト値は '1900/01/01 00:00:00' です。

リモート ID が NULL で、Mobile Link ユーザ名が NULL でない場合、このプロシージャはこれらのカラムを、指定の Mobile Link ユーザ名によって参照される ml_subscription テーブル内の

ローのデフォルト値に設定します。Mobile Link ユーザ名が NULL で、リモート ID が NULL でない場合には、指定のリモート ID を持つ ml_subscription テーブル内のローのデフォルト値に設定します。

このストアドプロシージャは、細心の注意を払って使用してください。次回 Mobile Link クライアントがこのリモート ID の同期を要求したとき、Mobile Link サーバは、このリモート ID の同期ステータスをチェックしません。前回行われた同期が成功しなかったリモート ID をリセットすると、データの不整合が発生する場合があります。

ml_server_delete システムプロシージャ

このプロシージャは内部でのみ使用されます。

ml_server_update システムプロシージャ

このプロシージャは内部でのみ使用されます。

Mobile Link ユーティリティ

Mobile Link サーバは、次のユーティリティを備えています。

- 「Mobile Link 停止ユーティリティ (mlstop)」
- 「Mobile Link ユーザ認証ユーティリティ (mluser)」
- 「Mobile Link Replay ユーティリティ (mlreplay)」
- 「生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi)」
- 「Windows 用 Mobile Link 監視サーバユーティリティ (mlarbiter)」
- 「UNIX 用 Mobile Link 監視サーバユーティリティ (mlarbiter.sh)」
- 「Mobile Link 監視停止ユーティリティ (mlarbstop)」707 ページ

参照

- 「Mobile Link クライアントユーティリティ」『Mobile Link クライアント管理』
- 「Ultra Light ユーティリティ」『Ultra Light データベース管理とリファレンス』
- 「証明書ユーティリティ」『SQL Anywhere サーバ データベース管理』
- 「データベース管理ユーティリティ」『SQL Anywhere サーバ データベース管理』

Mobile Link 停止ユーティリティ (mlstop)

ローカルコンピュータ上の Mobile Link サーバを停止します。

構文

mlstop [*options*] [*name*]

オプション	説明
@data	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。「設定ファイル」『SQL Anywhere サーバ データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
-h	ハードシャットダウン。Mobile Link がすべての同期を停止して終了します。リモートによっては、エラーをレポートする場合があります。
-q	クワイエットモード。このモードにすると、バナーは表示されません。
-t *time*	ソフトシャットダウン。ただし、指定の時間が経過したらハードシャットダウン。*time* は数字で、後ろに D、H、M、または S (日、時間、分、秒) が続きます。たとえば、-t 10m と指定すると、10 分後または現在の同期が完了した時点 (早い方) で、サーバがシャットダウンされます。D、H、M、S の大文字と小文字は区別されません。
-w	コマンドから戻る前に、Mobile Link サーバがシャットダウンされるのを待機します。
name	-zs オプションで Mobile Link サーバを起動した場合は、同じサーバ名を指定してシャットダウンしてください。「-zs mlsrv16 オプション」104 ページを参照してください。

備考

デフォルト (-h、-t のいずれも指定されていない場合) では、mlstop によってソフトシャットダウンが行われます。

- **ソフトシャットダウン**　Mobile Link サーバは現在の同期が完了すると、新しい接続を受け入れずに終了します。
- **ハードシャットダウン**　Mobile Link サーバはすべての同期を停止して終了します。リモートによっては、エラーをレポートする場合があります。

Mobile Link ユーザ認証ユーティリティ (mluser)

統合データベース側で Mobile Link ユーザを登録します。SQL Anywhere リモートの場合は、CREATE SYNCHRONIZATION USER 文を使用して、リモートデータベース側でユーザを事前に作成しておきます。

構文

mluser [*options*] **-c** "*connection-string*"
{ **-f** *file* | **-u** *user* [**-p** *password*] }

オプション	説明
@*data*	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。「設定ファイル」『SQL Anywhere サーバ データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
-c "*keyword=value;...*"	これを使用して、データベース接続パラメータを指定します。接続文字列では、ODBC データソースを使用して統合データベースに接続するのに十分な権限をユーティリティに指定する必要があります。このパラメータは必須です。
-d	-f または -u で指定されたユーザ名を削除します。このオプションは、mluser -r オプションとは一緒に使用できません。
-f *filename*	指定したファイルから、ユーザ名とパスワードを読み込みます。ファイルは、1 行ごとにユーザ名とパスワードが 1 つずつ、空白スペースで区切って指定されているものを使用します。-f または -u を指定してください。
-fips	設定すると、FIPS 認定の暗号化のサポートがインストールされていない場合に mluser が失敗します。
-n *ユーザ認証ポリシー名*	Mobile Link を LDAP ユーザ認証に登録します。
-o *filename*	指定したファイルに出力メッセージのログを取ります。

オプション	説明
-ot *filename*	ログファイルをトランケートし、このファイルに出力メッセージを追加します。デフォルトでは、画面に出力を送信します。
-pc *collation-id*	ユーザ名とパスワードの文字セット変換用のデータベース照合 ID を指定します。 ここには、「代替照合」『SQL Anywhere サーバ データベース管理』にリストされている SQL Anywhere 照合ラベルの 1 つを指定します。 このオプションは、ファイルから読み込まれるユーザ名とパスワードが、ロケールで決定されるデフォルトの文字セットとは異なる文字セットでエンコードされている場合に必要です。
-p *password*	ユーザと関連付けるパスワード。このオプションは、-u を指定した場合のみ使用できます。
-r *remote-id*	このオプションを -u *username* と一緒に使用します。指定したリモート ID とユーザ名の同期状態が mluser によってリセットされます。ml_subscription テーブル内の last_upload_time カラムと last_download_time カラムは、指定されたユーザ名とリモート ID に対応するそれぞれのデフォルト値にリセットされます。progress、last_upload_time、last_download_time のデフォルト値は、それぞれ 0、'1900/01/01 00:00:00'、'1900/01/01 00:00:00' です。 このオプションは、mluser -d オプションとは一緒に使用できません。 **警告** このオプションは、指定されたユーザ名とリモート ID の同期状態をリセットします。このアクションは取り消せません。同期ステータスがリセットされた後、Mobile Link サーバは、常に、最後の同期ステータスをチェックしないでクライアントからの最初の同期要求を受け入れます。
-u *username*	追加するユーザ名を指定します (-d と一緒に使用した場合、ユーザを削除します)。1 行のコマンドラインで指定できるユーザは 1 人だけです。パスワードを使用している場合は、このオプションを -p とともに使用します。-f または -u を指定してください。

オプション	説明
-v	冗長ロギングを指定します。

備考

ユーザとパスワードのペアを指定すると、mluser ユーティリティはまずそのユーザを追加しようとします。ユーザをすでに統合データベースに追加してある場合は、そのユーザのパスワードを更新します。

ユーザ名を統合データベースに登録するには、別の方法も使用できます。

- Sybase Central を使用する。
- mlsrv16 で -zu+ コマンドラインオプションを指定する。この場合、最初に同期するときに、統合データベースに追加されていない既存の Mobile Link ユーザが追加されます。

追加されるのは、リモートデータベースにすでに存在する Mobile Link ユーザです。リモートデータベース側でユーザを追加する場合、次のオプションがあります。

- SQL Anywhere リモートの場合、CREATE SYNCHRONIZATION USER を使用して名前を設定し、そのユーザ名で同期する。
- Ultra Light リモートの場合は、ul_sync_info 構造体の user_name フィールドを使用するか、Java で、ULSynchInfo クラスの SetUserName() メソッドを使用してから同期する。

参照

- 「Mobile Link ユーザ」『Mobile Link クライアント管理』
- 「-zu mlsrv16 オプション」 104 ページ
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ SQL リファレンス』
- 「トランスポートレイヤセキュリティ」『SQL Anywhere サーバ データベース管理』

Mobile Link Replay ユーティリティ (mlreplay)

mlreplay ユーティリティは、Mobile Link サーバによって記録された Mobile Link プロトコル情報をリプレイするためのツールです。

構文

mlreplay [*options*] [*name=value* [*name2=value2...*]] [[*dll_name*] *filename*]

オプション	説明
@*data*	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。 「設定ファイル」『SQL Anywhere サーバ データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
-ap	mlreplay ユーティリティによって Mobile Link サーバ上で進行オフセット不一致警告が生成されないように、リプレイセッションでリプレイされている同期の進行を調整します。さらに、シーケンス番号を調整してシーケンス番号エラーを回避します。
-f *time_scale_factor*	記録された回数に均等に適用される乗数。
-ldt *last_download_time*	リプレイセッション中に Mobile Link サーバに送信する最終ダウンロード時間を指定します。リプレイ中の記録されたプロトコルに複数の同期が含まれている場合 (このことは、永続的な接続が記録された場合に発生する可能性があります)、最初の最終ダウンロード時間のみが置き換えられ、残りは、Mobile Link サーバがリプレイセッション中に mlreplay に送信する最終ダウンロード時間によって置き換えられます。-ldt オプションを使用しない場合も、mlreplay では最初の同期以外の最終ダウンロード時間を、リプレイセッション中に Mobile Link サーバから受信した最終ダウンロード時間に置き換えます。また、最終ダウンロード時間は、シミュレートされたクライアント情報ファイルを使用して指定する (-sci オプションを使用する場合) か、DLL が提供されている場合は IdentifySimulatedClient コールバックによって指定することもできます。

オプション	説明
-ls	実行時間の合計、リプレイに費やした時間の合計、(成功、エラー、またはスキップの) 繰り返し数の合計を、シミュレートされたクライアントごとに記録します。このオプションが指定されていないときでも、終了する前には、mlreplay によってこの情報が記録されます。
-n *number_of_simulated_clients*	実行するシミュレートされたクライアントの数。最小値は 1 です。 このオプションは、-n で指定されシミュレートされたクライアントの数がシミュレートされたクライアント情報ファイルのシミュレートされたクライアント数と同じか少ないときには、-sci オプションと同時に使用できます。同時に使用するときは、-n では実行するシミュレートされたクライアントの数を指定します。これらのオプションを使用することにより、1 つのシミュレートされたクライアント情報ファイルにシミュレートされたクライアント数 *x* を指定することによって、1 ～*x* の範囲の数のシミュレートされたクライアントのプロトコルをリプレイできます。
-o *file*	コマンドラインオプションと出力メッセージを指定したファイルに記録します。
-os *size*	ログファイルの最大サイズを制限します。ログファイルが指定したサイズ (最小で 10 KB) に達すると、*YYMMDDxx.rlg* という名前に変更され、元のファイル名で新しいログファイルの記録が開始されます。
-ot *file*	ログファイルをトランケートします。コマンドラインオプションと出力メッセージを指定したファイルに記録します。
-p *password*	パスワードを指定のパスワードに置き換えます。

オプション	説明
-ping *seconds*	Mobile Link サーバに ping を実行し、サーバに同期を受信する準備が整っているかどうかを確認します。デフォルトでは、mlreplay はサーバに 60 秒間 ping を実行します。 -ping オプションを使用する場合は、次のリターンコードが有効です。 ● **-1** エラーが発生したことを示します。 ● **0** mlreplay がサーバに ping を実行でき、サーバは同期を受信する準備が整っていることを示します。 ● **1** mlreplay はサーバに ping を実行しようとしたものの、応答がなかった、つまり、サーバは同期を受信する準備が整っていないことを示します。
-r *remote ID*	リモート ID を指定のリモート ID に置き換えます。このオプションは -rg オプションと同時に使用できません。
-rep *number_of_repetitions*	シミュレートクライアントが記録されたプロトコルをリプレイする回数を指定します。リプレイ DLL または共有オブジェクトが使用されている場合は、繰り返しごとにカスタマイズできます。生成されたリプレイ API を使用するときは、繰り返しが発生するたびにコールバック GetUploadTransaction、GetDownloadApplyTime、ReportEndOfReplay、および DelayStartOfReplay が呼び出されます。
-rg	リモート ID を GUID に置き換えます。
-rnt *seconds*	指定した秒数が経過した時点で、プロトコルリプレイの新しい繰り返しを開始するように、シミュレートされたクライアントに指示します。シミュレートされたクライアントは停止されませんが、それ以降は繰り返しが開始されません。 このオプションを使用すると、API コールバックの **numRepetitions** パラメータはすべて 0 に設定されます。

オプション	説明
-rp *pattern*	コマンドラインで指定されたユーザ名、パスワード、およびリモート ID を検索し、指定したパターンにマッチした部分を、シミュレートされたクライアントの番号で置き換えます。
-sci *file*	同期のリプレイに使用するユーザ名、パスワード、リモート ID、最終ダウンロード時刻、スクリプトバージョンのリストを mlreplay に提供します。mlreplay ではファイルの行ごとにシミュレートされたクライアントを作成し、このクライアント情報を使用して記録されたプロトコルをリプレイします。各行のフォーマットは、[ユーザ名]、[パスワード]、[リモート ID]、[最終ダウンロード時刻]、[スクリプトバージョン] にします。最終ダウンロード時刻のフォーマットは、yyyy-MM-dd hh:mm:ss.SSS です。ユーザ名、パスワード、最終ダウンロード時刻、またはスクリプトバージョンのフィールドを空白にすると、mlreplay では、記録されたプロトコルの対応する値が使用されます。リモート ID を空白にすると、mlreplay では、リモート ID が GUID で置き換えられます。-u、-p、-r、-rg、-ldt、-sv の各オプションはこのオプションと同時に使用できません。DLL でも同様です。 このオプションは、-n で指定されたシミュレートされたクライアントの数がシミュレートされたクライアント情報ファイルのシミュレートされたクライアント数と同じか少ないときには、-n オプションと同時に使用できます。同時に使用するときは、-n では実行するシミュレートされたクライアントの数を指定します。これらのオプションを使用することにより、1 つのシミュレートされたクライアント情報ファイルにシミュレートされたクライアント数 *x* を指定することによって、1 ～*x* の範囲の数のシミュレートされたクライアントのプロトコルをリプレイできます。
-sv *script version*	スクリプトバージョンを指定されたスクリプトバージョンに置換します
-u *user name*	ユーザ名を指定のユーザ名に置き換えます。

オプション	説明
-x *stream(opts)*	Mobile Link サーバへの接続に使用するプロトコルストリームとストリームオプション。このオプションでは活性タイムアウトを設定できます。活性タイムアウトは、Mobile Link サーバで使用されている内容に基づいて自動的に調整されます。

備考

オプションの dll_name パラメータは、mlreplay が使用するリプレイ DLL の名前です。リプレイ DLL は、リプレイ API でコンパイルされます。「生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi)」704 ページを参照してください。

「名前=値」ペアは、リプレイ API のコマンドライン引数に似ています。このペアにはすべての mlreplay コールバックでアクセスできます。このペアを使用してリプレイ DLL の動作をカスタマイズできます。このペアは、リプレイ DLL が使用されている場合にのみ使用されます。たとえば、同じリプレイ DLL を使用して各種データベース (mlreplay のインスタンスが異なる) への同期を実行し、同期の終了時にデータベースに接続してデータが正常にアップロードされたことを確認する場合は、リプレイ DLL 内に接続文字列をハードコードするのでなく、「名前=値」ペアを使用してデータベースの接続文字列を指定できます。

記録された各ファイルは、「記録されたプロトコルファイル」と呼ばれます。1 つの接続の開始から終了までに受信されたすべてのデータは、個別の記録されたプロトコルファイルに記録されます。記録されたプロトコルファイルには *recorded_protocol_x.mlr* という名前が付けられます。*x* はジョブ ID です。Mobile Link サーバの -rp オプションを使用すると、Mobile Link サーバがクライアントから受信するすべての Mobile Link プロトコルを記録するように指定できます。「-rp mlsrv16 オプション」75 ページを参照してください。

記録されたプロトコルファイルには、Mobile Link サーバとの間で送受信されるデータ以外に、タイミング情報も含まれます。これは、mlreplay が記録されたプロトコル情報を最初の実行時とまったく同じようにリプレイできるようにするためです。また、タイミング情報は、シミュレートされたクライアントの所要時間を元のクライアントの所要時間と同じにするためにも使用されます。

デフォルトでは、mlreplay は記録されたプロトコルファイルを変更しないでプレイバックします。ただし、さまざまなオプションを使用して、リプレイセッションをカスタマイズできます。シミュレートされたクライアント情報は、ユーザ名、パスワード、リモート ID、最終ダウンロード時間、スクリプトバージョンで構成されます。この情報は、-u、-p、-r (または -rg)、-ldt、-sv の各オプションを使用してそれぞれカスタマイズできます。

mlreplay ユーティリティでは、複数の異なるシミュレートクライアントを使用して記録されたプロトコルファイルを同時にリプレイできます。このようにするには、次の 3 つの方法があります。

- **コマンドラインのみの使用**　-n、-u、-p、-sv、-r、-rg、-rp の各オプションを組み合わせて使用することにより、記録されたプロトコルファイルを同時にリプレイできます。シミュレートクライアントの数を指定するには -n オプションを使用します。一方、各クライアントに関する情報を指定するには、-u、-p、-sv、-r、-rg を使用します。 デフォルトでは、-u、-p、-sv、

および -r、またはそのいずれかを使用する場合 (何度でも必要なだけ) に、アスタリスク (*) を指定できます。これを受けて、mlreplay はアスタリスクをシミュレートクライアント番号に置換します。アスタリスクを他の文字に変更するには、-rp オプションを使用します。

たとえば、mlreplay -ap -x tcpip -n 2 -rp $ -u user_$ -p pwd_$ -r rid_$ -sv test_script recorded_protocol.mlr は、2 つのシミュレートされたクライアントで mlreplay を実行します。シミュレートされたクライアント 1 には次の情報があります。

○ ユーザ：user_1
○ パスワード：pwd_1
○ リモート ID：rid_1
○ スクリプトバージョン：test_script

シミュレートされたクライアント 2 には次の情報があります。

○ ユーザ：user_2
○ パスワード：pwd_2
○ リモート ID：rid_2
○ スクリプトバージョン：test_script

オプションのいずれかが省略された場合は、次のルールが使用されます。

○ ユーザ名、パスワード、またはスクリプトバージョンが指定されていない場合、シミュレートされたクライアントは記録されたプロトコルファイルに記録されているユーザ名、パスワード、またはスクリプトバージョンを使用します。
○ リモート ID が指定されず、シミュレートされたクライアントの数が 1 より大きい場合には、リモート ID ごとに異なる GUID が自動的に生成されます。シミュレートされたクライアントの数が 1 のときは、記録されたプロトコルファイルに記録されているリモート ID が使用されます。GUID を強制的に生成するには、-rg オプションを使用します。
○ ユーザ名、パスワード、リモート ID、またはスクリプトバージョンは指定されているが、それらにアスタリスク (*) が含まれていない場合 (または、-rp オプションで指定されている文字が何であろうと)、シミュレートされたクライアントはそれぞれ、同じユーザ名、パスワード、リモート ID、またはスクリプト ID を使用します。

● **シミュレートされたクライアント情報ファイルの使用**　-sci オプションを使用して、シミュレートされたクライアント情報ファイルを指定することにより、記録されたプロトコルファイルを同時にリプレイすることができますシミュレートされたクライアント情報ファイルは *.csv* ファイルです。このファイルの各行には、ユーザ名、パスワード、リモート ID、最終ダウンロード時間、スクリプトバージョンが (この順番で) 含まれます。

mlreplay ユーティリティは、**Using only the command line** オプションの場合と同じルールで空のフィールドに入力します。

デフォルトでは、mlreplay はシミュレートされたクライアント情報ファイル内の情報行ごとにシミュレートされたクライアントを作成します。ただし、-n オプションを -sci ファイルと一緒に使用すれば、シミュレートされたクライアントの数を制限できます。シミュレートされたクライアント情報ファイルで *x* 個のシミュレートされたクライアントが指定されている場合は、-n オプションを使用して 1 から *x* までの数を指定できます。この場合、mlreplay では、その指定されたクライアント数だけが使用されます。

シミュレートされたクライアント情報ファイルの使用は、単にコマンドラインを使用する場合より柔軟ですが、リプレイ DLL を使用する場合よりは柔軟性が乏しくなります。

- **リプレイ DLL の使用**　リプレイ DLL を使用するときは、-n オプションを使用して、シミュレートされたクライアント数を指定します。他の情報はすべて、ユーザによって実装されたコールバックが mlreplay によって呼び出されたときに取得されます。このアプローチはかなり柔軟性に優れており、リプレイの他の部分もカスタマイズできます。「Mobile Link Replay C++ コールバック」635 ページと「生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi)」704 ページを参照してください。

mlreplay ユーティリティは、コマンドラインから複数のシミュレートされたクライアントを実行して、プロトコルをリプレイできます。実行するシミュレートされたクライアントの数は -n オプションで指定できます。-u、-p、-r、-sv オプションでそれぞれユーザ名、パスワード、リモート ID、スクリプトバージョンを指定するときには、アスタリスク記号を使用して、シミュレートされたクライアントの数を指定できます。それぞれのシミュレートされたクライアントのユーザ名、パスワード、リモート ID、スクリプトバージョンは、次の規則に従って決定されます。

- ユーザ名またはパスワードが指定されないときは、すべてのシミュレートされたクライアントでは、リプレイされる記録されたプロトコルファイルのユーザ名またはパスワードが使用されます。

- リモート ID が指定されないときに、シミュレートされたクライアントの数が 1 より大きい場合は、それぞれのリモート ID は自動的に生成された GUID になります。シミュレートされたクライアントの数が 1 のときは、記録されたプロトコルファイルのリモート ID が使用されます。-rg オプションを使用すれば、GUID の値を強制することもできます。

- 指定したユーザ名、パスワード、リモート ID にアスタリスクが含まれていない場合は、シミュレートされたクライアントでは、同じユーザ名、パスワード、リモート ID が使用されます。指定したユーザ名、パスワード、リモート ID に少なくとも 1 つのアスタリスク記号が含まれている場合は、シミュレートされたクライアントでは、アスタリスクの場所がシミュレートされたクライアントの番号で置き換えられるため、それぞれ固有のユーザ名、パスワード、リモート ID が使用されることになります。

記録内容には元の同期に要した時間が含まれるため、mlreplay では同じ長さの時間だけ同期をリプレイすることを試行できます。

mlreplay ユーティリティで次の Mobile Link サーバオプションを使用します。

- **-rp**　mlreplay ユーティリティでプレイバックする同期の記録元ディレクトリを指定するには、このオプションを使用します。

- **-rrp**　Mobile Link サーバの起動時に mlreplay ユーティリティを実行するには、このオプションを使用します。

- **-lsc**　mlreplay ユーティリティがローカルサーバに接続できるようにサーバの接続情報を指定するには、このオプションを使用します。

リプレイセッションは、生成された Mobile Link リプレイ API を使用して、さらにカスタマイズできます。

参照

- 「生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi)」704 ページ
- 「-rrp mlsrv16 オプション」74 ページ
- 「-rp mlsrv16 オプション」75 ページ
- 「-lsc mlsrv16 オプション」63 ページ

生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi)

mlgenreplayapi ツールでは、記録されたプロトコルファイルを読み込み、そのファイルのスキーマ用の「Mobile Link リプレイ API」 を生成します。

構文

mlgenreplayapi [*options*] *filename*

オプション	説明
@*data*	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。 「設定ファイル」『SQL Anywhere サーバ データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化します。「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ データベース管理』を参照してください。
-d *directory*	生成されたファイルを出力するディレクトリ。
-o *file*	コマンドラインオプションと出力メッセージを指定したファイルに記録します。
-os*size*	ログファイルの最大サイズを制限します。ログファイルが指定したサイズ (最小で 10 KB) に達すると、*YYMMDDxx.rlg* という名前に変更され、元のファイル名で新しいログファイルの記録が開始されます。
-ot *file*	ログファイルをトランケートします。コマンドラインオプションと出力メッセージを指定したファイルに追加します。デフォルトでは、画面に出力を送信します。

備考

リプレイ API を変更すると (*mlreplaycallbacks.cpp* のコードのみ変更する必要があります)、リプレイセッション中に Mobile Link サーバにアップロードされたデータをカスタマイズできます。リプレイ API は 「リプレイ DLL」 にコンパイルされ、mlreplay ではこのリプレイ DLL を使用してリプレイセッションをカスタマイズします。リプレイ DLL とシミュレートされたクライアント情報ファイルを同時に使用することはできません。リプレイ API には、シミュレートされたクライアントごとにシミュレートされたクライアント情報を提供するために使用できるコールバックが含まれています。リプレイ DLL を使用する場合に起動するシミュレートされたクライアントの数は、-n コマンドラインオプションを使用して mlreplay に指定します。

参照

- 「Mobile Link Replay C++ コールバック」635 ページ
- 「Mobile Link Replay ユーティリティ (mlreplay)」696 ページ

Windows 用 Mobile Link 監視サーバユーティリティ (mlarbiter)

mlarbiter コマンドは Mobile Link 監視サーバを起動します。

構文

mlarbiter

備考

Mobile Link 監視サーバは、デフォルトでポート 4953 で受信します。

Mobile Link 監視サーバは、サーバファーム内の 1 台の Mobile Link サーバのみがプライマリサーバとして動作するようにします。こうすることで、サーバ起動同期環境での冗長な通知を防ぐことができます。

このコマンドは、Mobile Link サーバに監視サーバのホスト名を提供する Mobile Link サーバの -ca オプションとともに使用します。

Mobile Link サーバは、監視サーバの起動後監視サーバに接続できない場合、15 秒ごとに接続の確立を試行し、定期的にエラーメッセージを表示します。

サーバファームの Mobile Link サーバからプライマリサーバが選択された後に監視サーバの接続が切断した場合、プライマリサーバは直ちに停止し、セカンダリサーバが監視サーバへの接続の再確立を 15 秒ごとに試行します。監視サーバへの接続が確立されると、Mobile Link サーバからプライマリサーバが再度選択されます。

例

次の例は、Mobile Link サーバファームで Mobile Link 監視サーバを使用する方法を示します。

1. 次のコマンドラインを使用してコンピュータ上で Mobile Link 監視サーバを起動します。

```
mlarbiter
```

2. 次のようなコマンドラインを使用して Mobile Link サーバを起動します。Mobile Link サーバは監視サーバと同じコンピュータ上で起動したり、別のコンピュータ上で起動したりできます。

   ```
   mlsrv16 -c parameter1 -lsc parameter2 -ca Host_1 -notifier
   ```

   上記の例では、*parameter1* は統合データベースの接続パラメータ、*parameter2* はローカル Mobile Link サーバの接続パラメータです。同じサーバファーム内のすべての Mobile Link サーバに -ca オプションの同じ設定が含まれている必要があります。

参照

- 「-ca mlsrv16 オプション」 52 ページ
- 「-lsc mlsrv16 オプション」 63 ページ
- 「アーキテクチャ」『Mobile Link クイックスタート』

UNIX 用 Mobile Link 監視サーバユーティリティ (*mlarbiter.sh*)

mlarbiter.sh コマンドは、Mobile Link 監視サーバの起動と停止を行います。

構文

mlarbiter.sh [*option*]

オプション	説明
start	Mobile Link 監視サーバユーティリティを起動します。
stop	Mobile Link 監視サーバユーティリティを停止します。

備考

Mobile Link 監視サーバは、デフォルトでポート 4953 で受信します。

このコマンドは、Mobile Link サーバに監視サーバのホスト名を提供する mlsrv16 -ca オプションとともに使用します。

Mobile Link 監視サーバは、サーバファーム内の 1 台の Mobile Link サーバのみがプライマリサーバとして動作するようにします。これにより、サーバ起動同期環境での冗長な通知が回避されます。

Mobile Link サーバは、監視サーバの起動後監視サーバに接続できない場合、15 秒ごとに接続の確立を試行し、定期的にエラーメッセージを表示します。

サーバファームの Mobile Link サーバからプライマリサーバが選択された後に監視サーバの接続が切断した場合、プライマリサーバは直ちに停止し、セカンダリサーバが監視サーバへの接続の再確立を 15 秒ごとに試行します。監視サーバへの接続が確立されると、Mobile Link サーバからプライマリサーバが再度選択されます。

例

次の例は、Mobile Link サーバファームで Mobile Link 監視サーバを使用する方法を示します。

1. 次のコマンドラインを使用してコンピュータ上で Mobile Link 監視サーバを起動します。

   ```
   mlarbiter.sh start
   ```

2. 次のようなコマンドラインを使用して Mobile Link サーバを起動します。Mobile Link サーバは監視サーバと同じコンピュータ上で起動することも、別のコンピュータ上で起動することもできます。

   ```
   mlsrv16 -c parameter1 -lsc parameter2 -ca Host_1 -notifier
   ```

   上記の例では、*parameter1* は統合データベースの接続パラメータ、*parameter2* はローカル Mobile Link サーバの接続パラメータです。同じサーバファーム内のすべての Mobile Link サーバに -ca オプションの同じ設定が含まれている必要があります。

参照

- 「-ca mlsrv16 オプション」 52 ページ
- 「-lsc mlsrv16 オプション」 63 ページ
- 「アーキテクチャ」『Mobile Link クイックスタート』

Mobile Link 監視停止ユーティリティ (mlarbstop)

mlarbstop コマンドを使用して、Mobile Link 監視サーバを停止します。

構文

mlarbstop [*option*]

オプション	説明
-y	接続中でも Mobile Link 監視サーバをすぐに停止します。

備考

Mobile Link 監視サーバがローカルコンピュータで動作している場合でも、mlarbstop ユーティリティを使用して Mobile Link 監視サーバを停止できます。

-y オプションを指定しないで mlarbstop を実行したときに監視サーバへの接続が存在しない場合、監視サーバはすぐに停止されます。

-y オプションを指定しないで mlarbstop を実行したときに監視サーバへの接続が存在する場合、Mobile Link サーバはエラーを発行します。

例

この例は、ローカルコンピュータで動作している Mobile Link 監視サーバを停止する方法を示します。

1. 次のコマンドラインを使用して、ローカルコンピュータでの Mobile Link 監視を停止します。

```
mlarbstop -y
```

参照

- 「Windows 用 Mobile Link 監視サーバユーティリティ (mlarbiter)」705 ページ
- 「UNIX 用 Mobile Link 監視サーバユーティリティ (mlarbiter.sh)」706 ページ
- 「-ca mlsrv16 オプション」52 ページ
- 「-lsc mlsrv16 オプション」63 ページ
- 「アーキテクチャ」『Mobile Link クイックスタート』

リモートデータベースと統合データベース間での Mobile Link データマッピング

使用する統合データベースによって異なりますが、Mobile Link サーバでは、多くの場合、指定したデータ型を異なるデータ型にマッピングできます。次のセクションでは、SQL Anywhere または Ultra Light とサポートされている統合データベースとの間でのデータ型マッピングを示します。

- 「Adaptive Server Enterprise データのマッピング」
- 「IBM DB2 LUW データのマッピング」
- 「Microsoft SQL Server データのマッピング」
- 「MySQL データのマッピング」
- 「Oracle データのマッピング」
- 「SAP Sybase IQ Enterprise のデータマッピング」755 ページ

Adaptive Server Enterprise データのマッピング

Adaptive Server Enterprise の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように Adaptive Server Enterprise の統合データ型にマッピングされるのかを示します。たとえば、リモートデータベースの LONG VARCHAR 型のカラムは、統合データベースでは TEXT 型である必要があります。

最大カラム長 (MCL) は、Adaptive Server Enterprise のページサイズによって異なります。ページサイズが 2K の場合、MCL は 1954 になります。ページサイズが 4K の場合、MCL は 4002 になります。MCL の詳細については、Adaptive Server Enterprise のマニュアルを参照してください。

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
BIGINT	BIGINT	

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
BIT	BIT	
BINARY(*n*=<MCL)	BINARY(*n*)	
BINARY(*n*>MCL)	IMAGE	
CHAR(*n*=<MCL)	VARCHAR(*n*)	
CHAR(*n*>MCL)	TEXT	ダウンロード時に、値が長すぎないようにします。
DATE	DATE	Adaptive Server Enterprise の DATETIME では、年は 1753 ～ 9999 の範囲内である必要があります。 SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DATETIME	DATETIME[1] または BIGDATETIME[2]	Adaptive Server Enterprise の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。 DATETIME をプライマリキーに使用すると、競合を解決できないことがあります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。また、年は 1753 ～ 9999 の範囲内である必要があります。
DECIMAL(*p*<39, *s*)	DECIMAL(*p*,*s*)	Adaptive Server Enterprise の NUMERIC の精度は 1 ～ 38 桁 (*p*<39) です。

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
DECIMAL(*p*>=39,*s*)		Adaptive Server Enterprise には対応するデータ型がありません。
DOUBLE	DOUBLE PRECISION	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	IMAGE	
INTEGER	INTEGER	
LONG BINARY	IMAGE	
LONG NVARCHAR	UNITEXT	
LONG VARBIT	TEXT	
LONG VARCHAR	TEXT	
MONEY	MONEY	
NCHAR(*c*=<MCL)	UNIVARCHAR(*c*/2)	
NCHAR(*c*>MCL)	UNITEXT	ダウンロード時に、値が長すぎないようにします。
NTEXT	UNITEXT	
NUMERIC(*p*<39,*s*)	NUMERIC(*p*,*s*)	Adaptive Server Enterprise の 10 進数の精度は 1 ～ 38 桁 (*p*<39) です。
NUMERIC(*p*>=39,*s*)		Adaptive Server Enterprise には対応するデータ型がありません。
NVARCHAR(*c*=<MCL)	UNIVARCHAR(*c*/2)	
NVARCHAR(*c*>MCL)	UNIVARCHAR(*c*/2)	
REAL	REAL	

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
SMALLDATETIME	DATETIME[1] または BIGDATETIME[2]	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。 Adaptive Server Enterprise の DATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。また、年は 1753 ～ 9999 の範囲内である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	TEXT	
TIME	TIME[1] または BIGTIME[2]	Adaptive Server Enterprise の TIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。TIME をプライマリキーに使用すると、競合を解決できないことがあります。TIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
TIMESTAMP	DATETIME[1] または BIGDATETIME[2]	Adaptive Server Enterprise の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。 DATETIME をプライマリキーに使用すると、競合を解決できないことがあります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。また、年は 1753 ～ 9999 の範囲内である必要があります。
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	Adaptive Server Enterprise には対応するデータ型がありません。したがって、TIMESTAMP WITH TIME ZONE カラムを VARCHAR(34) カラムにマッピングする必要があります。アップロード時、Mobile Link サーバは、まずデータを YYYY-MM-DD HH:NN:SS.SSSSSS [+|-]HH:NN フォーマットの文字列に変換してから統合データベースに適用します。ダウンロード時は、データを文字列から TIMESTAMP WITH TIME ZONE に変換します。統合データベース内のデータがこのフォーマットに従っていない場合は、ダウンロードが失敗します。
TINYINT	TINYINT	
UNIQUEIDENTIFIER	CHAR(36)	

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR は使用しないでください。代わりに UNIQUEIDENTIFIER を使用してください。
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INTEGER	UNSIGNED INT	
UNSIGNED SMALLINT	UNSIGNED SMALLINT	
UNSIGNED TINYINT	TINYINT	
VARBINARY(*n*=<MCL)	VARBINARY	
VARBINARY(*n*>MCL)	IMAGE	
VARBIT(*n*=<MCL)	VARCHAR(*n*)	
VARBIT(*n*>MCL)	TEXT	
VARCHAR(*n*=<MCL)	VARCHAR(*n*)	
VARCHAR(*n*>MCL)	TEXT	
XML	TEXT	

[1] バージョン 15.5 より前の Adaptive Server Enterprise のみに該当します。

[2] バージョン 15.5 以降の Adaptive Server Enterprise のみに該当します。

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、Adaptive Server Enterprise の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるのかを示します。たとえば、統合データベースの DOUBLE PRECISION 型のカラムは、リモートデータベースでは DOUBLE 型である必要があります。

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	説明
BIGINT	BIGINT	
BIGDATETIME[1]	TIMESTAMP	

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	説明
BIGTIME[1]	TIME	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	
CHAR(*n*)	VARCHAR(*n*)	SQL Anywhere の CHAR/NCHAR は、Adaptive Server Enterprise の CHAR/NCHAR と同等ではありません。SQL Anywhere の CHAR/NCHAR に対応するのは、VARCHAR/NVARCHAR です。同期される統合データベースのカラムでは、CHAR/NCHAR を使用しないでください。SQL Anywhere 以外の CHAR/NCHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DATETIME	DATETIME	Adaptive Server Enterprise の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。競合は解決できない場合があります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。また、年は 1753 ～ 9999 の範囲内である必要があります。
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	説明
DOUBLE PRECISION	DOUBLE	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(*n*)	VARCHAR(*n*)	Adaptive Server Enterprise の NCHAR と NVARCHAR は、SQL Anywhere の NCHAR と NVARCHAR とは異なり、マルチバイトの各国の文字列を格納します。マルチバイト環境では、SQL Anywhere または Ultra Light の VARCHAR を使用してください。
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
NVARCHAR(*n*)	VARCHAR(*n*)	Adaptive Server Enterprise の NCHAR と NVARCHAR は、SQL Anywhere の NCHAR と NVARCHAR とは異なり、マルチバイトの各国の文字列を格納します。マルチバイト環境では、SQL Anywhere または Ultra Light の VARCHAR を使用してください。
REAL	REAL	

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	説明
SMALLDATETIME	SMALLDATETIME	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。 Adaptive Server Enterprise の SMALLDATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。また、年は 1900 〜 2078 の範囲内である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	Adaptive Server Enterprise の TIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。競合は解決できない場合があります。TIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	説明
TIMESTAMP	VARBINARY(8)	Adaptive Server Enterprise 内では、TIMESTAMP はローが変更されるたびに増分されるバイナリカウンタです。各テーブルに存在できる TIMESTAMP カラムは 1 つのみであるため、TIMESTAMP カラムを同期する意味はありません。同期に含める必要がある場合は、SQL Anywhere または Ultra Light の VARBINARY(8) データ型にマッピングします。 この TIMESTAMP カラムはサーバによって維持されるため、明示的に挿入および更新することはできません。このようなカラムがあるテーブルのアップロードスクリプトを実装する場合は、このことを念頭に置いてください。
TINYINT	TINYINT	
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INT	UNSIGNED INT	
UNSIGNED SMALLINT	UNSIGNED SMALLINT	
VARBINARY(*n*)	VARBINARY(*n*)	
VARCHAR(*n*)	VARCHAR(*n*)	
UNICHAR(*n*)	NVARCHAR(*n*)	Ultra Light では使用できません。
UNITEXT	LONG NVARCHAR	Ultra Light では使用できません。
UNIVARCHAR(*n*)	NVARCHAR(*n*)	Ultra Light では使用できません。

[1] バージョン 15.5 以降の Adaptive Server Enterprise のみに該当します。

IBM DB2 LUW データのマッピング

IBM DB2 LUW の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように IBM DB2 LUW の統合データ型にマッピングされるのかを示します。たとえば、リモートデータベースの BIT 型のカラムは、統合データベースでは SMALLINT 型である必要があります。

IBM DB2 LUW テーブルを作成する場合は、DB2 のページサイズに注意する必要があります。IBM DB2 LUW には、ページサイズに基づく最大ロー長 (MRL) があります。ページサイズが 4K の場合、MRL は 4005、8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 です。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、IBM DB2 LUW のマニュアルを参照してください。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	説明
BIGINT	BIGINT	
BINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	
BINARY(*n*>=MRL)	BLOB(*n*)	
BIT	SMALLINT	
CHAR(*n*<MRL)	VARCHAR(*n*)	
CHAR(*n*>=MRL)	CLOB(*n*)	IBM DB2 LUW の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DATETIME	TIMESTAMP	
DECIMAL(*p*<32,*s*)	DECIMAL(*p*,*s*)	SQL Anywhere の DECIMAL の精度は 1 ～ 127 です。IBM DB2 LUW DECIMAL の最大精度は 31 です。
DECIMAL(*p*>=32,*s*)		SQL Anywhere の DECIMAL で精度が 31 より大きいデータは、IBM DB2 LUW に同期できません。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	説明
DOUBLE	DOUBLE	DOUBLE は、丸め誤差が出る可能性がある概数値データ型です。種類の異なるコンピュータを使用すると、DOUBLE の基本となる記憶領域が異なることが多く、したがって、丸めの結果も異なります。プライマリキーは等しいかどうかの確認に使用されるので、プライマリキーで DOUBLE を使用することはおすすめできません。これは特に同期環境で言えます。特に、統合データベースはリモートデータベースとは異なるハードウェアで実行されることが多いからです。
FLOAT(1-24)	REAL	FLOAT はあいまいな値なので、統合データベースとリモートデータベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。使用できるすべての値をテストしたわけではないため、注意が必要です。問題の発生を回避するため、このような型をプライマリキーの一部として使用しないでください。
FLOAT(25-53)	DOUBLE	FLOAT はあいまいな値なので、統合データベースとリモートデータベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。使用できるすべての値をテストしたわけではないため、注意が必要です。問題の発生を回避するため、このような型をプライマリキーの一部として使用しないでください。
IMAGE	BLOB(*n*)	
INTEGER	INTEGER	
LONG BINARY	BLOB(*n*)	
LONG NVARCHAR	CLOB(*n*)	IBM DB2 LUW には対応するデータ型がありません。IBM DB2 LUW の文字セットがユニコードの場合は、SQL Anywhere の LONG NVARCHAR を DB2 の CLOB に同期できます。Ultra Light には LONG NVARCHAR はありません。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	説明
LONG VARBIT	CLOB(*n*)	
LONG VARCHAR	CLOB(*n*)	
MONEY	DECIMAL(19,4)	
NCHAR(*c*)	VARCHAR(*n*) または CLOB(*n*)	IBM DB2 LUW には対応するデータ型がありません。IBM DB2 LUW の文字セットが Unicode の場合は、NCHAR を IBM DB2 LUW の VARCHAR または CLOB に同期できます。SQL Anywhere の NCHAR のサイズは文字単位で、IBM DB2 LUW の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NCHAR は CLOB にマッピングします。NCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
NUMERIC(*p*<32,*s*)	NUMERIC(*p*,*s*)	
NUMERIC(*p*>=32,*s*)		IBM DB2 LUW には対応するデータ型がありません。
NTEXT	CLOB(*n*)	IBM DB2 LUW には対応するデータ型がありません。IBM DB2 LUW の文字セットが Unicode の場合は、NTEXT を IBM DB2 LUW CLOB に同期できます。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	説明
NVARCHAR(*c*)	VARCHAR(*n*) または CLOB(*n*)	IBM DB2 LUW には対応するデータ型がありません。IBM DB2 LUW の文字セットが Unicode の場合は、NVARCHAR を IBM DB2 LUW の VARCHAR または CLOB に同期できます。SQL Anywhere の NVARCHAR のサイズは文字単位で、IBM DB2 LUW の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NVARCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NVARCHAR は CLOB にマッピングします。NVARCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモートデータベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。使用できるすべての値をテストしたわけではないため、注意が必要です。問題の発生を回避するため、このような型をプライマリキーの一部として使用しないでください。
SMALLDATETIME	TIMESTAMP	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
ST_GEOMETRY	ST_GEOMETRY	
TEXT	CLOB(*n*)	

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	説明
TIME	TIMESTAMP または TIME	小数点以下の秒がある SQL Anywhere と Ultra Light の TIME 値には、IBM DB2 LUW の TIMESTAMP が必要です。小数点以下の秒が常に 0 の SQL Anywhere と Ultra Light の TIME 値には、IBM DB2 の TIME を使用できます。時刻カラムの精度を保持するため、Mobile Link サーバでは、TIME カラムが常に ODBC SQL_TYPE_TIMESTAMP データ型にバインドされます。統合データベースが DB2 9.7 サーバで実行されているときは、カラムがプライマリキーの一部である場合、DB2 変換関数を使用してカラムを TIMESTAMP と TIME の間で明示的に変換する必要があります。
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	IBM DB2 LUW には対応するデータ型がありません。したがって、TIMESTAMP WITH TIME ZONE カラムを VARCHAR(34) カラムにマッピングする必要があります。アップロード時、Mobile Link サーバは、まずデータを YYYY-MM-DD HH:NN:SS.SSSSSS [+|-]HH:NN フォーマットの文字列に変換してから統合データベースに適用します。ダウンロード時は、データを文字列から TIMESTAMP WITH TIME ZONE に変換します。統合データベース内のデータがこのフォーマットに従っていない場合は、ダウンロードが失敗します。
TINYINT	SMALLINT	ダウンロードの場合、IBM DB2 LUW の値は負でない必要があります。
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR の IBM DB2 LUW での使用はおすすめしません。代わりに UNIQUEIDENTIFIER を使用してください。
UNSIGNED BIGINT	DECIMAL(20)	ダウンロードの場合、IBM DB2 LUW の値は負でない必要があります。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	説明
UNSIGNED INTEGER	DECIMAL(11)	ダウンロードの場合、IBM DB2 LUW の値は負でない必要があります。
UNSIGNED SMALLINT	DECIMAL(5)	ダウンロードの場合、IBM DB2 LUW の値は負でない必要があります。
UNSIGNED TINYINT	SMALLINT	ダウンロードの場合、IBM DB2 LUW の値は負でない必要があります。
VARBINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	
VARBINARY(*n*>=MRL)	BLOB(*n*)	
VARBIT(*n*<MRL)	VARCHAR(*n*)	
VARBIT(*n*>=MRL)	CLOB(*n*)	
VARCHAR(*n*<MRL)	VARCHAR(*n*)	
VARCHAR(*n*>=MRL)	CLOB(*n*)	IBM DB2 LUW の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
XML	CLOB(*n*)	

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、IBM DB2 LUW の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるのかを示します。たとえば、統合データベースの INT 型のカラムは、リモートデータベースでは INTEGER 型である必要があります。

IBM DB2 LUW テーブルを作成する場合は、ページサイズに注意する必要があります。IBM DB2 LUW には、ページサイズに基づく最大ロー長があります。ページサイズが 4K の場合、MRL は 4005、8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 です。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、IBM DB2 LUW のマニュアルを参照してください。

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
BLOB	LONG BINARY	

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
BIGINT	BIGINT	
CHAR(*n*)	VARCHAR(*n*)	SQL Anywhere には IBM DB2 LUW の CHAR に対応するデータ型がありません。同期される統合データベースのカラムでは、CHAR を使用しないでください。IBM DB2 LUW の CHAR カラムを同期させる必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
CHAR(*n*) FOR BIT DATA	BINARY(*n*)	
CLOB(*n*)	LONG VARCHAR	
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DB2GSE.ST_GEOMETRY	ST_GEOMETRY	
DBCLOB(*n*)	LONG VARCHAR	DBCLOB(*n*) データ型は 2 バイト文字用にのみ使用します。SQL Anywhere には対応するデータ型がありません。IBM DB2 LUW の文字セットがユニコードの場合、DBCLOB(*n*) は CLOB と同じです。
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
DOUBLE	DOUBLE	DOUBLE は、丸め誤差が出る可能性がある概数値データ型です。種類の異なるコンピュータを使用すると、DOUBLE の基本となる記憶領域が異なることが多く、したがって、丸めの結果も異なります。プライマリキーは等しいかどうかの確認に使用されるので、プライマリキーで DOUBLE を使用することはおすすめできません。これは特に同期環境で言えます。特に、統合データベースはリモートデータベースとは異なるハードウェアで実行されることが多いからです。

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
FLOAT	DOUBLE	FLOAT はあいまいな値なので、統合データベースとリモートデータベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。使用できるすべての値をテストしたわけではないため、注意が必要です。問題の発生を回避するため、このような型をプライマリキーの一部として使用しないでください。
GRAPHIC(*n*)	VARCHAR(2*n*)	IBM DB2 LUW の GRAPHIC にはブランクが埋め込まれますが、SQL Anywhere の CHAR には埋め込まれません。このデータ型は使用しないことをおすすめします。 GRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere には対応するデータ型がありません。IBM DB2 LUW の文字セットが Unicode の場合、GRAPHIC は CHAR と同じです。
INT	INTEGER	
LONG VARCHAR	VARCHAR(32700)	
LONG VARCHAR FOR BIT DATA	VARBINARY(32700)	
LONG VARGRAPHIC(*n*)	VARCHAR(32700)	LONG VARGRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere には対応するデータ型がありません。IBM DB2 LUW の文字セットがユニコードの場合、LONG VARGRAPHIC は LONG VARCHAR と同じです。
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモートデータベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。使用できるすべての値をテストしたわけではないため、注意が必要です。問題の発生を回避するため、このような型をプライマリキーの一部として使用しないでください。
SMALLINT	SMALLINT	
TIME	TIME	SQL Anywhere TIME 値の小数点以下の秒の値は、ダウンロード時にトランケートされます。問題を回避するには、小数点以下の秒を使用しないでください。TIME カラムの精度を保持するため、Mobile Link サーバでは、時刻カラムが常に ODBC SQL_TYPE_TIMESTAMP データ型にバインドされます。統合データベースが DB2 9.7 サーバで実行されているときは、カラムがプライマリキーの一部である場合、DB2 変換関数を使用してカラムを TIMESTAMP と TIME の間で明示的に変換する必要があります。
TIMESTAMP	TIMESTAMP	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(*n*) FOR BIT DATA	VARBINARY(*n*)	
VARGRAPHIC(*n*)	VARCHAR(2*n*)	VARGRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere には対応するデータ型がありません。IBM DB2 LUW の文字セットがユニコードの場合、VARGRAPHIC は VARCHAR と同じです。

Microsoft SQL Server データのマッピング

Microsoft SQL Server の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように Microsoft SQL Server の統合データ型にマッピングされるのかを示します。たとえば、リモートデータベースの DATETIME 型のカラムは、統合データベースでは DATETIME2 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	説明
BIGINT	BIGINT	
BINARY(*n*<=8000)	VARBINARY(*n*)	
BINARY(*n*>8000)	VARBINARY(MAX)	
BIT	BIT	
CHAR(*n*<=8000)	VARCHAR(*n*)	
CHAR(*n*>8000)	VARCHAR(MAX)	
DATE	DATE	
DATETIME	DATETIME2	Microsoft SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
DECIMAL(*p*=<38,*s*)	DECIMAL(*p*,*s*)	Microsoft SQL Server の DECIMAL/NUMERIC の精度は 1 ～ 38 なので、*p* は 39 より小さい必要があります。
DECIMAL(*p*>38,*s*)		Microsoft SQL Server には対応するデータ型がありません。
DOUBLE	FLOAT(53)	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	VARBINARY(MAX)	
INTEGER	INT	

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	説明
LONG BINARY	VARBINARY(MAX)	
LONG NVARCHAR	NVARCHAR(MAX)	
LONG VARBIT	VARCHAR(MAX)	
LONG VARCHAR	VARCHAR(MAX)	
MONEY	MONEY	
NCHAR(*n*<=4000)	NVARCHAR(*c*)	
NCHAR(*n*>4000)	NVARCHAR(MAX)	
NTEXT	NVARCHAR(MAX)	
NUMERIC(*p*=<38,*s*)	NUMERIC(*p*,*s*)	Microsoft SQL Server の DECIMAL/NUMERIC の精度は 1 ～ 38 なので、*p* は 39 より小さい必要があります。
NUMERIC(*p*>38,*s*)		Microsoft SQL Server には対応するデータ型がありません。
NVARCHAR(*n*<=4000)	NVARCHAR(*c*)	
NVARCHAR(*n*>4000)	NVARCHAR(MAX)	
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。Microsoft SQL Server の SMALLDATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。年は、1900 ～ 2078 の範囲内である必要があります。
SMALLINT	SMALLINT	

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	説明
SMALLMONEY	SMALLMONEY	
ST_GEOMETRY	GEOMETRY	
TEXT	VARCHAR(MAX)	
TIME	TIME	Microsoft SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
TIMESTAMP	DATETIME2	Microsoft SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
TIMESTAMP WITH TIME ZONE	DATETIMEOFFSET	
TINYINT	TINYINT	ダウンロードの場合、値は負でない必要があります。
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNIQUEIDENTIFIERSTR	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	NUMERIC(20)	ダウンロードの場合、値は負でない必要があります。
UNSIGNED INTEGER	NUMERIC(11)	ダウンロードの場合、値は負でない必要があります。
UNSIGNED TINYINT	TINYINT	ダウンロードの場合、値は負でない必要があります。

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	説明
UNSIGNED SMALLINT	INT	ダウンロードの場合、値は負でない必要があります。
VARBINARY(*n*<=8000)	VARBINARY(*n*)	
VARBINARY(*n*>8000)	VARBINARY(MAX)	
VARBIT(*n*<=8000)	VARCHAR(*n*)	
VARBIT(*n*>8000)	VARCHAR(MAX)	
VARCHAR(*n*<=8000)	VARCHAR(*c*)	
VARCHAR(*n*>8000)	VARCHAR(MAX)	
XML	XML または VARCHAR(MAX)	Microsoft SQL Server 2005 の場合は、XML を使用します。それ以外のバージョンの場合は、VARCHAR(MAX) を使用します。

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、Microsoft SQL Server の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるのかを示します。たとえば、リモートデータベースの TEXT 型のカラムは、統合データベースでは LONG VARCHAR 型である必要があります。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	説明
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	

 Copyright © 2013, SAP AG or an SAP affiliate company.

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	説明
CHAR(*n*)	VARCHAR(*n*)	Microsoft SQL Server の CHAR カラムは、ブランクが埋め込まれます。SQL Anywhere の CHAR カラムは、デフォルトではブランクが埋め込まれず、VARCHAR カラムと同等になります。そのため、Microsoft SQL Server の同期テーブルでは CHAR データ型の使用を避けるようにしてください。Microsoft SQL Server 統合データベースで CHAR データ型を使用する必要がある場合は、SQL Anywhere の CHAR と SQL Anywhere 以外の CHAR との違いを解決できるように、-b コマンドラインオプションを指定して Mobile Link サーバを実行してください。
DATE	DATE	
DATETIME	TIMESTAMP または DATETIME	Microsoft SQL Server の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。ダウンロード時には、SQL Anywhere は SQL Server からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。DATETIME をプライマリキーに使用すると、競合を解決できないことがあります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。年は、1753 ～ 9999 の範囲内である必要があります。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	説明
DATETIME2	TIMESTAMP	Microsoft SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
FLOAT(*p*)	FLOAT(*p*)	
GEOMETRY	ST_GEOMETRY	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(*n*)	NVARCHAR(*c*)	Ultra Light では使用できません。 SQL Anywhere の NCHAR は、SQL Anywhere 以外のデータベースの NCHAR と同等ではありません。SQL Anywhere の NCHAR に対応するのは NVARCHAR です。同期される統合データベースのカラムでは、NCHAR を使用しないでください。SQL Anywhere 以外の NCHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
NTEXT	LONG NVARCHAR	Ultra Light では使用できません。
NVARCHAR(*c*)	NVARCHAR(*c*)	Ultra Light では使用できません。
NVARCHAR(MAX)	LONG NVARCHAR	Ultra Light では使用できません。
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	説明
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモートデータベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。使用できるすべての値をテストしたわけではないため、注意が必要です。問題の発生を回避するため、このような型をプライマリキーの一部として使用しないでください。
SMALLDATETIME	SMALLDATETIME	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。Microsoft SQL Server の SMALLDATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。年は、1900 ～ 2078 の範囲内である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	Microsoft SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	説明
TIMESTAMP	VARBINARY(8)	Microsoft SQL Server 内では、TIMESTAMP はローが変更されるたびに増分されるバイナリカウンタです。各テーブルに存在できる TIMESTAMP カラムは 1 つのみであるため、TIMESTAMP カラムを同期する意味はありません。同期に含める必要がある場合は、SQL Anywhere または Ultra Light の VARBINARY(8) データ型にマッピングします。 この TIMESTAMP カラムはサーバによって維持されるため、明示的に挿入および更新することはできません。このようなカラムがあるテーブルのアップロードスクリプトを実装する場合は、このことを念頭に置いてください。
DATETIMEOFFSET	TIMESTAMP WITH TIME ZONE	
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
VARBINARY(*n*)	VARBINARY(*n*)	
VARBINARY(MAX)	LONG BINARY	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(MAX)	LONG VARCHAR	
XML	XML	

MySQL データのマッピング

MySQL の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように MySQL の統合データ型にマッピングされるのかを示します。たとえば、リモートデータベースの TEXT 型のカラムは、統合データベースでは LONGTEXT 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	MySQL のデータ型	説明
BIGINT	BIGINT	
BINARY(*n*<=255)	BINARY(*n*)	
BINARY(*n*>255)	BLOB	
BIT	BIT	
CHAR(*n*<=255)	CHAR(*n*)	
CHAR(*n*>255)	TEXT(*n*)	
DATE	DATE	年は 1000 ～ 9999 である必要があります。
DATETIME	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。
DECIMAL(*p*<=65,*s*<=30)	DECIMAL(*p*,*s*)	
DECIMAL(*p*>65,*s*>30)		精度が 65 より大きい場合、または位取りが 30 より大きい場合、MySQL には対応するデータ型がありません。
DOUBLE	DOUBLE	
FLOAT	FLOAT	
IMAGE	LONGBLOB	
INTEGER	INTEGER	
LONG BINARY	LONGBLOB	
LONG NVARCHAR	LONGTEXT CHARACTER SET UTF8	
LONG VARBIT	LONGTEXT	
LONG VARCHAR	LONGTEXT	
MONEY	NUMERIC(19,4)	

SQL Anywhere または Ultra Light のデータ型	MySQL のデータ型	説明
NCHAR(*n*<=255)	CHAR(*n*) CHARACTER SET UTF8	
NCHAR(*n*>255)	TEXT CHARACTER SET UTF8	
NTEXT	LONGTEXT CHARACTER SET UTF8	
NUMERIC(*p*<=65,*s*<=30)	DECIMAL(*p*,*s*)	
NUMERIC(*p*>65,*s*>30)		MySQL には対応するデータ型がありません。
NVARCHAR(*n*)	VARCHAR(*n*) CHARACTER SET UTF8	
REAL	REAL	
SMALLDATETIME	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	NUMERIC(10,4)	
ST_GEOMETRY	GEOMETRY	
TEXT	LONGTEXT	
TIME	TIME	MySQL の TIME データ型は、秒の小数点以下をサポートしていません。
TIMESTAMP	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。

SQL Anywhere または Ultra Light のデータ型	MySQL のデータ型	説明
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	MySQL には対応するデータ型がありません。したがって、TIMESTAMP WITH TIME ZONE カラムを VARCHAR(34) カラムにマッピングする必要があります。アップロード時、Mobile Link サーバは、まずデータを YYYY-MM-DD HH:NN:SS.SSSSSS [+|-]HH:NN フォーマットの文字列に変換してから統合データベースに適用します。ダウンロード時は、データを文字列から TIMESTAMP WITH TIME ZONE に変換します。統合データベース内のデータがこのフォーマットに従っていない場合は、ダウンロードが失敗します。
TINYINT	TINYINT UNSIGNED	TINYINT は、SQL Anywhere および Ultra Light では常に符号なしです。
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	
VARBINARY(*n*)	VARCHAR(*n*)	
VARBIT(*n*<=8000)	VARCHAR(*n*)	
VARBIT(*n*>8000)	TEXT	
VARCHAR(*n*)	VARCHAR(*n*)	
XML	LONGTEXT	

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、MySQL の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるのかを示します。たとえば、統合データベースの BOOL 型のカラムは、リモートデータベースでは BIT 型である必要があります。

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
BIT(1)	BIT	
BIT(*n*>1)	UNSIGNED BIGINT	
BLOB(*n*<=32767)	VARBINARY(*n*)	
BLOB(*n*>32767)	IMAGE	
BOOL	BIT	
CHAR(*n*)	CHAR(*n*)	
DATE	DATE	年は 1000 ～ 9999 である必要があります。
DATETIME	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。
DOUBLE	DOUBLE	
DECIMAL	DECIMAL	
ENUM		SQL Anywhere または Ultra Light には対応するデータ型がありません。
GEOMETRY	ST_GEOMETRY	
INTEGER	INTEGER	
LINESTRING		SQL Anywhere または Ultra Light には対応するデータ型がありません。
LONGBLOB	IMAGE	
LONGTEXT	TEXT	
MEDIUMBLOB	IMAGE	
MEDIUMINT	INTEGER	
MEDIUMTEXT	TEXT	
MULTILINESTRING		SQL Anywhere または Ultra Light には対応するデータ型がありません。

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
MULTIPOINT		SQL Anywhere または Ultra Light には対応するデータ型がありません。
MULTIPOLYGON		SQL Anywhere または Ultra Light には対応するデータ型がありません。
NCHAR	NCHAR	Ultra Light では使用できません。
NUMERIC	NUMERIC	
NVARCHAR	NVARCHAR	Ultra Light では使用できません。
POINT		SQL Anywhere または Ultra Light には対応するデータ型がありません。
POLYGON		SQL Anywhere または Ultra Light には対応するデータ型がありません。
REAL	REAL	
SET		SQL Anywhere または Ultra Light には対応するデータ型がありません。
SMALLINT	SMALLINT	
TEXT(*n*<=32767)	VARCHAR(*n*)	
TEXT(*n*>32767)	TEXT	
TIME	TIME	MySQL の TIME データ型は、秒の小数点以下をサポートしていません。MySQL の TIME の範囲は '-838:59:59' ～ '838:59:59' です。SQL Anywhere および Ultra Light の TIME の範囲は '00:00:00.000000' ～ '23:59:59:999999' です。
TIMESTAMP	TIMESTAMP	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。MySQL は TIMESTAMP カラムでの自動初期化と更新機能を提供していますが、SQL Anywhere および Ultra Light は自動初期化のみを提供しています。

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	説明
TINYBLOB	VARBINARY	
TINYINT	SMALLINT	TINYINT は、SQL Anywhere および Ultra Light では常に符号なしです。正の値にしてください。
TINYINT UNSIGNED	TINYINT	TINYINT は、SQL Anywhere および Ultra Light では常に符号なしです。
TINYTEXT	VARCHAR	
VARBINARY(*n*<=32767)	VARBINARY(*n*)	
VARBINARY(*n*>32767)	IMAGE	
VARCHAR(*n*<=32767)	VARCHAR(*n*)	
VARCHAR(*n*>32767)	TEXT	
YEAR[(2|4)]	INTEGER	SQL Anywhere および Ultra Light は YEAR データ型をサポートしていません。YEAR は、リモートデータベースの INTEGER にマッピングしてください。INTEGER 値は 1000 ～ 9999 である必要があります。

Oracle データのマッピング

Oracle の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように Oracle の統合データ型にマッピングされるのかを示します。たとえば、リモートデータベースの BIT 型のカラムは、統合データベースでは NUMBER 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
BIGINT	NUMBER(20)	
BINARY(*n*<=2000)	RAW(*n*)	

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
BINARY(*n*>2000)	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
BIT	NUMBER(1)	
CHAR(*n*<=4000)	VARCHAR2(*n* byte)	Oracle の VARCHAR2 を使用すると、バイト数または文字数の最大値を指定できます。VARCHAR2 データの最大長は 4000 バイトです。文字数を指定する場合は、データの最大長が 4000 バイトを超えないようにしてください。
CHAR(*n*>4000)	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
DATE	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。
DATETIME	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
DECIMAL(*p*<=38,*s*)	NUMBER(*p*, 0<=*s*<=38)	SQL Anywhere の DECIMAL では、p は 1 ～ 127 で、s は常に p 以下です。Oracle の NUMBER では、p は 1 ～ 38 で、s は -84 ～ 127 です。同期させるには、Oracle の NUMBER の位取りを 0 ～ 38 に制限する必要があります。
DECIMAL(*p*>38,*s*)		Oracle には対応するデータ型がありません。
DOUBLE	DOUBLE PRECISION または BINARY_DOUBLE[1]	Oracle Database 10g の BINARY_FLOAT と BINARY_DOUBLE の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
INTEGER	INT	
LONG BINARY	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
LONG NVARCHAR	NCLOB	Oracle の CLOB と NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の LONG VARCHAR と LONG NVARCHAR が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
LONG VARBIT	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
LONG VARCHAR	CLOB	Oracle の CLOB と NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の LONG VARCHAR と LONG NVARCHAR が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
MONEY	NUMBER(19,4)	
NCHAR(*c*)	NVARCHAR2(*c* char) または NCLOB	SQL Anywhere の NCHAR と Oracle の NVARCHAR2 では、サイズは Unicode 文字の最大文字数を示します。Oracle の NVARCHAR2 のデータ長が 4000 バイトを超えることはできません。文字サイズから最大バイト長を計算することは困難です。一般的に、サイズが 1000 を超える場合は NCLOB に、そうでない場合は NVARCHAR2 に、それぞれマッピングします。
NTEXT	NCLOB	Oracle の NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の NTEXT (または LONG NVARCHAR) が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
NUMERIC(*p*<=38,*s*)	NUMBER(*p*, 0<=*s*<=38)	SQL Anywhere の NUMERIC では、*p* は 1 ～ 127 であり、*s* は常に *p* 以下です。Oracle の NUMBER では、*p* は 1 ～ 38 であり、*s* は -84 ～ 127 です。同期させるには、Oracle の NUMBER の位取りを 0 ～ 38 に制限する必要があります。
NUMERIC(*p*>38,*s*)		Oracle には対応するデータ型がありません。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
NVARCHAR	NVARCHAR2(*c* char) または NCLOB	SQL Anywhere の NCHAR と Oracle の NVARCHAR2 では、サイズは Unicode 文字の最大文字数を示します。Oracle の NVARCHAR2 のデータ長が 4000 バイトを超えることはできません。文字サイズから最大バイト長を計算することは困難です。一般的に、サイズが 1000 を超える場合は NCLOB に、そうでない場合は NVARCHAR2 に、それぞれマッピングします。
REAL	REAL または BINARY_FLOAT[1]	Oracle Database 10g の BINARY_FLOAT と BINARY_DOUBLE の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。
SMALLDATETIME	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。
SMALLINT	NUMBER(5)	
SMALLMONEY	NUMBER(10,4)	
ST_GEOMETRY	SDO_GEOMETRY	
TEXT	CLOB	Oracle の CLOB は、最大で 4G のデータを保持できます。SQL Anywhere の TEXT (または LONG VARCHAR) が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
TIME	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。
TIMESTAMP	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	
TINYINT	NUMBER(3)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED BIGINT	NUMBER(20)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED INTEGER	NUMBER(11)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED SMALLINT	NUMBER(5)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED TINYINT	NUMBER(3)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNIQUEIDENTIFIER	CHAR(36)	

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	説明
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR の Oracle での使用はおすすめしません。代わりに UNIQUEIDENTIFIER を使用してください。
VARBINARY(*n*<=2000)	RAW(*n*)	
VARBINARY(*n*>2000)	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
VARBIT(*n*<=4000)	VARCHAR2(*n* byte)	
VARBIT(*n*>4000)	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
VARCHAR(*n*<=4000)	VARCHAR2(*n* byte)	Oracle の VARCHAR2 を使用すると、バイト数または文字数の最大値を指定できます。VARCHAR2 データの最大長は 4000 バイトです。文字数を指定する場合は、データの最大長が 4000 バイトを超えないようにしてください。
VARCHAR(*n*>4000)	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
XML	XMLTYPE	Oracle データベースサーバでは、XMLTYPE データの構文はチェックされますが、SQL Anywhere の構文はチェックされません。 Ultra Light では使用できません。 Oracle の XMLTYPE データ型 22 ページを参照してください。

[1] Oracle Database 10g 以降にのみ該当します。

[2] Oracle バージョン 8i 以降にのみ該当します。

注意
LONG データ型は、Oracle 8、8i、9i では使用されなくなりました。

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、Oracle の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるのかを示します。たとえば、統合データベースの LONG 型のカラムは、リモートデータベースでは LONG VARCHAR 型である必要があります。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	説明
BFILE	LONG BINARY	ダウンロードのみ。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
BINARY_DOUBLE	DOUBLE	BINARY_FLOAT の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。Oracle の FLOAT と DOUBLE の精度は、SQL Anywhere と Ultra Light のものとは異なります。精度によっては、データの値が変わる可能性があります。
BINARY_FLOAT	REAL	BINARY_FLOAT の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。Oracle の FLOAT と DOUBLE の精度は、SQL Anywhere と Ultra Light のものとは異なります。精度によっては、データの値が変わる可能性があります。
BLOB	LONG BINARY	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	説明
CHAR(*n* byte)	VARCHAR(*n*)	SQL Anywhere の CHAR は、Oracle の CHAR と同等ではありません。SQL Anywhere の CHAR に対応するのは VARCHAR です。同期される統合データベースのカラムでは、CHAR/NCHAR を使用しないでください。SQL Anywhere 以外の CHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。 SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
CLOB	LONG VARCHAR	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
DATE	TIMESTAMP	年は、1 ～ 9999 の範囲内である必要があります。
INTERVAL YEAR(*year_precision*) TO MONTH		SQL Anywhere または Ultra Light には対応するデータ型がありません。
INTERVAL DAY(*day_precision*) TO SECOND(*p*)		SQL Anywhere または Ultra Light には対応するデータ型がありません。
LONG	LONG VARCHAR	
LONG RAW	LONG BINARY	

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	説明
NCHAR(*c* char)	NVARCHAR(*c*)	SQL Anywhere の NCHAR は、Oracle の NCHAR と同等ではありません。SQL Anywhere の NCHAR に対応するのは NVARCHAR です。同期される統合データベースのカラムでは、NCHAR を使用しないでください。SQL Anywhere 以外の NCHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。 SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
NCLOB	LONG NVARCHAR	Ultra Light では使用できません。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
NUMBER(*p*,*s*)	NUMBER(*p*,*s*)	SQL Anywhere の NUMBER では、*p* は 1 ～ 127 であり、*s* は常に *p* 以下です。Oracle の NUMBER では、*p* は 1 ～ 38 であり、*s* は -84 ～ 127 です。同期させるには、Oracle の NUMBER の位取りを 0 ～ 38 にしてください。
NVARCHAR2(*c* char)	NVARCHAR(*c*)	Ultra Light では使用できません。 SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
RAW	BINARY	SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
ROWID	VARCHAR(64)	UROWID と ROWID は読み込み専用なので、同期対象になることはほとんどありません。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	説明
SDO_GEOMETRY	ST_GEOMETRY	
TIMESTAMP(*p*<=6)	TIMESTAMP	*p*<6 の場合、SQL Anywhere または Ultra Light の値が同じ精度になっていることを確認する必要があります。そうしないと、競合を検出できなかったり、ローの重複が発生したりする可能性が生じます。年は、1 ～ 9999 の範囲内である必要があります。
TIMESTAMP(*p*>6)		SQL Anywhere または Ultra Light には対応するデータ型がありません。
TIMESTAMP(*p*) WITH LOCAL TIME ZONE		SQL Anywhere または Ultra Light には対応するデータ型がありません。
TIMESTAMP(*p*<=6) WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	
TIMESTAMP(*p*) WITH TIME ZONE		SQL Anywhere または Ultra Light には対応するデータ型がありません。
UROWID	VARCHAR(64)	UROWID と ROWID は読み込み専用なので、同期対象になることはほとんどありません。
VARCHAR2(*n* byte)	VARCHAR(*n*)	SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
XMLTYPE	XML、LONG VARCHAR、または VARCHAR(*n*)	Oracle データベースサーバでは、XMLTYPE データの構文がチェックされます。SQL Anywhere では、XML データの内容がチェックされず、XML データは VARCHAR データとして処理されます。 Oracle の XMLTYPE データ型 22 ページを参照してください。

SAP HANA データベースサーバのデータ型マッピング

SAP HANA の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように SAP HANA の統合データ型にマッピングされるかを示します。たとえば、リモートデータベースの LONG VARBIT 型のカラムは、統合データベースでは CLOB 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	SAP HANA のデータ型	注意
BIT	TINYINT	
TINYINT	TINYINT	
SMALLINT	SMALLINT	
UNSIGNED SMALLINT	INTEGER	
INTEGER	INTEGER	
UNSIGNED INTEGER	BIGINT	
BIGINT	BIGINT	
UNSIGNED BIGINT	DECIMAL(20,0)	
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
NUMERIC(*p*,*s*)	DECIMAL(*p*,*s*)	
FLOAT	FLOAT	SAP HANA からデータをダウンロードするテーブルがあるリモートデータベースでは、FLOAT データ型の使用を避けてください。
REAL	REAL	SAP HANA からデータをダウンロードするテーブルがあるリモートデータベースでは、REAL データ型の使用を避けてください。
DOUBLE	DOUBLE	
SMALLMONEY	DECIMAL(10,4)	
MONEY	DECIMAL(19,4)	
DATE	DATE	

SQL Anywhere または Ultra Light のデータ型	SAP HANA のデータ型	注意
TIME	TIME	SAP HANA の TIME データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere および Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。
SMALLDATETIME	TIMESTAMP	
DATETIME	TIMESTAMP	
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	SAP HANA に対応するデータ型がありません。したがって、TIMESTAMP WITH TIME ZONE カラムを VARCHAR(34) カラムにマッピングする必要があります。アップロード時、Mobile Link サーバは、データを 'yyyy-mm-dd hh:nn:ss.ssssss [+|-]hh:nn' フォーマットの文字列に変換してから統合データベースに適用します。ダウンロード時は、データを文字列から TIMESTAMP WITH TIME ZONE に変換します。エラーおよび同期の失敗を回避するには、統合データベース内のデータがこのフォーマットに従っていることを確認します。
CHAR(*n*<=5000)	VARCHAR(*n*)	
CHAR(*n*>5000)	CLOB	
VARCHAR(*n*<=5000)	VARCHAR(*n*)	
VARCHAR(*n*>5000)	CLOB	
LONG VARCHAR	CLOB	
NCHAR(*n*<=5000)	NVARCHAR	
NCHAR(*n*>5000)	NCLOB	
NVARCHAR(*n*<=5000)	NVARCHAR(*n*)	
NVARCHAR(*n*>5000)	NCLOB	

SQL Anywhere または Ultra Light のデータ型	SAP HANA のデータ型	注意
LONG NVARCHAR	NCLOB	
BINARY(*n*<=5000)	VARBINARY(*n*)	
BINARY(*n*>5000)	BLOB	
VARBINARY(*n*<=5000)	VARBINARY(*n*)	
VARBINARY(*n*>5000)	BLOB	
LONG BINARY	BLOB	
VARBIT(*n*<=5000)	VARCHAR(*n*)	
VARBIT(*n*>5000)	CLOB	
LONG VARBIT	CLOB	
GUID	VARCHAR(40)	
GEOMETRY		サポートされていません。

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、SAP HANA の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるかを示します。たとえば、統合データベースの ALPHANUM(*n*) 型のカラムは、リモートデータベースでは VARCHAR(*n*) 型である必要があります。

SAP HANA のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
TINYINT	TINYINT	
SMALLINT	SMALLINT	
INTEGER	INTEGER	
BIGINT	BIGINT	
SMALLDECIMAL	DECIMAL(*p*,*s*)	SQL Anywhere の DECIMAL では、*p* は 1 ～ 127 であり、*s* は常に *p* 以下です。SAP HANA では、*p* は 1 ～ 16 であり、*s* は -369 ～ 368 です。

SAP HANA のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	SQL Anywhere の DECIMAL では、*p* は 1 ～ 127 であり、*s* は常に *p* 以下です。SAP HANA では、*p* は 1 ～ 34 であり、*s* は -6111 ～ 6176 です。
FLOAT	DOUBLE	SAP HANA では、FLOAT および REAL が DOUBLE に昇格されます。
REAL	DOUBLE	SAP HANA では、FLOAT および REAL が DOUBLE に昇格されます。
DOUBLE	DOUBLE	
DATE	DATE	
TIME	TIME	SAP HANA の TIME データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere および Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。
SECONDDATE	TIMESTAMP	SAP HANA の SECONDDATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere および Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。
TIMESTAMP	TIMESTAMP	
VARCHAR(*n*)	VARCHAR(*n*)	
NVARCHAR(*n*)	NVARCHAR(*n*)	
ALPHANUM(*n*)	VARCHAR(*n*)	
VARBINARY(*n*)	VARBINARY(*n*)	
CLOB	LONG VARCHAR	
NCLOB	LONG NVARCHAR	
BLOB	LONG BINARY	

SAP Sybase IQ Enterprise のデータマッピング

SAP Sybase IQ の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモートデータ型がどのように SAP Sybase IQ の統合データ型にマッピングされるかを示します。たとえば、リモートデータベースの LONG VARBIT 型のカラムは、統合データベースでは LONG VARCHAR 型にする必要があります。

SQL Anywhere または Ultra Light のデータ型	SAP Sybase IQ	注意
BIGINT	BIGINT	
BIT	BIT	
BINARY(*n*)	BINARY(*n*)	
CHAR(*n*)	CHAR(*n*)	255 バイトを超える CHAR および VARCHAR カラムにはいくつかの制限があります。詳細については、SAP Sybase IQ のマニュアルを参照してください。
DATE	DATE	
DATETIME	DATETIME	
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
DOUBLE	DOUBLE	
FLOAT(*p*)	FLOAT(*p*)	
INT	INT	
LONG BINARY / IMAGE	LONG BINARY / IMAGE	
LONG NVARCHAR / NTEXT		このデータ型は SAP Sybase IQ では利用できません。
LONG VARBIT	LONG VARCHAR	
LONG VARCHAR / TEXT	TEXT	
MONEY	MONEY	
NCHAR(*n*)		このデータ型は SAP Sybase IQ では利用できません。

SQL Anywhere または Ultra Light のデータ型	SAP Sybase IQ	注意
NVARCHAR(*n*)		このデータ型は SAP Sybase IQ では利用できません。
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
SMALLDATETIME	SMALLDATETIME	
SMALLMONEY	SMALLMONEY	
ST_GEOMETRY		このデータ型は SAP Sybase IQ では利用できません。
TIME	TIME	
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INT	UNSIGNED INT	
UNSIGNED SMALLINT	SMALLINT	
UNSIGNED TINYINT	TINYINT	
VARBINARY(*n*)	VARBINARY(*n*)	
VARBIT(*n*)	VARCHAR(*n*)	
VARCHAR(*n*)	VARCHAR(*n*)	255 バイトを超える CHAR および VARCHAR カラムにはいくつかの制限があります。詳細については、SAP Sybase IQ のマニュアルを参照してください。
XML	LONG BINARY / IMAGE	

SQL Anywhere または Ultra Light のリモートデータ型へのマッピング

次の表は、SAP Sybase IQ の統合データ型がどのように SQL Anywhere および Ultra Light のリモートデータ型にマッピングされるかを示します。たとえば、統合データベースの DOUBLE PRECISION 型のカラムは、リモートデータベースでは DOUBLE 型にする必要があります。

SAP Sybase IQ	SQL Anywhere または Ultra Light のデータ型	注意
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	
CHAR(*n*)	VARCHAR(*n*)	
DATE	DATE	
DATETIME	DATETIME	
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
DOUBLE	DOUBLE	
FLOAT(*p*)	FLOAT(*p*)	
INT	INT	
LONG BINARY / IMAGE	LONG BINARY / IMAGE	
LONG VARCHAR / TEXT	LONG VARCHAR / TEXT	
MONEY	MONEY	
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TIME	TIME	

SAP Sybase IQ	SQL Anywhere または Ultra Light のデータ型	注意
TIMESTAMP	TIMESTAMP	
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INT	UNSIGNED INT	
VARBINARY(*n*)	VARBINARY(*n*)	
VARCHAR(*n*)	VARCHAR(*n*)	

文字セットの考慮事項

テキストの各文字は、1 バイトまたはそれ以上のバイトで表現されます。文字からバイナリコードへのマッピングを「文字セットエンコード」といいます。ヨーロッパ言語など、小規模なアルファベット体系を持つ言語で使われる文字セットには、シングルバイト表現が使用されるものがあります。Unicode などのように 2 バイト表現を使用する文字セットもあります。2 バイト文字セットは、各文字に対して記憶領域を 2 倍使用するため、はるかに多くの文字を表現できます。

変換エラーが起きたり、データが失われたりするのは、1 つの文字セットを使用しているテキストを別の文字セットに変換しなければならないときです。すべての文字がすべての文字セットで表示できるわけではありません。特に、シングルバイト文字セットは、使用可能なコードの数が限られているので、表示できる文字はマルチバイトシステムより少なくなります。

Mobile Link リモートデータベースの文字セットがユーザの統合データベースと同じ場合は、文字変換の問題は起こりません。

ディレクトリのリストのようにインデックスを構築したり順序リストを準備したりするには、テキストをソートする必要がよくあります。「ソート順」は、文字の順序を特定します。たとえば、一般的にソート順では文字 "a" は文字 "b" の前に位置し、文字 "b" は文字 "c" の前に位置します。

各データベースには、「照合順」があります。照合順は、データベースを作成するときに設定します。設定方法は、データベースのシステムによって異なります。照合順は、データベースの文字セットとソート順の両方を定義します。

注意
可能な場合は常に、使用しているリモートデータベースの照合順を、使用している統合データベースの照合順と同じになるように定義してください。この方法により、間違った変換をする可能性が減ります。

参照

- SQL Anywhere クライアント： 「国際言語と文字セット」『SQL Anywhere サーバ データベース管理』
- Ultra Light クライアント： 「Ultra Light 文字セット」『Ultra Light データベース管理とリファレンス』
- RDBMS 固有の情報： 「Mobile Link 統合データベース」1 ページ

同期中の文字セット変換

同期中は、ある文字セットの文字を別の文字セットに変換しなければならないことがあります。次の変換は、文字がリモートアプリケーションと統合データベースの間で渡されるときに起こります。

アップロード中の文字セット変換

Mobile Link クライアントは、リモートデータベースの文字セットを使用して Mobile Link サーバにデータを送ります。

1. Mobile Link サーバは、Unicode ODBC API を使用して統合データベースと通信します。通信するために、Mobile Link サーバは、リモートデータベースから受信したすべての文字を Unicode に変換し、Unicode を ODBC ドライバに送信します。

2. 必要に応じて、統合データベースサーバの ODBC ドライバは文字を Unicode から統合データベースの文字セットに変換します。この変換を制御できるのは、使用している統合データベースシステムの ODBC ドライバだけです。したがって、2 つの異なるデータベースシステム (特に、異なるメーカーによって作成されたシステム) では、動作が異なる場合があります。Mobile Link 同期は、複数のデータベースシステムで動作可能です。詳細については、使用している統合サーバと ODBC ドライバのマニュアルを参照してください。

ダウンロード中の文字セット変換

1. 統合データベースシステムの ODBC ドライバは、文字を統合データベースのコードで受け取ります。このドライバは、これらの文字を Unicode に変換し、Unicode API をとおして Mobile Link サーバに渡します。この変換を制御できるのは、使用している統合データベースシステムの ODBC ドライバだけです。詳細については、使用している統合サーバと ODBC ドライバのマニュアルを参照してください。

2. Mobile Link サーバは、Unicode ODBC API をとおして文字を受け取ります。リモートデータベースで異なる文字セットを使用している場合は、ダウンロードする前に Mobile Link サーバが文字を変換します。

例

- Windows Mobile デバイス上の Ultra Light アプリケーションでは、Unicode 文字セットを使用します。

  Windows Mobile アプリケーションを同期するときは、Mobile Link サーバ内で文字変換は行われません。Mobile Link 同期サーバは、アプリケーションから送信されたデータがすでに Unicode であると判断し、それを ODBC ドライバに直接渡します。同様に、データのダウンロード時も文字セットの変換は必要ありません。

- Windows Mobile 以外のプラットフォーム上にあるすべての SQL Anywhere データベースとすべての Ultra Light アプリケーションは、リモートデータベースの照合順によって決定された文字セットを使用します。

  リモートデータベースを同期するときは、Mobile Link サーバがリモートデータベースの文字セットと Unicode の間の文字セット変換を実行します。

ODBC ドライバの文字セットの変換

統合データベースは Unicode を使用していない場合が多いので、使用している統合データベースシステムの ODBC ドライバがデータを Unicode に変換する方法と Unicode から変換する方法について理解しておくことが重要です。ODBC ドライバの中には、Mobile Link を実行しているコンピュータの言語設定を使って、使用する文字セットを決定するものがあります。この場合、Mobile Link サーバを実行しているコンピュータの言語設定とコードページ設定が、統合データベースの設定と一致するのが最も望ましいことです。

Sybase Adaptive Server Enterprise のドライバなどその他の ODBC ドライバでは、各接続で特定の文字セットを使用できます。変換のエラーを避けるために、Mobile Link で使用する文字セットが統合データベースの文字セットと一致するように設定してください。

使用している統合データベースサーバの ODBC ドライバで文字セット変換がどのように行われるかについては、各製品の ODBC ドライバのマニュアルを参照してください。

Mobile Link の ODBC ドライバ

Mobile Link でサポートされる ODBC ドライバ

次の表のとおり、Mobile Link サーバは各種の統合データベースや ODBC ドライバと連携させることができます。ドライバによっては、Mobile Link と互換性があっても、使用に関する機能上の制約がある場合があります。

サポートされるバージョンの詳細については、http://www.sybase.com/detail?id=1002288 を参照してください。

データベース	ODBC ドライバ
SQL Anywhere 16	SQL Anywhere 16[1]
Oracle Database 10g または Oracle Database 11g	SQL Anywhere 16 - Oracle[1]
Microsoft SQL Server	Microsoft SQL Server ODBC ドライバ [2]
Sybase Adaptive Server Enterprise 15.0 以降	Sybase Adaptive Server Enterprise ドライバ [2]

データベース	ODBC ドライバ
Windows、Linux、UNIX 用の IBM DB2 LUW 9.5	IBM DB2 9.5 CLI ドライバ [2]
Windows、Linux、UNIX 用の IBM DB2 LUW 9.7	IBM DB2 9.7 CLI ドライバ [2]
MySQL 5.1	MySQL ODBC ドライバ 5.1[2]

[1] SQL Anywhere バージョン 16 で提供されています。http://www.ianywhere.jp/tech/odbc_mobilink.html を参照してください。

[2] SQL Anywhere バージョン 16 では提供されていません。インストール方法と設定方法については、http://www.ianywhere.jp/tech/odbc_mobilink.html を参照してください。

SQL Anywhere 16 - Oracle ODBC ドライバ

SQL Anywhere 16 - Oracle ODBC ドライバは、iAnywhere ソフトウェア用にカスタマイズされています。このドライバは、サードパーティソフトウェアでは動作しません。

Mobile Link またはリモートデータアクセスで Oracle を使用する場合、この Oracle ドライバと同じコンピュータに Oracle クライアントをインストールする必要があります。

Oracle ドライバは、ODBC アドミニストレータ、*.odbc.ini* ファイル (UNIX)、または dbdsn ユーティリティを使用して設定できます。

次の表に、Oracle ドライバの設定オプションを示します。

Windows ODBC データソースアドミニストレータ	dbdsn コマンドラインまたは *.odbc.ini* ファイルの設定	説明
データソース名	dbdsn では、-w オプションを使用します。	データソースを識別するための名前。
ユーザ ID	**UserID** dbdsn では、接続文字列でこのオプションを設定します。	アプリケーションが Oracle データベースへの接続に使用するデフォルトのログオン ID。このフィールドを空白にすると、接続したときに情報が要求されます。

Windows ODBC データソースアドミニストレータ	dbdsn コマンドラインまたは *.odbc.ini* ファイルの設定	説明
パスワード	**Password** dbdsn では、接続文字列でこのオプションを設定します。	アプリケーションが Oracle データベースへの接続に使用するパスワード。このフィールドを空白にすると、接続したときに情報が要求されます。
TNS サービス名	**ServiceName**	Oracle インストールディレクトリの *network/admin/tnsnames.ora* に保存される TNS サービス名。
パスワードの暗号化	dbdsn では、-pe オプションを使用します。 *.odbc.ini* では、サポートされていません。	パスワードを暗号化形式でデータソースに保存するには、このチェックボックスをオンにします。
プロシージャから結果が返される、または VARRAY パラメータを使用する	**ProcResults** dbdsn では、接続文字列でこのオプションを設定します。 **ProcOwner** dbdsn では、接続文字列でこのオプションを設定します。	ストアドプロシージャで結果を返すことができる場合、またはストアドプロシージャで Oracle VARRAY を使用している場合は、このフィールドを選択します。デフォルトでは、このオプションは選択されていません。download_cursor スクリプトまたは download_delete_cursor スクリプトがストアドプロシージャ呼び出しの場合は、このチェックボックスをオンにしてください。 ストアドプロシージャで VARRAY を使用しておらず、結果セットも返されない場合は、パフォーマンス向上のためにこのチェックボックスをクリアします。

Windows ODBC データソースアドミニストレータ	dbdsn コマンドラインまたは *.odbc.ini* ファイルの設定	説明
配列サイズ	**ArraySize** dbdsn では、接続文字列でこのオプションを設定します。	文ごとのローのプリフェッチに使用するバイト配列のサイズ (バイト単位)。デフォルトは 60000 です。この値を大きくすると、使用するメモリが増えますが、フェッチのパフォーマンス (Mobile Link サーバのダウンロードなど) は大幅に向上します。
Microsoft 分散トランザクションを有効にする	dbdsn では、接続文字列で **enableMSDIC** オプションを使用します。 *.odbc.ini* では、サポートされていません。	トランザクションを Microsoft 分散トランザクションコーディネータにエンリストする場合は、このチェックボックスをオンにします。選択すると、Oracle ODBC ドライバに Oracle バイナリファイル *oramts10.dll* (Oracle データベース 10g クライアントの場合) または *oramts11.dll* (Oracle データベース 11g クライアントの場合) が必要になります。

Windows での Oracle ドライバ用の ODBC データソースの作成

Windows で Oracle ドライバ用の ODBC データソースを作成するには、次の手順に従います。

前提条件

この作業を実行するための前提条件はありません。

♦ Windows での Oracle ドライバ用の ODBC データソースの作成

1. ODBC アドミニストレータを開きます。
   - **[スタート]** » **[プログラム]** » **[SQL Anywhere 16]** » **[管理ツール]** » **[ODBC データソースアドミニストレータを開く]** をクリックします。

   [ODBC データソースアドミニストレータ] が表示されます。

2. **[追加]** をクリックします。

3. **[SQL Anywhere 16 - Oracle]** を選択し、**[完了]** をクリックします。

4. 必要な設定オプションを指定します。フィールドは上記の説明のとおりです。

5. **[接続テスト]** をクリックし、**[OK]** をクリックします。

結果

Oracle ドライバの ODBC データソースが作成されます。

次の手順

ODBC データソースを使用して、接続します。

UNIX での設定

UNIX では、ODBC システム情報ファイル (通常、*.odbc.ini*) でドライバを設定している場合、このドライバのセクションは次のように表示されます (各フィールドに適切な値が入力されます)。

```
[sample_dsn_using_the_ias_odbc_driver_for_oracle]
Driver=full-path/libdboraodbc12_r.so
UserID=user-id
Password=password
ServiceName=TNS-service-name
ProcResults=[yes|no]
ArraySize=bytes
```

各フィールドについては、「SQL Anywhere 16 - Oracle ODBC ドライバ」761 ページ の表を参照してください。

DBDSN 設定

Oracle DSN を dbdsn ユーティリティを使用して作成するには、次の構文を使用します。

dbdsn -w *data-source-name* **-or -c** *configuration-options*

configuration-options については、「SQL Anywhere 16 - Oracle ODBC ドライバ」761 ページ の説明を参照してください。

次に例を示します。

```
dbdsn -w MyOracleDSN -or -pe -c
"Userid=dba;Password=sql;ServiceName=abcd;ArraySize=100000;ProcResults=y;enableMSDIC=n"
```

「データソースユーティリティ (dbdsn)」『SQL Anywhere サーバ データベース管理』を参照してください。

参照

- Mobile Link の推奨 ODBC ドライバ： http://www.sybase.com/detail?id=1011880
- 「データソースユーティリティ (dbdsn)」『SQL Anywhere サーバ データベース管理』

Mobile Link アプリケーションの配備

Mobile Link アプリケーションを配備するときは、以下の作業を行います。

- 運用設定への Mobile Link サーバの配備
- SQL Anywhere Mobile Link クライアントの配備
- Ultra Light Mobile Link クライアントの配備

この項では、アプリケーションのインストールプログラムに含める必要があるファイルについて、これらの項目ごとに説明します。

Windows での配備には、**同期モデル展開ウィザード**を使用できます。 「Deployment ウィザード」『SQL Anywhere サーバ プログラミング』を参照してください。

注意

ライセンス契約の確認　ファイルの再配布はライセンス契約に従います。このマニュアル内の記述は、ライセンス契約のどの条項にも優先しません。配備について検討する前にライセンス契約を確認してください。

Mobile Link サーバの配備

Mobile Link サーバを運用環境に配備するには、SQL Anywhere のライセンス取得済みコピーを運用コンピュータにインストールするのが最も簡単です。

しかし、Mobile Link サーバを別のインストールプログラムで再配布する場合は、ファイルのサブセットだけを含めてもかまいません。この場合、インストール環境に次のファイルを含める必要があります。

注意

- 再配布する前にクリーンなコンピュータでテストしてください。
- サンプル以外のファイルは、SQL Anywhere インストールディレクトリにインストールしてください。
- 特に指定がないかぎり、ファイルを同じディレクトリにインストールしてください。
- ロケーションが指定されている場合、ファイルは同じ名前のディレクトリにコピーしてください。
- UNIX の場合は、システムが SQL Anywhere アプリケーションとライブラリを見つけることができるように環境変数を設定してください。必要な環境変数を設定するためのテンプレートとして、*sa_config.sh* と *sa_config.csh* (32 ビットの環境では *$SQLANY16/bin32* ディレクトリ内、64 ビットの環境では *$SQLANY16/bin64* 内) のいずれかのうち、シェルに適したファイルを使用することをおすすめします。sa_config ファイルによって設定される環境変数には PATH、LD_LIBRARY_PATH、SQLANY16、SQLANYSAMP16 などがあります。

- Windows の場合は、システムが SQL Anywhere アプリケーションとライブラリを見つけることができるように環境変数を設定してください。32 ビットの環境の場合は *%SQLANY16%¥Bin32*、64 ビットの環境の場合は *%SQLANY16%¥Bin64* が、PATH 変数に含まれていることを確認します。両方のエントリが存在する場合は、環境に該当しないパスを削除します。

- Java 同期論理を使用するには、JRE 1.6.0 以降がインストールされている必要があります。グラフィカルな管理ツール (Sybase Central と Mobile Link プロファイラ) を使用するには、JRE 1.7.0 以降がインストールされている必要があります。

- Sybase Central を配備する方法については、「管理ツールの配備」『SQL Anywhere サーバ プログラミング』を参照してください。

- Windows 用の配備ウィザードがあります。 「Deployment ウィザード」『SQL Anywhere サーバ プログラミング』を参照してください。

Windows の 32 ビットアプリケーション

すべてのディレクトリは、*%SQLANY16%* を基準とした相対ディレクトリです。 64 ビット Windows 環境のファイル構造の詳細については、「Windows の 64 ビットアプリケーション」768 ページを参照してください。

説明	Windows ファイル
Mobile Link サーバ	● *Bin32¥mlodbc16.dll* ● *Bin32¥mlsrv16.exe* ● *Bin32¥mlsrv16.lic* ● *Bin32¥mlsql16.dll* ● *Bin32¥dbicu16.dll* ● *Bin32¥dbicudt16.dll*
言語ライブラリ	● *Bin32¥dblgen16.dll*[1]
Java 同期論理	● *Java¥activation.jar*[2] ● *Java¥imap.jar*[2] ● *Java¥jodbc4.jar* ● *Java¥mailapi.jar*[2] ● *Java¥mlscript.jar* ● *Java¥mlsupport.jar* ● *Java¥pop3.jar*[2] ● *Java¥smtp.jar*[2] ● *Bin32¥mljava16.dll* ● *Bin32¥dbjodbc16.dll* ● *Bin32¥mljodbc16.dll*

説明	Windows ファイル
.NET 同期論理	● *MobiLink¥Setup¥Dnet¥mlDomConfig.xml* ● *Bin32¥mldnet16.dll* ● *Bin32¥dnetodbc16.dll* ● *Assembly¥V2¥iAnywhere.MobiLink.dll* ● *Assembly¥V2¥iAnywhere.MobiLink.Script.dll* ● *Assembly¥V2¥iAnywhere.MobiLink.Script.xml* ● *Bin32¥mlDomConfig.xsd*
セキュリティオプション [3]	● *Bin32¥mlrsa_tls16.dll* ● *Bin32¥mlrsa_tls_fips16.dll* ● *Bin32¥sbgse2.dll*
設定スクリプト (統合データベース用のファイルを配備)	● *MobiLink¥Setup¥* ● *MobiLink¥Upgrade¥*
mluser ユーティリティ	● *Bin32¥mluser.exe* ● *Bin32¥mlodbc16.dll* ● *Bin32¥dbicu16.dll* ● *Bin32¥dbicudt16.dll*
mlstop ユーティリティ	● *Bin32¥mlstop.exe* ● *Bin32¥dbicu16.dll*
mlreplay ユーティリティ [6]	● *Bin32¥mlreplay.exe* ● *Bin32¥mlgenreplayapi.exe*
Mobile Link 監視サーバ	● *Bin32¥dbserv16.dll* ● *Bin32¥mlarb16.exe* ● *Bin32¥mlarb16.lic* ● *Bin32¥mlarbiter.bat* ● *MobiLink¥mlarbiter.control* ● *mlarbstop.exe*
Mobile Link プロファイラ	● *Java¥mlprof.jar* ● *Java¥mlstream.jar* ● *Java¥JComponents1600.jar* ● *Java¥jsyblib1600.jar* ● *Bin32¥jsyblib1600.dll* ● *Bin32¥mlprof.exe* ● *Bin32¥mljstrm16.dll* Mobile Link プロファイラのセキュリティ用 [3] ● *Bin32¥mlcrsa16.dll* ● *Bin32¥mlcrsafips16.dll* ● *Bin32¥mlczlib16.dll*

説明	Windows ファイル
Mobile Link 16 プラグインと Mobile Link プロファイラのオンラインヘルプ	● *¥Documentation¥en¥htmlhelp¥sqlanywhere_en16.chm*[1] ● *¥Documentation¥en¥htmlhelp¥sqlanywhere_en16.map*[1]
Notifier	● *Java¥activation.jar*[2] ● *Java¥jodbc4.jar* ● *Java¥mailapi.jar*[2] ● *Java¥mlnotif.jar* ● *Java¥mlscript.jar* ● *Java¥smtp.jar*[2] ● *Bin32¥mljodbc16.dll* ● *Bin32¥mljstrm16.dll* Notifier のセキュリティ用 [3] ● *Bin32¥mlcrsa16.dll* ● *Bin32¥mlcrsafips16.dll* ● *Bin32¥mlczlib16.dll*
Relay Server Outbound Enabler	● *Bin32¥rsoe.exe* Outbound Enabler のセキュリティ用 ● *Bin32¥mlcrsa16.dll*
統合化 Outbound Enabler	● *Bin32¥rsoesupp16.dll*

[1] ドイツ語、日本語、中国語の版では、それぞれ *dblgde16.dll*、*dblgja16.dll*、および *dblgzh16.dll* を使用します。

[2] アプリケーションを再配布する場合は、これらのファイルを Oracle から直接入手してください。

[3] FIPS を使用するには、別途ライセンスが必要な SQL Anywhere セキュリティオプションを入手する必要があります。このセキュリティオプションは、輸出規制対象品目です。RSA セキュリティは、バージョン 10 以降の SQL Anywhere に付属しています。 このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

[4] 生成されたコードをコンパイルするには、*SDK/Include* 内にある **mlreplay** で始まるファイルが必要です。

Windows の 64 ビットアプリケーション

すべてのディレクトリは、*%SQLANY16%* を基準とした相対ディレクトリです。 32 ビット Windows 環境のファイル構造の詳細については、「Windows の 32 ビットアプリケーション」766 ページを参照してください。

説明	Windows ファイル
Mobile Link サーバ	● *Bin64¥mlodbc16.dll* ● *Bin64¥mlsrv16.exe* ● *Bin64¥mlsrv16.lic* ● *Bin64¥mlsql16.dll* ● *Bin64¥dbicu16.dll* ● *Bin64¥dbicudt16.dll*
言語ライブラリ	● *Bin64¥dblgen16.dll*[1]
Java 同期論理	● *Java¥activation.jar*[2] ● *Java¥imap.jar*[2] ● *Java¥jodbc4.jar* ● *Java¥mailapi.jar*[2] ● *Java¥mlscript.jar* ● *Java¥mlsupport.jar* ● *Java¥pop3.jar*[2] ● *Java¥smtp.jar*[2] ● *Bin64¥mljava16.dll* ● *Bin64¥dbjodbc16.dll* ● *Bin64¥mljodbc16.dll*
.NET 同期論理	● *MobiLink¥Setup¥Dnet¥mlDomConfig.xml* ● *Bin64¥mldnet16.dll* ● *Bin64¥dnetodbc16.dll* ● *Assembly¥V2¥iAnywhere.MobiLink.dll* ● *Assembly¥V2¥iAnywhere.MobiLink.Script.dll* ● *Assembly¥V2¥iAnywhere.MobiLink.Script.xml* ● *Bin64¥mlDomConfig.xsd*
セキュリティオプション [3]	● *Bin64¥mlrsa_tls16.dll* ● *Bin64¥mlrsa_tls_fips16.dll* ● *Bin64¥sbgse2.dll*
設定スクリプト (統合データベース用のファイルを配備)	● *MobiLink¥Setup¥* ● *MobiLink¥Upgrade¥*
mluser ユーティリティ	● *Bin64¥mluser.exe* ● *Bin64¥mlodbc16.dll* ● *Bin64¥dbicu16.dll* ● *Bin64¥dbicudt16.dll*
mlstop ユーティリティ	● *Bin64¥mlstop.exe* ● *Bin64¥dbicu16.dll*
mlreplay ユーティリティ [6]	● *Bin64¥mlreplay.exe* ● *Bin64¥mlgenreplayapi.exe*

説明	Windows ファイル
Mobile Link 監視サーバ	● *Bin64¥dbserv16.dll* ● *Bin64¥mlarb16.exe* ● *Bin64¥mlarb16.lic* ● *Bin64¥mlarbiter.bat* ● *MobiLink¥mlarbiter.control* ● *mlarbstop.exe*
Mobile Link プロファイラ	● *Java¥mlprof.jar* ● *Java¥mlstream.jar* ● *Java¥JComponents1600.jar* ● *Java¥jsyblib1600.jar* ● *Bin64¥jsyblib1600.dll* ● *Bin64¥mlprof.exe* ● *Bin64¥mljstrm16.dll* Mobile Link プロファイラのセキュリティ用 [3] ● *Bin64¥mlcrsa16.dll* ● *Bin64¥mlcrsafips16.dll* ● *Bin64¥mlczlib16.dll*
Mobile Link 16 プラグインと Mobile Link プロファイラのオンラインヘルプ	● *¥Documentation¥en¥htmlhelp ¥sqlanywhere_en16.chm*[1] ● *¥Documentation¥en¥htmlhelp ¥sqlanywhere_en16.map*[1]
Notifier	● *Java¥activation.jar*[2] ● *Java¥jodbc4.jar* ● *Java¥mailapi.jar*[2] ● *Java¥mlnotif.jar* ● *Java¥mlscript.jar* ● *Java¥smtp.jar*[2] ● *Bin64¥mljodbc16.dll* ● *Bin64¥mljstrm16.dll* Notifier のセキュリティ用 [3] ● *Bin64¥mlcrsa16.dll* ● *Bin64¥mlcrsafips16.dll* ● *Bin64¥mlczlib16.dll*
Relay Server Outbound Enabler	● *Bin64¥rsoe.exe* Outbound Enabler のセキュリティ用 ● *Bin64¥mlcrsa16.dll*

説明	Windows ファイル
統合化 Outbound Enabler	● *Bin64¥rsoesupp16.dll*

[1] ドイツ語、日本語、中国語の版では、それぞれ *dblgde16.dll*、*dblgja16.dll*、および *dblgzh16.dll* を使用します。

[2] アプリケーションを再配布する場合は、これらのファイルを Oracle から直接入手してください。

[3] FIPS を使用するには、別途ライセンスが必要な SQL Anywhere セキュリティオプションを入手する必要があります。このセキュリティオプションは、輸出規制対象品目です。RSA セキュリティは、バージョン 10 以降の SQL Anywhere に付属しています。 このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

[4] 生成されたコードをコンパイルするには、*SDK/Include* 内にある **mlreplay** で始まるファイルが必要です。

UNIX、Linux、Mac OS X での UNIX の 32 ビットアプリケーション

すべてのディレクトリは、*$SQLANY16* を基準とした相対ディレクトリです。 64 ビット UNIX 環境のファイル構造の詳細については、「UNIX の 64 ビットアプリケーション (UNIX と Linux)」774 ページを参照してください。

説明	UNIX ファイル
Mobile Link サーバ	● *bin32/mlsrv16* ● *bin32/mlsrv16.lic* ● *lib32/libdbodm16.so*[3] ● *lib32/libmlodbc16_r.so*[3] ● *lib32/libmlsql16_r.so*[3] ● *lib32/libdbtasks16_r.so*[3] ● *lib32/libdbicu16_r.so*[3] ● *lib32/libdbicudt16_r.so*[3] ● *lib32/libdbodbcinst16_r.so*[3]
言語ライブラリ	● *res/dblgen16.res*[1]

説明	UNIX ファイル
Java 同期論理	● *java/activation.jar*[2] ● *java/imap.jar*[2] ● *java/jodbc4.jar* ● *java/mailapi.jar*[2] ● *java/mlscript.jar* ● *java/mlsupport.jar* ● *java/pop3.jar*[2] ● *java/smtp.jar*[2] ● *lib32/libmljava16_r.so*[3] ● *lib32/libmljodbc16.so*[3]
.NET 同期論理	● 該当なし
セキュリティオプション [4]	● *lib32/libmlrsa_tls16_r.so*[3]
設定スクリプト (統合データベース用のファイルを配備)	● *mobiLink/setup* ● *mobilink/upgrade*
mluser ユーティリティ	● *bin32/mluser* ● *lib32/libmlodbc16_r.so*[3] ● *lib32/libdbicu16.so*[3] ● *lib32/libdbicudt16.so*[3]
mlstop ユーティリティ	● *bin32/mlstop* ● *lib32/libdbicu16.so*[3]
mlreplay ユーティリティ [6]	● *bin32/mlreplay* ● *bin32/mlgenreplayapi*
Mobile Link 監視サーバ	● *bin32/libdbserv16_r.so* ● *bin32/mlarb16* ● *bin32/mlarb16.lic* ● *bin32/mlarbiter.sh* ● *mobilink/mlarbiter.control* ● **mlarbstop.exe**

説明	UNIX ファイル
Mobile Link プロファイラ	● *bin32/mlprof* ● *java/mlprof.jar* ● *java/mlstream.jar* ● *java/JComponents1600.jar* ● *java/jsyblib1600.jar* ● *lib32/libjsyblib1600_r.so*[3] プロファイラのセキュリティ用： ● *lib32/libmlcrsa16_r.so* ● *lib32/libmlcrsafips16_r.so* ● *lib32/libmlczlib16_r.so*
Notifier	● *java/activation.jar*[2] ● *java/jodbc4.jar* ● *java/mailapi.jar*[2] ● *java/mlnotif.jar* ● *java/mlscript.jar* ● *java/smtp.jar*[2] ● *lib32/libmljstrm16_r.so*[3]
Relay Server Outbound Enabler	● *bin32/rsoe* Outbound Enabler のセキュリティ用 ● *lib32/libmlcrsa16_r.so*[3]
統合化 Outbound Enabler	● *lib32¥librsoesupp16_r.so*

[1] ドイツ語、日本語、中国語の版では、それぞれ *dblgde16.dll*、*dblgja16.dll*、および *dblgzh16.dll* を使用します。

[2] アプリケーションを再配布する場合は、これらのファイルを Oracle から直接入手してください。

[3] Linux のファイル拡張子は *.so* です。Mac OS X のファイル拡張子は *.dylib* です。

[4] トランスポートレイヤセキュリティを使用するには、別途ライセンスが必要な SQL Anywhere セキュリティオプションを入手する必要があります。このセキュリティオプションは、輸出規制対象品目です。 このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

[5] 生成されたコードをコンパイルするには、*sdk/include* 内にある **mlreplay** で始まるファイルが必要です。

UNIX の 64 ビットアプリケーション (UNIX と Linux)

すべてのディレクトリは、*$SQLANY16* を基準とした相対ディレクトリです。 32 ビット UNIX 環境のファイル構造の詳細については、「UNIX、Linux、Mac OS X での UNIX の 32 ビットアプリケーション」771 ページを参照してください。

説明	UNIX ファイル
Mobile Link サーバ	● *bin64/mlsrv16* ● *bin64/mlsrv16.lic* ● *lib64/libdbodm16.so*[3] ● *lib64/libmlodbc16_r.so*[3] ● *lib64/libmlsql16_r.so*[3] ● *lib64/libdbtasks16_r.so*[3] ● *lib64/libdbicu16_r.so*[3] ● *lib64/libdbicudt16_r.so*[3] ● *lib64/libdbodbcinst16_r.so*[3]
言語ライブラリ	● *res/dblgen16.res*[1]
Java 同期論理	● *java/activation.jar*[2] ● *java/imap.jar*[2] ● *java/jodbc4.jar* ● *java/mailapi.jar*[2] ● *java/mlscript.jar* ● *java/mlsupport.jar* ● *java/pop3.jar*[2] ● *java/smtp.jar*[2] ● *lib64/libmljava16_r.so*[3] ● *lib64/libmljodbc16.so*[3]
.NET 同期論理	● 該当なし
セキュリティオプション [4]	● *lib64/libmlrsa_tls16_r.so*[3]
設定スクリプト (統合データベース用のファイルを配備)	● *mobiLink/setup* ● *mobilink/upgrade*
mluser ユーティリティ	● *bin64/mluser* ● *lib64/libmlodbc16_r.so*[3] ● *lib64/libdbicu16.so*[3] ● *lib64/libdbicudt16.so*[3]
mlstop ユーティリティ	● *bin64/mlstop* ● *lib64/libdbicu16.so*[3]
mlreplay ユーティリティ [6]	● *bin64/mlreplay* ● *bin64/mlgenraplyapi*

説明	UNIX ファイル
Mobile Link 監視サーバ	● *bin64/libdbserv16_r.so* ● *bin64/mlarb16* ● *bin64/mlarb16.lic* ● *bin64/mlarbiter.sh* ● *mobilink/mlarbiter.control* ● **mlarbstop.exe**
Mobile Link プロファイラ	● *bin64/mlprof* ● *java/mlprof.jar* ● *java/mlstream.jar* ● *java/JComponents1600.jar* ● *java/jsyblib1600.jar* ● *lib64/libjsyblib1600_r.so*[3] プロファイラのセキュリティ用： ● *lib64/libmlcrsa16_r.so* ● *lib64/libmlcrsafips16_r.so* ● *lib64/libmlczlib16_r.so*
Notifier	● *java/activation.jar*[2] ● *java/jodbc4.jar* ● *java/mailapi.jar*[2] ● *java/mlnotif.jar* ● *java/mlscript.jar* ● *java/smtp.jar*[2]
Relay Server Outbound Enabler	● *bin64/rsoe* Outbound Enabler のセキュリティ用 ● *lib64/libmlcrsa16_r.so*[3]
統合化 Outbound Enabler	● *lib64¥librsoesupp16_r.so*

[1] ドイツ語、日本語、中国語の版では、それぞれ *dblgde16.dll*、*dblgja16.dll*、および *dblgzh16.dll* を使用します。

[2] アプリケーションを再配布する場合は、これらのファイルを Oracle から直接入手してください。

[3] Solaris SPARC および Linux のファイル拡張子は *.so* です。IBM AIX のファイル拡張子は *.a* です。

[4] トランスポートレイヤセキュリティを使用するには、別途ライセンスが必要な SQL Anywhere セキュリティオプションを入手する必要があります。このセキュリティオプションは、輸出規制

対象品目です。 このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

[5] 生成されたコードをコンパイルするには、*sdk/include* 内にある **mlreplay** で始まるファイルが必要です。

SQL Anywhere Mobile Link クライアントの配備

説明

- SQL Anywhere クライアントの場合、SQL Anywhere データベースサーバと Mobile Link クライアントを配備する必要があります。

  「データベースおよびアプリケーションの配備」『SQL Anywhere サーバ プログラミング』を参照してください。

- Mobile Link 同期クライアントを再配布する場合は、SQL Anywhere データベースに必要なファイルのほかに、以下のファイルをインストール環境に含める必要があります。

- 以下のファイルを配備するときは、特に指定がないかぎり、ファイルを同じディレクトリ構造に配置してください。

- Sybase Central を配備する方法については、「管理ツールの配備」『SQL Anywhere サーバ プログラミング』を参照してください。

- Windows 用の配備ウィザードがあります。 「Deployment ウィザード」『SQL Anywhere サーバ プログラミング』を参照してください。

- Windows Mobile の配備環境の場合、以下で *bin32* ディレクトリにあると示されているファイルは、*ce¥arm.50* フォルダにあります。また、.NET アセンブリは *ce¥Assembly¥v2* ディレクトリにあります。

Windows アプリケーション

すべてのディレクトリは、*%SQLANY16%* を基準とした相対ディレクトリです。これらのファイルの 64 ビットバージョンは、*Bin64* ディレクトリにあります。

説明	Windows ファイル
Mobile Link 同期クライアント (dbmlsync)	● *Bin32¥dbcon16.dll*[2] ● *Bin32¥dbicu16.dll*[3] ● *Bin32¥dblgen16.dll*[1] ● *Bin32¥dblib16.dll* ● *Bin32¥dbmlsync.exe* ● *Bin32¥dbtool16.dll*[2]
Dbmlsync C++ API または SQL SYNCHRONIZE 文	● Mobile Link 同期クライアントファイル ● Bin32¥dbmlsynccli16.dll

説明	Windows ファイル
Dbmlsync .NET API	● Mobile Link 同期クライアントファイル ● *Assembly¥V2¥iAnywhere.MobiLink.Client.dll*
セキュリティオプション [2]	● *Bin32¥mlcrsa16.dll* ● *Bin32¥mlcrsafips16.dll* ● *Bin32¥sbgse2.dll*
Microsoft ActiveSync ユーティリティ	● *Bin32¥mlasinst.exe* ● *Bin32¥mlasdesk.dll* ● *Bin32¥dbcon16.exe* ● *CE¥chip¥mlasdev.dll* (*chip* は、arm.50 などのサポートされる任意のチップ)
Listener	● *Bin32¥dblgen16.dll*[1] ● *Bin32¥dblsn.exe* ● *Bin32¥lsn_udp16.dll* ● *Bin32¥lsn_swi510.dll* ● *Bin32¥maac555.dll* ● *Bin32¥maac750.dll* ● *Bin32¥maac750r3.dll* ● *Bin32¥mabridge.dll* ● *dblsn_sms16.dll*[4]

説明	Windows ファイル
Mobile Link エージェント (リモートデータベースの集中管理に必要)	SQL Anywhere リモートデータベースを管理するには、次のファイルがリモートデバイスにあることが必要です。 ● *dbcon16.dll* ● *dbeng16.exe*[5] ● *dbeng16.lic*[5] ● *dbghelp.dll* ● *dbicu16.dll* ● *dbicudt16.dll*[5] ● *dbicudt16.dat*[4] ● *dblgen16.dll*[1] ● *dblib16.dll* ● *dbmlsync.exe* ● *dbmlsynccli16.dll* ● *dbscript16.dll* ● *dbsrv16.exe* ● *dbsrv16.lic* ● *dbtool16.dll* ● *mlagent.exe* ● *mlastop.exe* ● *mlasaadapter16.dll* SQL Anywhere の特定の機能では、追加ファイルの配備が必要になる場合があります。「データベースおよびアプリケーションの配備」『SQL Anywhere サーバ プログラミング』を参照してください。

[1] ドイツ語、日本語、中国語の版では、それぞれ *dblgde16.dll*、*dblgja16.dll*、および *dblgzh16.dll* を使用します。

[2] Windows Mobile では dbtools インタフェースを使用しないかぎり不要です。

[3] dbinit -zn UTF8BIN を使用してデータベースを初期化する場合は不要です。 「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ データベース管理』を参照してください。

[4] Windows Mobile 専用です。

[5] Windows オペレーティングシステム (Windows Mobile を除く) 用です。

UNIX、Linux、Mac OS X での UNIX アプリケーション

すべてのディレクトリは、*$SQLANY16* を基準とした相対ディレクトリです。

説明	UNIX ファイル
Mobile Link 同期クライアント	● *bin32/dbmlsync* ● *res/dblgen16.res* ● *lib32/libdbicu16_r.so*[1] ● *lib32/libdblib16_r.so*[1] ● *lib32/libdbtool16_r.so*[1]
Dbmlsync C++ API または SQL SYNCHRONIZE 文	● Mobile Link 同期クライアントファイル ● *lib32/libdbmlsynccli16_r.so*
セキュリティオプション [2]	● *lib32/libmlcrsa16_r.so*[1]

[1] Linux のファイル拡張子は *.so* です。Mac OS X のファイル拡張子は *.dylib* です。

[2] トランスポートレイヤセキュリティを使用するには、別途ライセンスが必要な SQL Anywhere セキュリティオプションを入手する必要があります。このセキュリティオプションは、輸出規制対象品目です。 このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 16 紹介』を参照してください。

Ultra Light Mobile Link クライアントの配備

Ultra Light クライアントの場合、Ultra Light ランタイムライブラリまたは Ultra Light コンポーネントには、必要な同期ストリーム機能が含まれています。Ultra Light ランタイムライブラリは、アプリケーションにコンパイルされます。配備はライセンス契約に応じて決まります。

説明	Windows ファイル
Mobile Link エージェント	Ultra Light リモートデータベースを管理するには、次のファイルがリモートデバイスにあることが必要です。 ● *dblgen16.dll*[1] ● *mlagent.exe* ● *mlastop.exe* ● *mlauladapt16.dll* ● *uleng16.exe*

[1] ドイツ語、日本語、中国語の版では、それぞれ *dblgde16.dll*、*dblgja16.dll*、および *dblgzh16.dll* を使用します。

参照

- 「Ultra Light の配備」『Ultra Light データベース管理とリファレンス』
- C/C++ : 「Ultra Light C++ アプリケーションの構築と配備」『Ultra Light C/C++ プログラミング』
- .NET : 「レッスン 5 : アプリケーションの構築と配備」『Ultra Light .NET プログラミング』

索引

記号

.NET
Mobile Link オブジェクトベースデータフロー, 623
Mobile Link データ型, 537
Mobile Link 同期スクリプト, 533
.NET CLR
Mobile Link オプション, 76
.NET Mobile Link サーバ API (*参照* .NET 用 Mobile Link サーバ API)
.NET からの情報の出力
Mobile Link .NET 同期論理, 540
.NET クラス
.NET 同期論理のためのインスタンス化, 536
.NET での Mobile Link サーバエラーの処理
Mobile Link .NET 同期論理, 541
.NET 同期のサンプル
Mobile Link .NET 同期論理, 546
.NET 同期論理
.NET クラスのインスタンス化, 536
32 ビットの UNIX での配備, 771
32 ビットの Windows での配備, 766
64 ビットの UNIX での配備, 774
64 ビットの Windows での配備, 768
Mobile Link のパフォーマンス, 147
サポートされる言語, 533
サンプル, 546
設定, 533
デバッグ, 542
メソッド, 538
.NET 同期論理の実行
説明, 533
.NET 同期論理の設定
説明, 533
.NET 同期論理のデバッグ
説明, 542
.NET の同期の方法
説明, 544
@data オプション
Mobile Link サーバオプション (mlsrv16), 49
Mobile Link 停止ユーティリティ (mlstop), 693
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
@EmployeeID 変数
Mobile Link プライマリキープールで使用, 123
1 つの SQL 文で複数のエラーが処理される場合
Mobile Link, 267
1 方向同期
説明, 116
-ap オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-a オプション
Mobile Link サーバオプション (mlsrv16), 50
-bn オプション
Mobile Link サーバオプション (mlsrv16), 51
-b オプション
Mobile Link サーバオプション (mlsrv16), 50
-ca オプション
Mobile Link サーバオプション (mlsrv16), 52
-cinit オプション
Mobile Link サーバオプション (mlsrv16), 53
-classic オプション
Mobile Link mlsrv16 -sl java オプション, 78
-classpath オプション
Mobile Link mlsrv16 -sl java オプション, 78
-clrConGC オプション
Mobile Link mlsrv16 -sl dnet オプション, 76
-clrFlavor オプション
Mobile Link mlsrv16 -sl dnet オプション, 76
-clrVersion オプション
Mobile Link mlsrv16 -sl dnet オプション, 76
-cmax オプション
Mobile Link サーバオプション (mlsrv16), 54
-cmin オプション
Mobile Link サーバオプション (mlsrv16), 54
-cm オプション
Mobile Link サーバオプション (mlsrv16), 54
-cn オプション
Mobile Link サーバオプション (mlsrv16), 55
-cp オプション
Mobile Link mlsrv16 -sl java オプション, 78
-cr オプション
Mobile Link サーバオプション (mlsrv16), 56
-cs オプション
Mobile Link サーバオプション (mlsrv16), 56
-ct オプション
Mobile Link サーバオプション (mlsrv16), 57
-c オプション

Mobile Link サーバオプション (mlsrv16), 52
Mobile Link プロファイラ (dbmlsync), 193
Mobile Link ユーザ認証ユーティリティ (mluser), 694
-dl オプション
Mobile Link サーバオプション (mlsrv16), 57
Mobile Link ユーザ認証ユーティリティ (mluser), 694
-DMLStartClasses オプション
Java ユーザ定義起動クラス, 467
Mobile Link mlsrv16 -sl java オプション, 78
-dr オプション
Mobile Link サーバオプション (mlsrv16), 58
-dsd オプション
Mobile Link サーバオプション (mlsrv16), 59
-ds オプション
Mobile Link サーバオプション (mlsrv16), 58
-dt オプション
Mobile Link サーバオプション (mlsrv16), 59
-d オプション
Mobile Link mlsrv16 -sl java オプション, 78
Mobile Link ユーザ認証ユーティリティ (mluser), 694
生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi), 704
-esu オプション
Mobile Link サーバオプション (mlsrv16), 61
-et オプション
Mobile Link サーバオプション (mlsrv16), 61
-e オプション
Mobile Link サーバオプション (mlsrv16), 60
-ftru オプション
Mobile Link サーバオプション (mlsrv16), 63
-ftr オプション
Mobile Link サーバオプション (mlsrv16), 62
-f オプション
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
-hotspot オプション
Mobile Link mlsrv16 -sl java オプション, 78
-h オプション
Mobile Link 停止ユーティリティ (mlstop), 693
-jrepath オプション
Mobile Link mlsrv16 -sl java オプション, 78
-ldt オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-lsc オプション
Mobile Link サーバオプション (mlsrv16), 63
-ls オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-MLAutoLoadPath オプション
Mobile Link mlsrv16 -sl dnet オプション, 76
説明, 544
-MLDomConfigFile オプション
Mobile Link mlsrv16 -sl dnet オプション, 76
説明, 544
-MLStartClasses
.NET ユーザ定義起動クラス, 539
Mobile Link mlsrv16 -sl dnet オプション, 76
-mt オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-nc オプション
Mobile Link サーバ (mlsrv16), 64
-notifier オプション
Mobile Link サーバオプション (mlsrv16), 65
-n オプション
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
-on オプション
Mobile Link サーバオプション (mlsrv16), 66
-oq オプション
Mobile Link サーバオプション (mlsrv16), 67
-os オプション
Mobile Link サーバオプション (mlsrv16), 67
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
-ot オプション
Mobile Link サーバオプション (mlsrv16), 68
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi), 704
-o オプション
Mobile Link サーバオプション (mlsrv16), 65

Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi), 704
-pc オプション
Mobile Link ユーザ認証ユーティリティ (mluser), 694
-ping オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-ppv オプション
Mobile Link サーバオプション (mlsrv16), 68
-p オプション
Mobile Link プロファイラ (mlprof), 193
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
-q オプション
Mobile Link サーバオプション (mlsrv16), 73
Mobile Link 停止ユーティリティ (mlstop), 693
-rd オプション
Mobile Link サーバオプション (mlsrv16), 74
-rep オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-rg オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-rp オプション
Mobile Link サーバオプション (mlsrv16), 75
Mobile Link リプレイユーティリティ (mlreplay), 697
-rrp オプション
Mobile Link サーバオプション (mlsrv16), 74
-r オプション
Mobile Link サーバオプション (mlsrv16), 73
Mobile Link プロファイラ (mlprof), 193
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
-sci オプション
Mobile Link リプレイユーティリティ (mlreplay), 697
-server オプション
Mobile Link mlsrv16 -sl java オプション, 78
-sl dnet オプション
-MLAutoLoadPath の使用, 535
-MLDomConfigFile の使用, 544
Mobile Link サーバオプション (mlsrv16), 76
ユーザ定義起動クラス, 539
-sl java オプション
Mobile Link サーバオプション (mlsrv16), 78
ユーザ定義起動クラス, 467
-sm オプション
Mobile Link サーバオプション (mlsrv16), 79
-sv オプション
Mobile Link Replay ユーティリティ (mlreplay), 697
-s オプション
Mobile Link サーバオプション (mlsrv16), 75
-tc オプション
Mobile Link サーバオプション (mlsrv16), 80
-tf オプション
Mobile Link サーバオプション (mlsrv16), 81
-ts オプション
Mobile Link サーバオプション (mlsrv16), 82
-tx オプション
Mobile Link サーバオプション (mlsrv16), 83
-t オプション
Mobile Link 停止ユーティリティ (mlstop), 693
-ud オプション
Mobile Link サーバオプション (mlsrv16), 84
-ui オプション
Mobile Link サーバオプション (mlsrv16), 84
-urc オプション
Mobile Link のパフォーマンスの利点, 147
-ux オプション
Mobile Link サーバオプション (mlsrv16), 85
-u オプション
Mobile Link プロファイラ (mlprof), 193
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 697
-v+ オプション
Mobile Link サーバオプション (mlsrv16), 85
-vc オプション
Mobile Link サーバオプション (mlsrv16), 85
-verbose オプション
Mobile Link mlsrv16 -sl java オプション, 78
-ve オプション
Mobile Link サーバオプション (mlsrv16), 85

-vf オプション
Mobile Link サーバオプション (mlsrv16), 85
-vh オプション
Mobile Link サーバオプション (mlsrv16), 85
-vm オプション
Mobile Link サーバオプション (mlsrv16), 85
-vn オプション
Mobile Link サーバオプション (mlsrv16), 85
-vp オプション
Mobile Link サーバオプション (mlsrv16), 85
-vr オプション
Mobile Link サーバオプション (mlsrv16), 85
-vs オプション
Mobile Link サーバオプション (mlsrv16), 85
-vt オプション
Mobile Link サーバオプション (mlsrv16), 85
-vu オプション
Mobile Link サーバオプション (mlsrv16), 85
-v オプション
Mobile Link SQL Anywhere クライアントユーティリティ (dbmlsync) のパフォーマンス, 146
Mobile Link サーバオプション (mlsrv16), 85
-wm オプション
Mobile Link サーバオプション (mlsrv16), 90
-wn mlsrv16 オプション
パフォーマンス, 143
-wn オプション
Mobile Link サーバオプション (mlsrv16), 80
-wu オプション
Mobile Link サーバオプション (mlsrv16), 91
-w オプション
Mobile Link サーバオプション (mlsrv16), 89
Mobile Link 停止ユーティリティ (mlstop), 693
-x オプション
Mobile Link mlsrv16 -sl java オプション, 78
Mobile Link サーバオプション (mlsrv16), 92
Mobile Link プロファイラ (mlprof), 193
Mobile Link リプレイユーティリティ (mlreplay), 697
-zf オプション
Mobile Link サーバオプション (mlsrv16), 103
-zp オプション
Mobile Link サーバオプション (mlsrv16), 103
-zs オプション
Mobile Link サーバオプション (mlsrv16), 104
共有サーバステータス, 104
-zt オプション
Mobile Link サーバオプション (mlsrv16), 104
-zus オプション
Mobile Link サーバオプション (mlsrv16), 105
-zu オプション
Mobile Link サーバオプション (mlsrv16), 104
-zwd オプション
Mobile Link サーバオプション (mlsrv16), 106
-zwe オプション
Mobile Link サーバオプション (mlsrv16), 106
-zw オプション
Mobile Link サーバオプション (mlsrv16), 105

A

a.
Mobile Link 名前付きパラメータのプレフィクス, 235
Mobile Link ユーザ定義のパラメータのプレフィクス, 251
active プロパティ
Mobile Link プロファイラの同期統計情報, 208
Adaptive Server Enterprise
begin_connection_autocommit イベント, 301
Mobile Link での DDL の使用, 301
Mobile Link データマッピング, 708
Mobile Link 同期, 11
Mobile Link 統合データベース, 10
Mobile Link 独立性レベル, 140
StaticCursorLongColBuffLen, 11
addErrorListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 503
addInfoListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 503
addShutdownListenerr メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 504
addWarningListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 505
Add メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 573
AdventureWorks
同期の問題, 16
agent_db_file
リモートタスク, 187
agent_id

リモートタスク, 187
agent_log_file
リモートタスク, 187
API
Java 用 Mobile Link サーバ API, 473
ASE
(*参照* Adaptive Server Enterprise)
authenticate_file_transfer
接続イベント, 282
authenticate_file_upload
接続イベント, 285
authenticate_parameters
接続イベント, 287
authenticate_user
接続イベント, 291
authenticate_user_hashed
接続イベント, 297
authenticate_user プロパティ
Mobile Link プロファイラの同期統計情報, 208
authentication_status 同期パラメータ
説明, 291
AUTOINCREMENT
(*参照* GLOBAL AUTOINCREMENT)
AvantGo (*参照* UltraLite for M-Business Anywhere)

B

battery_level
リモートタスク, 187
begin_connection
接続イベント, 301
begin_connection_autocommit
接続イベント, 301
begin_download
接続イベント, 302
テーブルイベント, 304
begin_download_deletes
テーブルイベント, 307
begin_download_rows
テーブルイベント, 309
begin_publication
接続イベント, 310
begin_synchronization
接続イベント, 314
テーブルイベント, 316
begin_sync プロパティ
Mobile Link プロファイラの同期統計情報, 208
begin_upload
接続イベント, 319
テーブルイベント, 321
begin_upload_deletes
テーブルイベント, 324
begin_upload_rows
テーブルイベント, 326
BLOB
ASE からダウンロードされた BLOB, 11
buffer_size オプション
Mobile Link OE 用 mlsrv16 -x オプション, 100
buffer_size プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95

C

C# プログラミング言語
Mobile Link .NET でのサポート, 533
Mobile Link オプション, 76
Mobile Link 同期スクリプト, 533
C++ プログラミング言語
Mobile Link .NET でのサポート, 533
CHAR カラム
ASE Mobile Link 統合データベース, 11
IBM DB2 Mobile Link 統合データベース, 13
Mobile Link サーバオプション (mlsrv16) -b, 50
Mobile Link の問題, 5
Oracle Mobile Link 統合データベース, 20
SQL Server Mobile Link 統合データベース, 15
CHAR データ型
Mobile Link と他の DBMS, 5
CLASSPATH 環境変数
Mobile Link Java 同期論理, 459
Clear メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 573
ClientCertificates プロパティ
NetworkData インタフェース [MobiLink サーバ .NET API], 595
Close メソッド
DBCommand インタフェース [Mobile Link サーバ .NET API], 558
DBConnection インタフェース [Mobile Link サーバ .NET API], 560
DBRowReader インタフェース [Mobile Link サーバ .NET API], 581
CLR

Mobile Link オプション, 76
collect_network_data プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 92
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
ColumnNames プロパティ
DBRowReader インタフェース [Mobile Link サーバ .NET API], 582
ColumnTypes プロパティ
DBRowReader インタフェース [Mobile Link サーバ .NET API], 582
CommandText プロパティ
DBCommand インタフェース [Mobile Link サーバ .NET API], 559
Commit メソッド
DBConnection インタフェース [Mobile Link サーバ .NET API], 560
Common Language Runtime
Mobile Link オプション, 76
completed プロパティ
Mobile Link プロファイラの同期統計情報, 208
config オプション
Mobile Link OE 用 mlsrv16 -x オプション, 100
conflicted_updates プロパティ
Mobile Link プロファイラの同期統計情報, 208
connection_retries プロパティ
Mobile Link プロファイラの同期統計情報, 208
Contains メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 573
CopyTo メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 574
Count プロパティ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 578
CreateAndInitMLReplayUploadTransaction コールバック [mlreplaycallbacks.cpp]
説明, 635
CreateCommand メソッド
DBConnection インタフェース [Mobile Link サーバ .NET API], 560
createSpatialValue メソッド
SpatialUtilities クラス [Mobile Link サーバ Java API], 513
CreateSpatialValue メソッド
SpatialUtilities クラス [Mobile Link サーバ .NET API], 607

D

DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API]
DateTimeWithTimeZone コンストラクタ, 550
DateTime プロパティ, 554
Day プロパティ, 554
Hour プロパティ, 554
Millisecond プロパティ, 555
Minute プロパティ, 555
Month プロパティ, 555
Parse メソッド, 552
Second プロパティ, 555
TimeZoneHour プロパティ, 555
TimeZoneMinute プロパティ, 556
ToString メソッド, 552
Year プロパティ, 556
説明, 549
DateTimeWithTimeZone コンストラクタ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 550
DateTime プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 554
Day プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 554
db_location
リモートタスク, 187
DBCommand インタフェース [Mobile Link サーバ .NET API]
Close メソッド, 558
CommandText プロパティ, 559
ExecuteNonQuery メソッド, 558
ExecuteReader メソッド, 558
Parameters プロパティ, 559
Prepare メソッド, 558
説明, 556
DBConnectionContext
コンストラクタ, 537
DBConnectionContext インタフェース [Mobile Link サーバ .NET API]

GetConnection メソッド, 562
GetDownloadData メソッド, 563
GetProperties メソッド, 564
GetRemoteID メソッド, 564
GetServerContext メソッド, 565
GetVersion メソッド, 565
NetworkData プロパティ, 566
説明, 561
DBConnectionContext インタフェース [Mobile Link サーバ Java API]
getConnection メソッド, 474
getDownloadData メソッド, 475
getNetworkData メソッド, 475
getProperties メソッド, 476
getRemoteID メソッド, 477
getServerContext メソッド, 477
getVersion メソッド, 478
説明, 473
DBConnection インタフェース [Mobile Link サーバ .NET API]
Close メソッド, 560
Commit メソッド, 560
CreateCommand メソッド, 560
Rollback メソッド, 561
説明, 559
dblsn ユーティリティ
Windows での Mobile Link クライアントの配備, 776
dbmlsync (*参照* dbmlsync ユーティリティ)
dbmlsync ユーティリティ
UNIX での配備, 778
Windows での配備, 776
配備, 776
DBParameterCollection クラス [Mobile Link サーバ .NET API]
Add メソッド, 573
Clear メソッド, 573
Contains メソッド, 573
CopyTo メソッド, 574
Count プロパティ, 578
DBParameterCollection コンストラクタ, 572
GetEnumerator メソッド, 575
IndexOf メソッド, 575
Insert メソッド, 576
IsFixedSize プロパティ, 578
IsReadOnly プロパティ, 578
IsSynchronized プロパティ, 578
RemoveAt メソッド, 577
Remove メソッド, 576
SyncRoot プロパティ, 579
this プロパティ, 579
説明, 571
DBParameterCollection コンストラクタ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 572
DBParameter クラス [Mobile Link サーバ .NET API]
DbType プロパティ, 568
Direction プロパティ, 568
HasChanged フィールド, 570
IsNullable プロパティ, 569
ParameterName プロパティ, 569
Precision プロパティ, 569
Scale プロパティ, 570
Size プロパティ, 570
Value プロパティ, 570
説明, 567
DBRowReader インタフェース [Mobile Link サーバ .NET API]
Close メソッド, 581
ColumnNames プロパティ, 582
ColumnTypes プロパティ, 582
NextRow メソッド, 581
説明, 580
DBTools インタフェース
(*参照* データベースツールインタフェース)
DbType プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 568
DelayCreationOfSimulatedClient コールバック [mlreplaycallbacks.cpp]
説明, 636
DelayDestructionOfSimulatedClient コールバック [mlreplaycallbacks.cpp]
説明, 636
DelayStartOfReplay コールバック [mlreplaycallbacks.cpp]
説明, 637
DestroyMLReplayUploadTransaction コールバック [mlreplaycallbacks.cpp]
説明, 638
Direction プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 568
download_bytes プロパティ
Mobile Link プロファイラの同期統計情報, 208

download_cursor
Mobile Link、切断分割, 114
外部キーテーブルの分割, 116
重複のある分割, 115
ストアドプロシージャコールの使用, 137
ストアドプロシージャコールの使用例, 137
説明, 263
タイムスタンプベースの同期, 109
テーブルイベント, 329
パフォーマンス, 146
ローをダウンロードするスクリプトの作成, 262
download_delete_cursor
外部キーテーブルの分割, 116
重複のある分割, 115
ストアドプロシージャコールの使用, 137
ストアドプロシージャコールの使用例, 137
切断分割, 114
説明, 265
テーブルイベント, 332
パフォーマンス, 146
ローをダウンロードするスクリプトの作成, 262
download_delete_cursor スクリプトを使用したローの削除
Mobile Link, 265
download_deleted_rows プロパティ
Mobile Link プロファイラの同期統計情報, 208
download_errors プロパティ
Mobile Link プロファイラの同期統計情報, 208
download_fetched_rows プロパティ
Mobile Link プロファイラの同期統計情報, 208
download_filtered_rows プロパティ
Mobile Link プロファイラの同期統計情報, 208
download_statistics
接続イベント, 334
テーブルイベント, 337
download_timestamp
Mobile Link でのせいせい, 110
download_warnings プロパティ
Mobile Link プロファイラの同期統計情報, 208
DownloadData インタフェース [Mobile Link サーバ .NET API]
GetDownloadTableByName メソッド, 583
GetDownloadTables メソッド, 584
説明, 582
DownloadData インタフェース [Mobile Link サーバ Java API]
getDownloadTableByName メソッド, 479
getDownloadTables メソッド, 480
説明, 478
DownloadTableData インタフェース [Mobile Link サーバ .NET API]
GetDeleteCommand メソッド, 586
GetLastDownloadTime メソッド, 587
GetName メソッド, 587
GetSchemaTable メソッド, 588
GetUpsertCommand メソッド, 588
説明, 584
DownloadTableData インタフェース [Mobile Link サーバ Java API]
getDeletePreparedStatement メソッド, 483
getLastDownloadTime メソッド, 484
getMetaData メソッド, 485
getName メソッド, 486
getUpsertPreparedStatement メソッド, 487
説明, 481
download プロパティ
Mobile Link プロファイラの同期統計情報, 208
duration プロパティ
Mobile Link プロファイラの同期統計情報, 208

E

e2ee_private_key_password プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 93
e2ee_private_key プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 93
end_connection
接続イベント, 340
end_download
接続イベント, 342
テーブルイベント, 344
end_download_deletes
テーブルイベント, 346
end_download_rows
テーブルイベント, 348
end_publication
接続イベント, 350
end_synchronization

接続イベント, 353
テーブルイベント, 355
end_sync プロパティ
Mobile Link プロファイラの同期統計情報, 208
end_upload
接続イベント, 358
テーブルイベント, 361
end_upload_deletes
テーブルイベント, 363
end_upload_rows
テーブルイベント, 366
equals メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 518
ErrorListener イベント
ServerContext インタフェース [Mobile Link サーバ .NET API], 603
ERROR 変数
LogMessage クラス [Mobile Link サーバ Java API], 494
Excel
Mobile Link との同期, 623
ExecuteNonQuery メソッド
DBCommand インタフェース [Mobile Link サーバ .NET API], 558
ExecuteReader メソッド
DBCommand インタフェース [Mobile Link サーバ .NET API], 558

F

FatalException クラス [Mobile Link サーバ .NET API]
FatalException コンストラクタ, 589
説明, 589
FatalException コンストラクタ
FatalException クラス [Mobile Link サーバ .NET API], 589
file_authentication_code
authenticate_file_transfer パラメータ, 282
FiniIdentifySimulatedClient コールバック [mlreplaycallbacks.cpp]
説明, 638
FIPS オプション
Mobile Link サーバオプション (mlsrv16), 62
Mobile Link ユーザ認証ユーティリティ (mluser), 694
FIPS プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
TCP/IP を使用した Mobile Link mlsrv16 -x オプション, 93
FTP
Mobile Link ファイルベースのダウンロード, 212

G

generate_next_last_download_timestamp
接続イベント, 368
getBytes メソッド
SpatialUtilities クラス [Mobile Link サーバ Java API], 514
GetBytes メソッド
SpatialUtilities クラス [Mobile Link サーバ .NET API], 607
getCertificateChain メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 499
getConnection メソッド
DBConnectionContext インタフェース [Mobile Link サーバ Java API], 474
GetConnection メソッド
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 562
GetDeleteCommand メソッド
DownloadTableData インタフェース [Mobile Link サーバ .NET API], 586
getDeletePreparedStatement メソッド
DownloadTableData インタフェース [Mobile Link サーバ Java API], 483
getDeletes メソッド
UploadedTableData インタフェース [Mobile Link サーバ Java API], 528
GetDeletes メソッド
UploadedTableData インタフェース [Mobile Link サーバ .NET API], 615
GetDownloadApplyTime コールバック [mlreplaycallbacks.cpp]
説明, 639
getDownloadData メソッド
DBConnectionContext インタフェース [Mobile Link サーバ Java API], 475
GetDownloadData メソッド
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 563

getDownloadTableByName メソッド
DownloadData インタフェース [Mobile Link サーバ Java API], 479
GetDownloadTableByName メソッド
DownloadData インタフェース [Mobile Link サーバ .NET API], 583
getDownloadTables メソッド
DownloadData インタフェース [Mobile Link サーバ Java API], 480
GetDownloadTables メソッド
DownloadData インタフェース [Mobile Link サーバ .NET API], 584
GetEnumerator メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 575
getHTTPHeaders メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 499
getHTTPHeaderValues メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 500
GetHTTPHeaderValues メソッド
NetworkData インタフェース [Mobile Link サーバ .NET API], 595
getHTTPHeaderValue メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 500
GetHTTPHeaderValue メソッド
NetworkData インタフェース [Mobile Link サーバ .NET API], 595
getInserts メソッド
UploadedTableData インタフェース [Mobile Link サーバ Java API], 529
GetInserts メソッド
UploadedTableData インタフェース [Mobile Link サーバ .NET API], 616
getLastDownloadTime メソッド
DownloadTableData インタフェース [Mobile Link サーバ Java API], 484
GetLastDownloadTime メソッド
DownloadTableData インタフェース [Mobile Link サーバ .NET API], 587
getMetaData メソッド
DownloadTableData インタフェース [Mobile Link サーバ Java API], 485
UploadedTableData インタフェース [Mobile Link サーバ Java API], 529
GetMLReplayAPIVersion コールバック [mlreplaycallbacks.cpp]
説明, 640
getName メソッド
DownloadTableData インタフェース [Mobile Link サーバ Java API], 486
UploadedTableData インタフェース [Mobile Link サーバ Java API], 531
GetName メソッド
DownloadTableData インタフェース [Mobile Link サーバ .NET API], 587
UploadedTableData インタフェース [Mobile Link サーバ .NET API], 617
getNetworkData メソッド
DBConnectionContext インタフェース [Mobile Link サーバ Java API], 475
getPropertiesByVersion メソッド
ServerContext インタフェース [Mobile Link サーバ .NET API], 601
ServerContext インタフェース [Mobile Link サーバ Java API], 506
getProperties メソッド
DBConnectionContext インタフェース [Mobile Link サーバ Java API], 476
ServerContext インタフェース [Mobile Link サーバ .NET API], 600
ServerContext インタフェース [Mobile Link サーバ Java API], 505
GetProperties メソッド
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 564
getPropertySetNames メソッド
ServerContext インタフェース [Mobile Link サーバ .NET API], 601
ServerContext インタフェース [Mobile Link サーバ Java API], 506
getRemoteID メソッド
DBConnectionContext インタフェース [Mobile Link サーバ Java API], 477
GetRemoteID メソッド
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 564
GetSchemaTable メソッド
DownloadTableData インタフェース [Mobile Link サーバ .NET API], 588
UploadedTableData インタフェース [Mobile Link サーバ .NET API], 617
getServerContext メソッド

DBConnectionContext インタフェース [Mobile Link サーバ Java API], 477
GetServerContext メソッド
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 565
getSRID メソッド
SpatialUtilities クラス [Mobile Link サーバ Java API], 514
GetSRID メソッド
SpatialUtilities クラス [Mobile Link サーバ .NET API], 607
getStartClassInstances メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 507
GetStartClassInstances メソッド
ServerContext インタフェース [Mobile Link サーバ .NET API], 602
getText メソッド
LogMessage クラス [Mobile Link サーバ Java API], 493
getTimeZoneOffsetHours メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 520
getTimeZoneOffsetMinutes メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 520
getType メソッド
LogMessage クラス [Mobile Link サーバ Java API], 493
getUpdates メソッド
UploadedTableData インタフェース [Mobile Link サーバ Java API], 531
GetUpdates メソッド
UploadedTableData インタフェース [Mobile Link サーバ .NET API], 618
getUploadedTableByName メソッド
UploadData インタフェース [Mobile Link サーバ Java API], 525
GetUploadedTableByName メソッド
UploadData インタフェース [Mobile Link サーバ .NET API], 613
getUploadedTables メソッド
UploadData インタフェース [Mobile Link サーバ Java API], 525
GetUploadedTables メソッド
UploadData インタフェース [Mobile Link サーバ .NET API], 613
GetUploadTransaction コールバック [mlreplaycallbacks.cpp]
説明, 640
GetUpsertCommand メソッド
DownloadTableData インタフェース [Mobile Link サーバ .NET API], 588
getUpsertPreparedStatement メソッド
DownloadTableData インタフェース [Mobile Link サーバ Java API], 487
getUser メソッド
LogMessage クラス [Mobile Link サーバ Java API], 493
getValue メソッド
InOutInteger インタフェース [Mobile Link サーバ Java API], 489
InOutString インタフェース [Mobile Link サーバ Java API], 491
getVersion メソッド
DBConnectionContext インタフェース [Mobile Link サーバ Java API], 478
GetVersion メソッド
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 565
global_database_id オプション
Mobile Link 設定, 120
GLOBAL AUTOINCREMENT
(*参照* AUTOINCREMENT)
GlobalFini コールバック [mlreplaycallbacks.cpp]
説明, 641
GlobalInit コールバック [mlreplaycallbacks.cpp]
説明, 642
GUID
(*参照* UUID)

H

handle_DownloadData
接続イベント, 371
handle_error
接続イベント, 374
同期スクリプト, 267
handle_odbc_error
接続イベント, 379
handle_UploadData
接続イベント, 383
HasChanged フィールド
DBParameter クラス [Mobile Link サーバ .NET API], 570

header_limit プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95
Mobile Link OE 用 mlsrv16 -x オプション, 100
host プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 92
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
Hour プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 554
HTTP
Mobile Link mlsrv16 -x オプション, 95
HTTPHeaders プロパティ
NetworkData インタフェース [MobiLink サーバ .NET API], 596
HTTPS
Mobile Link mlsrv16 -x オプション, 97

I

ianywhere.ml.script パッケージ
Mobile Link サーバ Java API リファレンス, 473
iAnywhere.MobiLink.Script ネームスペース
Mobile Link サーバ .NET API リファレンス, 548
IBM DB2
LUW の Mobile Link データマッピング, 718
Mobile Link 統合データベースとしての IBM DB2 LUW, 12
Mobile Link 独立性レベル, 140
IBM DB2 LUW
Mobile Link 統合データベース, 12
IBM DB2 LUW 統合データベース
Mobile Link, 12
IBM DB2 LUW 統合データベースの設定
Mobile Link, 12
IdentifySimulatedClient コールバック [mlreplaycallbacks.cpp]
説明, 642
identity_password プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
identity オプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
identity プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
ignored_deletes プロパティ
Mobile Link プロファイラの同期統計情報, 208
ignored_inserts プロパティ
Mobile Link プロファイラの同期統計情報, 208
ignored_updates プロパティ
Mobile Link プロファイラの同期統計情報, 208
ignore プロトコルオプション
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 92
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
IndexOf メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 575
InfoListener イベント
ServerContext インタフェース [Mobile Link サーバ .NET API], 603
INFO 変数
LogMessage クラス [Mobile Link サーバ Java API], 495
InOutInteger インタフェース [Mobile Link サーバ Java API]
getValue メソッド, 489
setValue メソッド, 489
説明, 488
InOutString インタフェース [Mobile Link サーバ Java API]
getValue メソッド, 491
setValue メソッド, 491
説明, 490
Insert メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 576
is_on_ac_power
リモートタスク, 187
IsEndToEndEncrypted プロパティ

NetworkData インタフェース [MobiLink サーバ .NET API], 596
isEndToEndEncrypted メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 500
IsFixedSize プロパティ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 578
IsHTTP プロパティ
NetworkData インタフェース [MobiLink サーバ .NET API], 597
isHTTP メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 501
IsNullable プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 569
IsReadOnly プロパティ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 578
IsSynchronized プロパティ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 578
IsTLS プロパティ
NetworkData インタフェース [MobiLink サーバ .NET API], 597
isTLS メソッド
NetworkData インタフェース [MobiLink サーバ Java API], 501

J

Java
Mobile Link オブジェクトベースデータフロー, 623
Mobile Link サーバ API リファレンス, 473
Mobile Link データ型, 462
Mobile Link 同期スクリプト, 459
Javadoc
Mobile Link, 473
Java Mobile Link サーバ API (*参照* Java 用 Mobile Link サーバ API)
Java VM
Mobile Link オプション, 78
Java クラス
Java 同期論理のためのインスタンス化, 462
Java クラスのデバッグ
Mobile Link Java 同期論理, 464
Java での Mobile Link サーバエラーの処理
Mobile Link Java 同期論理, 465
Java と .NET 用の Mobile Link オブジェクトベースデータフロー
説明, 623
Java 同期
Mobile Link、Java 同期論理, 468
Java 同期スクリプトの作成
Mobile Link Java 同期論理の例, 469
Java 同期論理
32 ビットの UNIX での配備, 771
32 ビットの Windows での配備, 766
64 ビットの UNIX での配備, 774
64 ビットの Windows での配備, 768
Java クラスのインスタンス化, 462
Mobile Link サーバ API, 473
Mobile Link サーバのコマンドライン指定, 459
Mobile Link のパフォーマンス, 147
設定, 459, 464
例, 468
Java 同期論理と SQL 同期論理
Mobile Link のパフォーマンス, 147
Java 同期論理の実行
説明, 459

K

keep partial download 同期パラメータ
再起動可能なダウンロード, 136

L

last_download
Mobile Link 名前付きパラメータ, 109
modify_last_download_timestamp 接続イベント, 391
last_download_timestamp
Mobile Link でのせいせい, 110
Mobile Link 名前付きパラメータ, 109
last_table_download
Mobile Link 名前付きパラメータ, 109
modify_last_download_timestamp 接続イベント, 391
LDAP のシステムプロシージャ
Mobile Link サーバ, 645
Linux
dbmlsync の配備, 778
Listener ユーティリティ (dblsn)

Windows での Mobile Link クライアントの配備, 776
log_bad_request オプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95
Mobile Link OE 用 mlsrv16 -x オプション, 100
LogCallback デリゲート [Mobile Link サーバ .NET API]
説明, 619
LogListener インタフェース [Mobile Link サーバ Java API]
messageLogged メソッド, 492
説明, 491
LogMessage クラス [Mobile Link サーバ .NET API]
LogMessage コンストラクタ, 591
MessageType 列挙体, 591
Text プロパティ, 592
Type プロパティ, 592
User プロパティ, 592
説明, 590
LogMessage クラス [Mobile Link サーバ Java API]
ERROR 変数, 494
getText メソッド, 493
getType メソッド, 493
getUser メソッド, 493
INFO 変数, 495
WARNING 変数, 496
説明, 492
LogMessage コンストラクタ
LogMessage クラス [Mobile Link サーバ .NET API], 591
LONG データ型
Oracle の同期, 747
LUW
Mobile Link 統合データベースとしての IBM DB2 LUW, 12

M

Mac OS X
dbmlsync の配備, 778
makeConnection メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 507
MakeConnection メソッド
ServerContext インタフェース [Mobile Link サーバ .NET API], 603
Manage Anywhere
Mobile Link ファイルベースのダウンロード, 212
messageLogged メソッド
LogListener インタフェース, 492
MessageType 列挙体
LogMessage クラス [Mobile Link サーバ .NET API], 591
Microsoft ActiveSync
Windows での Mobile Link クライアントの配備, 776
Microsoft Excel
Mobile Link との同期, 623
Microsoft SQL Server
Mobile Link データマッピング, 727
Mobile Link 統合データベースとして使用, 14
Mobile Link 独立性レベル, 140
Microsoft SQL Server 統合データベース
Mobile Link, 14
Microsoft 分散トランザクションコーディネーター
Oracle ドライバのオプション, 761
Microsoft 分散トランザクションの有効化
Oracle ドライバのオプション, 761
Millisecond プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 555
Minute プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 555
ml_add_certificates_file システムプロシージャ
構文, 646
ml_add_column システムプロシージャ
構文, 646
ml_add_connection_script システムプロシージャ
構文, 648
ml_add_dnet_connection_script システムプロシージャ
構文, 649
ml_add_dnet_table_script システムプロシージャ
構文, 650
ml_add_java_connection_script システムプロシージャ
構文, 651
ml_add_java_table_script システムプロシージャ
構文, 652
ml_add_lang_connection_script_chk システムプロシージャ

構文, 653
ml_add_lang_connection_script システムプロシージャ
構文, 653
ml_add_lang_table_script_chk システムプロシージャ
構文, 654
ml_add_lang_table_script システムプロシージャ
構文, 654
ml_add_lcs_chk システムプロシージャ
構文, 653
ml_add_lcs システムプロシージャ
構文, 653
ml_add_ldap_server システムプロシージャ
構文, 654
ml_add_lts_chk システムプロシージャ
構文, 654
ml_add_lts システムプロシージャ
構文, 654
ml_add_missing_dnld_scripts システムプロシージャ
構文, 655
ml_add_passthrough_repair システムプロシージャ
構文, 657
ml_add_passthrough_script システムプロシージャ
構文, 658
ml_add_passthrough システムプロシージャ
構文, 656
ml_add_property システムプロシージャ
構文, 660
ml_add_table_script システムプロシージャ
構文, 664
ml_add_user_auth_policy システムプロシージャ
構文, 665
ml_add_user システムプロシージャ
構文, 665
ml_delete_passthrough_repair システムプロシージャ
構文, 667
ml_delete_passthrough_script システムプロシージャ
構文, 668
ml_delete_passthrough システムプロシージャ
構文, 667
ml_delete_sync_state_before システムプロシージャ
構文, 670
ml_delete_sync_state システムプロシージャ
構文, 668
ml_delete_user システムプロシージャ
構文, 670
ml_global スクリプトバージョン
説明, 253
ml_model_check_all_schema システムプロシージャ
構文, 671
ml_model_check_version_schema システムプロシージャ
構文, 673
ml_model_drop システムプロシージャ
構文, 671
ml_password
リモートタスク, 187
ml_ra_add_agent_id システムプロシージャ
構文, 674
ml_ra_assign_task システムプロシージャ
構文, 674
ml_ra_cancel_notification システムプロシージャ
構文, 675
ml_ra_cancel_task_instance システムプロシージャ
構文, 676
ml_ra_clone_agent_properties システムプロシージャ
構文, 676
ml_ra_delete_agent_id システムプロシージャ
構文, 677
ml_ra_delete_events_before システムプロシージャ
構文, 677
ml_ra_delete_remote_id システムプロシージャ
構文, 678
ml_ra_delete_task システムプロシージャ
構文, 678
ml_ra_get_agent_events システムプロシージャ
構文, 679
ml_ra_get_agent_ids システムプロシージャ
構文, 682
ml_ra_get_agent_properties システムプロシージャ
構文, 682
ml_ra_get_latest_event_id システムプロシージャ
構文, 683
ml_ra_get_orphan_taskdbs システムプロシージャ
構文, 683
ml_ra_get_remote_ids システムプロシージャ
構文, 684
ml_ra_get_task_results システムプロシージャ
構文, 685

ml_ra_get_task_status システムプロシージャ
構文, 686
ml_ra_manage_remote_db システムプロシージャ
構文, 688
ml_ra_notify_agent_sync システムプロシージャ
構文, 688
ml_ra_notify_task システムプロシージャ
構文, 689
ml_ra_reassign_taskdb システムプロシージャ
構文, 689
ml_ra_set_agent_property システムプロシージャ
構文, 690
ml_ra_unmanage_remote_db システムプロシージャ
構文, 690
ml_reset_sync_state システムプロシージャ
構文, 691
ml_server_delete システムプロシージャ
構文, 692
ml_server_update システムプロシージャ
構文, 692
ml_stream
リモートタスク, 187
ml_user
Mobile Link ユーザ認証ユーティリティ (mluser), 694
ml_username
リモートタスク, 187
mlagent コマンド
実行, 157
説明, 157
mlagent
構文, 158
mlarbiter ユーティリティ
UNIX, 706
Windows, 705
mlarbstop
Mobile Link 監視停止ユーティリティ, 707
mlastop コマンド
説明, 163
mlDomConfig.xml
説明, 545
mlgenreplayapi ユーティリティ
オプション, 704
構文, 704
mlprof
Mobile Link プロファイラの説明, 191
オプション, 193
起動, 193
mlreplaycallbacks.cpp 参照
CreateAndInitMLReplayUploadTransaction コールバック, 635
DelayCreationOfSimulatedClient コールバック, 636
DelayDestructionOfSimulatedClient コールバック, 636
DelayStartOfReplay コールバック, 637
FiniIdentifySimulatedClient コールバック, 638
GetDownloadApplyTime コールバック, 639
GetMLReplayAPIVersion コールバック, 640
GetUploadTransaction コールバック, 640
GlobalFini コールバック, 641
GlobalInit コールバック, 642
IdentifySimulatedClient コールバック, 642
ReportEndOfReplayy コールバック, 643
mlreplaycallbacks.cpp リファレンス
DestroyMLReplayUploadTransaction コールバック, 638
Mobile Link Replay C++ コールバック, 635
mlreplay ユーティリティ
オプション, 697
構文, 696
mlscript.jar
Mobile Link の Java 同期論理, 459
mlsrv16
(*参照* Mobile Link サーバ)
Notifier, 65
アルファベット順のオプションリスト, 43
オプション, 43
起動, 29
構文, 43
接続文字列, 52
説明, 43
停止, 31
メッセージログにエラーコンテキストをレポート, 65
ロギング, 31
mlstop ユーティリティ
32 ビットの UNIX での配備, 771
32 ビットの Windows での配備, 766
64 ビットの UNIX での配備, 774
64 ビットの Windows での配備, 768
Mobile Link サーバの停止方法, 31
オプション, 693
構文, 692
mluser ユーティリティ

32 ビットの UNIX での配備, 771
32 ビットの Windows での配備, 766
64 ビットの UNIX での配備, 774
64 ビットの Windows での配備, 768
オプション, 694
構文, 694
Mobile Link
.NET 同期論理, 533
Java 同期論理, 459
mlsrv16 オプション, 43
mlsrv16 の接続パラメータ, 92
Mobile Link サーバの停止, 31
ODBC ドライバのサポート, 760
Ultra Light クライアントの配備, 779
アプリケーションの配備, 765
イベントのアルファベット順リスト, 269
イベントの概要, 269
開発のヒント, 107
起動, 29
競合の解決, 124
現在のセッション外での Mobile Link の実行, 33
システムテーブル, 4
システムプロシージャ, 644
スクリプト, 229
データ型, 708
同期サーバの実行, 29
同期の方法, 107
統合データベース, 1
パフォーマンス, 142
ファイルベースのダウンロード, 212
プロファイラ, 191
プロファイラの接続パラメータ, 193
文字セットの考慮事項, 758
Mobile Link Listener ユーティリティ (dblsn)
Windows での Mobile Link クライアントの配備, 776
Mobile Link Replay C++ コールバック
mlreplaycallbacks.cpp リファレンス, 635
Mobile Link アプリケーションの配備
説明, 765
Mobile Link イベント
リスト, 269
Mobile Link イベントの順序
擬似コード, 274
Mobile Link エージェント
Sybase Central, 164
Ultra Light クライアントを Windows で展開, 779
Windows での SQL Anywhere クライアントの配備, 776
実行, 157
設定, 157
設定ウィンドウ, 161
説明, 156
定義, 153
停止, 163
認証, 168
プロパティ, 166
リモートデータベースの追加, 167
Mobile Link エージェント ID
定義, 154
Mobile Link エージェントプロパティ
Mobile Link クライアントのネットワークプロトコルオプション, 166
接続文字列, 166
同期間隔, 166
表示と変更, 166
Mobile Link 監視サーバユーティリティ
UNIX, 706
Windows, 705
Mobile Link クライアント
配備, 776
Mobile Link グループ
追加, 167
Mobile Link サーバ
(*参照* mlsrv16)
Mobile Link 停止ユーティリティ (mlstop), 692
Mobile Link リプレイユーティリティ (mlreplay), 696
オプション, 43
起動, 29
構文, 43
実行, 29
生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi), 704
配備, 765
Mobile Link サーバ .NET API
DateTimeWithTimeZone クラス, 549
DBCommand インタフェース, 556
DBConnectionContext インタフェース, 561
DBConnection インタフェース, 559
DBParameterCollection クラス, 571
DBParameter クラス, 567
DBRowReader インタフェース, 580

DownloadData インタフェース, 582
DownloadTableData インタフェース, 584
FatalException クラス, 589
LogCallback デリゲート, 619
LogMessage クラス, 590
NetworkData インタフェース, 593
ScriptExecutionException クラス, 597
ServerContext インタフェース, 599
ServerException クラス, 604
ShutdownCallback デリゲート, 620
SpatialUtilities クラス, 606
SQLType 列挙体, 620
SynchronizationException クラス, 608
UpdateDataReader インタフェース, 610
UploadData インタフェース, 612
UploadedTableData インタフェース, 614
Mobile Link サーバ .NET API リファレンス
iAnywhere.MobiLink.Script ネームスペース, 548
Mobile Link サーバ Java API
DBConnectionContext インタフェース, 473
DownloadData インタフェース, 478
DownloadTableData インタフェース, 481
InOutInteger インタフェース, 488
InOutString インタフェース, 490
LogListener インタフェース, 491
LogMessage クラス, 492
NetworkData インタフェース, 497
ServerContext インタフェース, 501
ServerException クラス, 510
ShutdownListener インタフェース, 511
SpatialUtilities クラス, 513
SynchronizationException クラス, 515
TimestampWithTimeZone クラス, 516
UpdateResultSet インタフェース, 522
UploadData インタフェース, 524
UploadedTableData インタフェース, 526
Mobile Link サーバ Java API リファレンス
ianywhere.ml.script パッケージ, 473
Mobile Link サーバ共有ステータス
サーバファーム, 39
Mobile Link サーバ動作のロギング
説明, 31
Mobile Link サーバのオプション
説明, 43
Mobile Link サーバの実行
サーバファーム内, 39
サービスとして実行, 34
説明, 29
Mobile Link サーバの配備
説明, 765
Mobile Link サーバファーム
Mobile Link mlsrv16 -lsc オプション, 63
フェイルオーバ, 39
負荷分散, 39
Mobile Link ユーティリティ
生成された Mobile Link リプレイ API (mlgenreplayapi) の構文, 704
Mobile Link サーバログファイルビューア
Mobile Link サーバログ, 32
Mobile Link システムデータベース
説明, 4
Mobile Link システムデータベース (MLSD)
mlsrv16 -cs オプション, 56
Mobile Link システムテーブル
説明, 4
統合データベースに作成, 3
Mobile Link システムプロシージャ
説明, 644
Mobile Link スクリプト
リスト, 269
Mobile Link ストアドプロシージャ (*参照* Mobile Link システムプロシージャ)
Mobile Link 接続
HTTP および HTTPS での検証, 30, 42
デバッグ, 42
Mobile Link ダイレクトの競合の処理
.NET の例, 631
Java の例, 630
Mobile Link 停止ユーティリティ (mlstop)
構文, 692
Mobile Link データ型
.NET と SQL, 537
Java と SQL, 462
Mobile Link データマッピング
説明, 708
Mobile Link 同期
.NET クラスの作成, 538
.NET 同期論理, 533
Java クラスの作成, 464
Java 同期論理, 459
イベントの概要, 269
再起動可能なダウンロード, 134
統合データベース, 1
パフォーマンス, 142
ファイルベースのダウンロード, 212

Mobile Link 同期サーバ (*参照* Mobile Link サーバ)
Mobile Link 同期スクリプト
.NET クラスの構成, 537
.NET クラスの作成, 538
.NET でのデータベーストランザクションの保存, 537
Java クラスの構成, 463
Java クラスの作成, 464
Java クラスのデバッグ, 464
Java でのデータベーストランザクションの保存, 462
スクリプトのアルファベット順リスト, 269
説明, 229
データベーストランザクションと .NET クラス, 537
データベーストランザクションと Java クラス, 462
Mobile Link 同期論理
.NET, 533
.NET と SQL のデータ型, 537
Java, 459
Java と SQL に対応するデータ型, 462
スクリプトのアルファベット順リスト, 269
スクリプトの作成, 229
同期の方法, 107
Mobile Link 統合データベース
ASE, 10
IBM DB2 LUW, 12
MySQL, 16
Oracle, 19
SAP HANA, 24
SQL Anywhere, 26
SQL Server, 14
説明, 1
としての SAP Sybase IQ, 27
Mobile Link の実行
現在のセッション外, 33
説明, 29
デーモンとして実行, 33
Mobile Link の世代番号
ファイルベースのダウンロード, 219
Mobile Link の統計のプロパティ
Mobile Link プロファイラ, 208
Mobile Link のパフォーマンス
主要な要因, 148
説明, 142
モニタ, 151
Mobile Link のパフォーマンスのモニタ
概要, 151
Mobile Link のローハンドリング (*参照* ダイレクトローハンドリング)
Mobile Link ファイル転送ユーティリティ (mlfiletransfer)
Mobile Link mlsrv16 -ftru オプション, 63
Mobile Link mlsrv16 -ftr オプション, 62
Mobile Link ファーム (*参照* Mobile Link サーバファーム)
Mobile Link プロジェクト
定義, 153
Mobile Link プロファイラ
32 ビットの UNIX での配備, 771
32 ビットの Windows での配備, 766
64 ビットの UNIX での配備, 774
64 ビットの Windows での配備, 768
新しいウォッチ, 205
以前のセッションの削除, 195
以前のセッションを開く, 195
ウォッチの指定, 204
ウォッチマネージャ, 204
オプション, 193, 203
概要ウィンドウ枠, 201
起動, 192, 193
グラフウィンドウ枠, 199
グラフ画面の使用, 200
サンプルプロパティ, 203
終了, 195
使用, 197
[詳細テーブル] ウィンドウ枠, 197
ズーム, 200
セッションプロパティ, 203
説明, 191
[チャート] ウィンドウ枠, 200
データベース, 196
データベースビュー, 205
統計プロパティ, 208
マーキーツール, 201
ユーザインタフェース, 197
Mobile Link プロファイラの起動
説明, 192
Mobile Link ユーザ
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link ユーザ認証ユーティリティ (mluser) を使用した登録, 694
Mobile Link ユーザ認証ユーティリティ (mluser)

構文, 694
Mobile Link ユーザの登録
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link ユーティリティ
Mobile Link arbiter の構文、Windows, 705
Mobile Link 監視停止の構文、Windows, 707
Mobile Link 監視ユーティリティの構文、UNIX, 706
Mobile Link 停止 (mlstop) 構文, 692
Mobile Link ユーザ認証 (mluser) 構文, 694
Mobile Link リプレイ (mlreplay) 構文, 696
サーバ, 692
説明, 692
Mobile Link リプレイユーティリティ (mlreplay)
構文, 696
Mobile Link ログの表示
説明, 32
Mobile Link ログファイルビューア
Mobile Link サーバログ, 32
Mobile デバイスセンター (*参照* Microsoft ActiveSync)
modify_error_message
接続イベント, 389
modify_last_download_timestamp
接続イベント, 391
modify_next_last_download_timestamp
接続イベント, 394
modify_user
接続イベント, 397
Month プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 555
MySQL
Mobile Link データマッピング, 734
Mobile Link 統合データベース, 16
MySQL 統合データベース
Mobile Link, 16
MySQL 統合データベースの設定
Mobile Link, 16

N

NetworkData インタフェース [Mobile Link サーバ .NET API]
ClientCertificates プロパティ, 595
GetHTTPHeaderValues メソッド, 595
GetHTTPHeaderValue メソッド, 595
HTTPHeaders プロパティ, 596
IsEndToEndEncrypted プロパティ, 596
IsHTTP プロパティ, 597
IsTLS property プロパティ, 597
説明, 593
NetworkData インタフェース [Mobile Link サーバ Java API]
getCertificateChain メソッド, 499
getHTTPHeaders メソッド, 499
getHTTPHeaderValues メソッド, 500
getHTTPHeaderValue メソッド, 500
isEndToEndEncrypted メソッド, 500
isHTTP メソッド, 501
isTLS メソッド, 501
説明, 497
NetworkData プロパティ
DBConnectionContext インタフェース [Mobile Link サーバ .NET API], 566
NextRow メソッド
DBRowReader インタフェース [Mobile Link サーバ .NET API], 581
nonblocking_download_ack
接続イベント, 400
Notifier
32 ビットの UNIX での配備, 771
32 ビットの Windows での配備, 766
64 ビットの UNIX での配備, 774
64 ビットの Windows での配備, 768
起動, 65

O

o.
Mobile Link 名前付きパラメータのプレフィクス, 235
ODBC
Mobile Link での複数のエラー, 267
Mobile Link ドライバ, 760
Oracle ドライバ, 761
ODBC ドライバ
Mobile Link のサポート, 760
Mobile Link 文字セット変換, 760
Oracle, 761
OE
Mobile Link mlsrv16 -x オプション, 100
Oracle
Mobile Link データマッピング, 740
Mobile Link 同期でのシーケンス, 21

Mobile Link 統合データベースとして使用, 19
Mobile Link 独立性レベル, 140
ODBC ドライバ, 761
XMLTYPE データ型, 22
空間データの同期, 7
同期する LONG データ, 747
Oracle varray
ストアドプロシージャでの使用, 23
制限, 24
例, 23
Oracle 統合データベース
Mobile Link, 19
Oracle 統合データベースの設定
Mobile Link, 19
Oracle ドライバ
ODBC, 761
パスワードの暗号化, 761

P

ParameterName プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 569
Parameters プロパティ
DBCommand インタフェース [Mobile Link サーバ .NET API], 559
Parse メソッド
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 552
partial download retained 同期パラメータ
再起動可能なダウンロード, 136
port プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 92
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
Precision プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 569
prepare_for_download
接続イベント, 403
prepare_for_download プロパティ
Mobile Link プロファイラの同期統計情報, 208
Prepare メソッド
DBCommand インタフェース [Mobile Link サーバ .NET API], 558
ProcResults
Oracle ドライバのオプション, 761
publication_nonblocking_download_ack
接続イベント, 406

R

r.
Mobile Link 名前付きパラメータのプレフィクス, 235
READPAST テーブルヒント
download_cursor に関する問題, 330
download_delete_cursor に関する問題, 333
upload_fetch に関する問題, 436
relayserver (*参照* Relay Server)
Relay Server Outbound Enabler
(*参照* Outbound Enabler)
Relay Server のホスティングサービス
(*参照* Sybase ホストのリレーサービス)
Relay Server のホスト
(*参照* Sybase ホストのリレーサービス)
remote_id
リモートタスク, 187
RemoveAt メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 577
removeErrorListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 508
removeInfoListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 508
removeShutdownListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 509
removeWarningListener メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 509
Remove メソッド
DBParameterCollection クラス [Mobile Link サーバ .NET API], 576
report_error
構文, 268
接続イベント, 409
report_odbc_error
接続イベント, 413

ReportEndOfReplay コールバック [mlreplaycallbacks.cpp]
説明, 643
resolve_conflict
使用, 128
テーブルイベント, 417
resume partial download 同期パラメータ
再起動可能なダウンロード, 136
Rollback メソッド
DBConnection インタフェース [Mobile Link サーバ .NET API], 561
rsa プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TCP/IP 用 mlsrv16 -x オプション, 93
rshost (*参照* Relay Server ステイトマネージャ)
rsoe (*参照* Outbound Enabler)
rtnotifier
構文, 179

S

s.
Mobile Link 名前付きパラメータのプレフィクス, 235
SAP HANA
Mobile Link 同期, 25
Mobile Link 統合データベース, 24
SAP HANA 統合データベースの設定
Mobile Link, 24
SAP Sybase IQ
Mobile Link 統合データベースとしての, 27
SAP Sybase IQ 統合データベース
Mobile Link, 27
SAP Sybase IQ 統合データベースの設定
Mobile Link, 27
Scale プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 570
ScriptExecutionException クラス [Mobile Link サーバ .NET API]
ScriptExecutionException コンストラクタ, 598
説明, 597
ScriptExecutionException コンストラクタ
ScriptExecutionException クラス [Mobile Link サーバ .NET API], 598
Second プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 555
ServerContext インタフェース [Mobile Link サーバ .NET API]
ErrorListener イベント, 603
getPropertiesByVersion メソッド, 601
getProperties メソッド, 600
getPropertySetNames メソッド, 601
GetStartClassInstances メソッド, 602
InfoListener イベント, 603
MakeConnection メソッド, 603
ShutdownListener イベント, 604
Shutdown メソッド, 603
WarningListener イベント, 604
説明, 599
ServerContext インタフェース [Mobile Link サーバ Java API]
addErrorListener メソッド, 503
addInfoListener メソッド, 503
addShutdownListener メソッド, 504
addWarningListener メソッド, 505
getPropertiesByVersion メソッド, 506
getProperties メソッド, 505
getPropertySetNames メソッド, 506
getStartClassInstances メソッド, 507
makeConnection メソッド, 507
removeErrorListener メソッド, 508
removeInfoListener メソッド, 508
removeShutdownListener メソッド, 509
removeWarningListener メソッド, 509
shutdown メソッド, 509
説明, 501
ServerException クラス [Mobile Link サーバ .NET API]
ServerException コンストラクタ, 605
説明, 604
ServerException クラス [Mobile Link サーバ Java API]
ServerException コンストラクタ, 511
説明, 510
ServerException コンストラクタ
ServerException クラス [Mobile Link サーバ .NET API], 605
ServerException クラス [Mobile Link サーバ Java API], 511
ServiceName
Oracle ドライバのオプション, 761
setNewRowValues メソッド

UpdateResultSet インタフェース [Mobile Link サーバ Java API], 523
SetNewRowValues メソッド
UpdateDataReader インタフェース [Mobile Link サーバ .NET API], 611
SET NOCOUNT
SQL Server Mobile Link 統合データベース, 15
setOldRowValues メソッド
UpdateResultSet インタフェース [Mobile Link サーバ Java API], 523
SetOldRowValues メソッド
UpdateDataReader インタフェース [Mobile Link サーバ .NET API], 612
setSRID メソッド
SpatialUtilities クラス [Mobile Link サーバ Java API], 514
SetSRID メソッド
SpatialUtilities クラス [Mobile Link サーバ .NET API], 608
setTimeZoneOffsetHours メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 520
setTimeZoneOffsetMinutes メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 521
setValue メソッド
InOutInteger インタフェース [Mobile Link サーバ Java API], 489
InOutString インタフェース [Mobile Link サーバ Java API], 491
ShutdownCallback デリゲート [Mobile Link サーバ .NET API]
説明, 620
ShutdownListener イベント
ServerContext インタフェース [Mobile Link サーバ .NET API], 604
ShutdownListener インタフェース [Mobile Link サーバ Java API]
shutdownPerformed メソッド, 512
説明, 511
shutdownPerformed メソッド
ShutdownListener インタフェース [Mobile Link サーバ Java API], 512
shutdown メソッド
ServerContext インタフェース [Mobile Link サーバ Java API], 509
Shutdown メソッド
ServerContext インタフェース [Mobile Link サーバ .NET API], 603
SIRT (*参照* サーバ起動リモートタスク)
サーバ起動リモートタスク, 179
展開済みリモートタスクの開始, 177
sis (*参照* サーバ起動同期)
Size プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 570
SpatialUtilities クラス [Mobile Link サーバ .NET API]
CreateSpatialValue メソッド, 607
GetBytes メソッド, 607
GetSRID メソッド, 607
SetSRID メソッド, 608
説明, 606
SpatialUtilities クラス [Mobile Link サーバ Java API]
createSpatialValue メソッド, 513
getBytes メソッド, 514
getSRID メソッド, 514
setSRID メソッド, 514
説明, 513
SQL_TXN_READ_COMMITTED
Mobile Link 独立性レベル, 140
SQL Anywhere
Mobile Link 統合データベースとして使用, 26
Mobile Link 独立性レベル, 140
SQL Anywhere 16 - Oracle ODBC ドライバ
UNIX の設定, 764
Windows の設定, 763
説明, 761
SQL Anywhere 16 - Oracle ODBC ドライバ
Mobile Link さぽーと, 760
SQL Anywhere Mobile Link クライアントの配備
説明, 776
SQL Anywhere 統合データベース
Mobile Link, 26
SQL Anywhere 統合データベースの設定
Mobile Link, 26
SQL Server
(*参照* Microsoft SQL Server)
Mobile Link データマッピング, 727
Mobile Link 統合データベースとして使用, 14
SQLType 列挙体 [Mobile Link サーバ .NET API]
説明, 620
SQL 構文
Mobile Link サーバ (mlsrv16), 43

SQL データ型と .NET データ型
Mobile Link .NET 同期論理, 537
SQL データ型と Java データ型
説明, 462
SQL 同期論理
Mobile Link, 229
Mobile Link のパフォーマンス, 147
SRID
空間データの同期, 7
start_time プロパティ
Mobile Link プロファイラの同期統計情報, 208
StaticCursorLongColBuffLen
ASE, 11
STOP SYNCHRONIZATION DELETE 文
SQL Anywhere クライアント, 133
使用方法, 265
Sybase Adaptive Server Enterprise (*参照* Adaptive Server Enterprise)
Sybase ASE 統合データベースの設定
Mobile Link, 10
SAP HANA
Mobile Link のデータマッピング, 751, 755
sync_deadlocks プロパティ
Mobile Link プロファイラの同期統計情報, 208
sync_errors プロパティ
Mobile Link プロファイラの同期統計情報, 208
sync_request プロパティ
Mobile Link プロファイラの同期統計情報, 208
sync_tables プロパティ
Mobile Link プロファイラの同期統計情報, 208
sync_warnings プロパティ
Mobile Link プロファイラの同期統計情報, 208
syncase.sql
説明, 10
syncdb2long.sql
説明, 12
synchana.sql
説明, 24
synchronization_statistics
接続イベント, 421
テーブルイベント, 424
SynchronizationException クラス [Mobile Link サーバ .NET API]
SynchronizationException コンストラクタ, 609
説明, 608
SynchronizationException クラス [Mobile Link サーバ Java API]
SynchronizationException コンストラクタ, 516
説明, 515
SynchronizationException コンストラクタ
SynchronizationException クラス [Mobile Link サーバ .NET API], 609
SynchronizationException クラス [Mobile Link サーバ Java API], 516
syncmss.sql
説明, 14
syncora.sql
説明, 19
SyncRoot プロパティ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 579
syncsa.sql
説明, 26
sync プロパティ
Mobile Link プロファイラの同期統計情報, 208

T

TCP/IP
Mobile Link mlsrv16 -x オプション, 92
Text プロパティ
LogMessage クラス [Mobile Link サーバ .NET API], 592
this プロパティ
DBParameterCollection クラス [Mobile Link サーバ .NET API], 579
time_statistics
接続イベント, 427
テーブルイベント, 430
TimestampWithTimeZone クラス [Mobile Link サーバ Java API]
equals メソッド, 518
getTimeZoneOffsetHours メソッド, 520
getTimeZoneOffsetMinutes メソッド, 520
setTimeZoneOffsetHours メソッド, 520
setTimeZoneOffsetMinutes メソッド, 521
TimestampWithTimeZone コンストラクタ, 517
toString メソッド, 521
toTimestampWithTimeZone メソッド, 521
valueOf メソッド, 521
説明, 516
TimestampWithTimeZone コンストラクタ
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 517
TimeZoneHour プロパティ

DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 555
TimeZoneMinute プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 556
TLS
(*参照* トランスポートレイヤセキュリティ)
32 ビットの UNIX での Mobile Link サーバの配備, 771
32 ビットの Windows での Mobile Link サーバ配備, 766
64 ビットの UNIX での Mobile Link サーバの配備, 774
64 ビットの Windows での Mobile Link サーバの配備, 768
Mobile Link mlsrv16 -x オプション, 93
UNIX での Mobile Link クライアントの配備, 778
Windows での Mobile Link クライアントの配備, 776
toString メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 521
ToString メソッド
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 552
toTimestampWithTimeZone メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 521
trusted_certificates
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link TLS over TCP/IP 用 mlsrv16 -x オプション, 93
Type プロパティ
LogMessage クラス [Mobile Link サーバ .NET API], 592

U

u.
Mobile Link ユーザ定義のパラメータのプレフィクス, 250
ULRollbackPartialDownload 関数
再起動可能なダウンロード, 136
Ultra Light
配備, 779
Ultra Light Mobile Link クライアントの配備
説明, 779
UNIX
dbmlsync の配備, 778
デーモンとしての Mobile Link サーバ, 33
UpdateDataReader インタフェース [Mobile Link サーバ .NET API]
SetNewRowValues メソッド, 611
SetOldRowValues メソッド, 612
説明, 610
UpdateResultSet インタフェース [Mobile Link サーバ Java API]
setNewRowValues メソッド, 523
setOldRowValues メソッド, 523
説明, 522
UPDATE の競合
Mobile Link, 124
upload_bytes プロパティ
Mobile Link プロファイラの同期統計情報, 208
upload_deadlocks プロパティ
Mobile Link プロファイラの同期統計情報, 208
upload_delete
テーブルイベント, 433
upload_deleted_rows プロパティ
Mobile Link プロファイラの同期統計情報, 208
upload_errors プロパティ
Mobile Link プロファイラの同期統計情報, 208
upload_fetch
競合検出の概要, 125
競合の検出, 126
テーブルイベント, 435
upload_fetch_column_conflict
競合検出の概要, 125
競合の検出, 126
テーブルイベント, 437
upload_insert
テーブルイベント, 439
upload_inserted_rows プロパティ
Mobile Link プロファイラの同期統計情報, 208
upload_new_row_insert
テーブルイベント, 441
upload_old_row_insert
テーブルイベント, 443
upload_statistics
接続イベント, 445
テーブルイベント, 450
upload_update
競合検出の概要, 125
競合の検出, 128

使用, 130
テーブルイベント, 455
upload_updated_rows プロパティ
Mobile Link プロファイラの同期統計情報, 208
upload_warnings プロパティ
Mobile Link プロファイラの同期統計情報, 208
UploadData インタフェース [Mobile Link サーバ .NET API]
GetUploadedTableByName メソッド, 613
GetUploadedTables メソッド, 613
説明, 612
UploadData インタフェース [Mobile Link サーバ Java API]
getUploadedTableByName メソッド, 525
getUploadedTables メソッド, 525
説明, 524
UploadedTableData インタフェース [Mobile Link サーバ .NET API]
GetDeletes メソッド, 615
GetInserts メソッド, 616
GetName メソッド, 617
GetSchemaTable メソッド, 617
GetUpdates メソッド, 618
説明, 614
UploadedTableData インタフェース [Mobile Link サーバ Java API]
getDeletes メソッド, 528
getInserts メソッド, 529
getMetaData メソッド, 529
getName メソッド, 531
getUpdates メソッド, 531
説明, 526
upload プロパティ
Mobile Link プロファイラの同期統計情報, 208
user プロパティ
Mobile Link プロファイラの同期統計情報, 208
User プロパティ
LogMessage クラス [Mobile Link サーバ .NET API], 592
UTC TIMESTAMP
Mobile Link, 111
UUID
Mobile Link 同期アプリケーション, 118

V

valueOf メソッド
TimestampWithTimeZone クラス [Mobile Link サーバ Java API], 521
Value プロパティ
DBParameter クラス [Mobile Link サーバ .NET API], 570
VARBIT データ型
ASE Mobile Link 統合データベースでの制限, 11
VARCHAR データ型
Mobile Link と他の DBMS, 5
varray (Oracle)
ストアドプロシージャでの使用, 23
制限, 24
例, 23
VB (*参照* Visual Basic)
version オプション
Mobile Link OE 用 mlsrv16 -x オプション, 100
version プロトコルオプション
Mobile Link HTTPS 用 mlsrv16 -x オプション, 97
Mobile Link HTTP 用 mlsrv16 -x オプション, 95
version プロパティ
Mobile Link プロファイラの同期統計情報, 208
Visual Basic
Mobile Link .NET でのサポート, 533
Mobile Link 同期スクリプト, 533
Visual Studio
Mobile Link 同期スクリプト, 533

W

WarningListener イベント
ServerContext インタフェース [Mobile Link サーバ .NET API], 604
WARNING 変数
LogMessage クラス [Mobile Link サーバ Java API], 496
WebLogic
Mobile Link, 623
Web サーバ
Mobile Link との同期, 623
Web サービス
Mobile Link との同期, 623
Windows CE (*参照* Windows Mobile)
Windows Mobile デバイスセンター (*参照* Microsoft ActiveSync)

X

Xusage.txt
ロケーション, 78

Y

Year プロパティ
DateTimeWithTimeZone クラス [Mobile Link サーバ .NET API], 556

あ

アセンブリ
Mobile Link .NET 同期論理での配置, 533
Mobile Link での実装, 544
アセンブリのロード
Mobile Link .NET 同期論理, 544
新しいリモートの同期
Mobile Link ファイルベースのダウンロード, 215
アップロード
Mobile Link 一時停止, 133
Mobile Link トランザクション, 271
Mobile Link ローをアップロードするスクリプト, 260
アップロードイベント
Mobile Link 同期, 277
説明, 260
アップロード専用の同期
説明, 116
必要なスクリプト, 255
アップロード専用の同期とダウンロード専用の同期
説明, 116
アップロード中のイベント
説明, 277
ローをアップロードするスクリプトの作成, 260
アップロードトランザクション
Mobile Link, 271
アップロードのみの同期
(*参照* アップロード専用の同期)
アプリケーション
Mobile Link アプリケーションの配備, 765
アプリケーション間メッセージング
(*参照* メッセージング)
アプリケーションサーバ
Mobile Link との同期, 623
アプリケーションのアップグレード
複数の Mobile Link スクリプトバージョンの使用, 252
アンチエイリアス処理
Mobile Link プロファイラオプション, 200

い

一貫性
(*参照* 同期)
イベント
Mobile Link, 269
Mobile Link イベントの説明, 230
Mobile Link ダイレクトローハンドリング, 624
Mobile Link 同期の説明, 269
イベントモデル
Mobile Link 擬似コード, 274
インクリメンタルアップロード
Mobile Link のパフォーマンス, 144
インデックス
Mobile Link のパフォーマンス, 146
引用符
IBM DB2 Mobile Link 統合データベース, 13

え

エラー
Mobile Link modify_error_message 接続イベント, 389
Mobile Link 同期中の処理, 267
記録, 268
エラー処理
Mobile Link 同期中, 267
エラーの処理
Mobile Link サーバ, 374
エラーのレポート
Mobile Link 同期, 268
エラーログ
Mobile Link サーバ オプション (mlsrv16), 60
エンドツーエンド暗号化のタイプ
Mobile Link サーバオプション (mlsrv16), 92
エージェント ID
説明, 154
エージェントデータベース
定義, 154

お

オブジェクト
Java 用 Mobile Link サーバ API, 473

オブジェクトベースデータフロー (*参照* ダイレクトローハンドリング)
オプション
mlsrv16, 43
Mobile Link サーバ (mlsrv16), 43
Mobile Link 停止ユーティリティ (mlstop), 692
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Mobile Link リプレイユーティリティ (mlreplay), 696
生成された Mobile Link リプレイ API ユーティリティ (mlgenreplayapi), 704
[オプション] ウィンドウ
Mobile Link プロファイラ, 203
オートインクリメントメソッド
Oracle Mobile Link 統合データベース, 20

か

解決
Mobile Link 競合, 124
Mobile Link 競合解決, 124
開発のヒント
Mobile Link ダイレクトローハンドリング, 626
Mobile Link 同期, 107
外部キーテーブルの分割
Mobile Link, 116
概要ウィンドウ枠
Mobile Link プロファイラ, 201
カスタム検証
Mobile Link ファイルベースのダウンロード, 219
可動性 (*参照* Mobile Link)
空の文字列
Oracle Mobile Link 統合データベース, 20
Oracle でのサポートなし, 21
カラムサイズ
ASE Mobile Link 統合データベース, 11
完全なイベントモデル
Mobile Link, 269
Mobile Link 擬似コード, 274
完全な同期 (*参照* 双方向同期)
管理対象のリモートデータベース
追加, 167
管理ポーリング間隔
Mobile Link エージェント, 166
カーソルスクリプト
定義, 233

き

企業データベース
Mobile Link との同期, 623
擬似コード
Mobile Link イベント, 269
起動
Mobile Link サーバ, 29
起動クラス
.NET の MLStartClasses オプション, 76
Java の DMLStartClasses オプション, 78
Mobile Link .NET 同期論理, 539
Mobile Link Java 同期論理, 467
基本的な規則
Mobile Link, 107
疑問符
Mobile Link スクリプトパラメータ, 234
競合
Mobile Link, 124
Mobile Link 競合解決, 124
Mobile Link ダイレクトローハンドリング, 629
Mobile Link での検出, 125
Mobile Link のデフォルト動作, 124
Mobile Link のパフォーマンス, 142
Mobile Link のパフォーマンスの説明, 149
競合解決
Mobile Link, 124
Mobile Link 競合検出, 125
Mobile Link での検出, 125
Mobile Link のデフォルト動作, 124
resolve_conflict スクリプト, 128
upload_update スクリプト, 130
競合検出
Mobile Link, 125
Mobile Link 文ベースのアップロード, 125
競合の解決
Mobile Link, 124
Mobile Link、resolve_conflict スクリプトを使用, 128
Mobile Link、upload_update スクリプトを使用, 130
Mobile Link の概要, 128
resolve_conflict スクリプト, 128
upload_update スクリプト, 130
競合の検出
Mobile Link, 125
Mobile Link、upload_fetch スクリプトを使用, 126

Mobile Link、upload_update スクリプトを使用, 128
競合の処理
Mobile Link ダイレクトローハンドリング, 629
競合を検出する方法
Mobile Link, 125
強制終了
Mobile Link サーバ, 31
共有アセンブリ
Mobile Link での実装, 544
共有規則 (*参照* 分割)
共有サーバステータス
Mobile Link mlsrv16 -zs オプション, 104
共有ステータス
Mobile Link サーバファーム, 39
キープール
Mobile Link 同期アプリケーション, 122

く

クイックスタート
Mobile Link ダイレクトローハンドリング, 625
空間データ
同期, 6
空間データの同期
Oracle の考慮事項, 7
SRID 要件, 7
アップロードスクリプトとダウンロードスクリプト, 8
制限, 6
名前付きパラメータ, 7
空間データを同期するためのスクリプト
IBM DB2 サンプル, 9
Microsoft SQL Server サンプル, 9
MySQL サンプル, 10
Oracle サンプル, 9
SQL Anywhere サンプル, 8
アップロードスクリプトとダウンロードスクリプト, 8
クライアントイベントフックプロシージャ (*参照* イベントフック)
クラスインスタンス
Java 同期論理, 462
Mobile Link .NET 同期論理, 536
グラフウィンドウ枠
Mobile Link プロファイラ, 199
グループ
Mobile Link, 167
グローバル
Mobile Link のスクリプトバージョン, 253
グローバルアセンブリキャッシュ
Mobile Link での実装, 544
グローバルオートインクリメント
Mobile Link での宣言, 120
Mobile Link のユニークなプライマリキー, 119
Mobile Link 用の global_database_id の設定, 120
アルゴリズム, 121
グローバルスクリプトバージョン
Mobile Link, 253

け

権限
Mobile Link サーバ, 30
言語ライブラリ
32 ビットの UNIX での Mobile Link サーバ配備, 771
32 ビットの Windows での Mobile Link サーバ配備, 766
64 ビットの UNIX での Mobile Link サーバ配備, 774
64 ビットの Windows での Mobile Link サーバ配備, 768
検証
Mobile Link、カスタム, 219
Mobile Link、自動的, 217
Mobile Link ファイルベースのダウンロード, 216
検証チェック
Mobile Link ファイルベースのダウンロード, 216

こ

構文
Mobile Link サーバ (mlsrv16), 43
Mobile Link システムプロシージャ, 644
Mobile Link スクリプト, 269
Mobile Link 停止ユーティリティ (mlstop), 692
Mobile Link 同期ユーティリティ, 692
Mobile Link ユーザ認証ユーティリティ (mluser), 694
コマンド
Mobile Link リモートタスク, 180
リモートタスクへの追加, 186
コマンドライン
mlsrv16 の起動, 43

Mobile Link サーバ (mlsrv16), 43
コマンドラインユーティリティ
Mobile Link 停止 (mlstop) 構文, 692
Mobile Link 同期, 692
Mobile Link ユーザ認証 (mluser) 構文, 694
Mobile Link リプレイ (mlreplay) 構文, 696
生成された Mobile Link リプレイ API (mlgenreplayapi), 704
コンストラクタ
Mobile Link .NET 同期論理, 537
Mobile Link Java 同期論理, 463

さ

再起動可能なダウンロード
Mobile Link, 134
最終ダウンロード時刻
Mobile Link の説明, 109
最終ダウンロード時刻の変更
Mobile Link, 111
最終ダウンロードタイムスタンプ
handle_DownloadData 接続イベント, 368
Mobile Link での生成, 110
Mobile Link の説明, 109
modify_last_download_timestamp 接続イベント, 391
modify_next_last_download_timestamp 接続イベント, 394
最終変更カラム
Mobile Link, 108
削除
Mobile Link .NET 接続スクリプト, 649
Mobile Link .NET テーブルスクリプト, 650
Mobile Link Java 接続スクリプト, 651
Mobile Link Java テーブルスクリプト, 652
Mobile Link SQL 接続スクリプト, 648
Mobile Link SQL テーブルスクリプト, 664
Mobile Link ダウンロード, 265
Mobile Link のプロパティ, 660
Mobile Link リモートデータベースのロー, 265
SQL Anywhere クライアントのアップロードの停止, 133
削除の処理
Mobile Link, 132
削除のダウンロード
Mobile Link download_delete_cursor スクリプト, 265
作成
.NET 同期論理, 533
Java 同期論理, 459
Mobile Link 統合データベース, 3
Mobile Link ファイルベースのダウンロード用のダウンロードファイル, 215
ファイル定義データベース, 213
サブセット
リモートへのデータサブセットのダウンロード, 113
サンプル
.NET 同期論理, 546
サンプルのドメイン設定ファイル
Mobile Link, 545
サンプルプロパティ
Mobile Link プロファイラ, 203
サーバ
Mobile Link サーバユーティリティ (mlsrv16), 29
サーバ起動同期 (*参照* サーバ起動同期)
サーバ起動リモートタスク
SIRT, 179
定義, 155
サーバシステムプロシージャ
Mobile Link, 644
サーバファーム
Mobile Link, 39
Mobile Link mlsrv16 -zs オプション, 104
負荷分散, 39
サービス
Mobile Link, 34
Mobile Link サーバ, 33
Mobile Link での追加, 34
Mobile Link の依存性, 37
Mobile Link 複数実行, 37
削除, 34
サービスとしての Mobile Link の実行, 34
設定, 34
サービス作成ウィザード
Mobile Link, 34
サービスの依存性
Mobile Link, 37

し

識別子
IBM DB2 LUW の最大長, 4
時刻の切り替え
Mobile Link, 111

自己参照テーブル
Mobile Link, 139
自己参照テーブルからのデータのアップロード
説明, 139
自己参照テーブルの同期
Mobile Link, 139
システムデータベース
Mobile Link, 4
システムテーブル
Mobile Link 統合データベースに作成, 3
システムパラメータ
Mobile Link スクリプト, 235
システムプロシージャ
ml_add_cs, 648
ml_add_dcs, 649
ml_add_dts, 650
ml_add_jcs, 651
ml_add_lcs, 653
ml_add_lcs_chk, 653
ml_add_lts, 654
Mobile Link, 644
Mobile Link システムプロシージャのアルファベット順リスト, 644
実行
Mobile Link サーバ, 29
失敗したダウンロード
Mobile Link, 134
同期の方法, 134
失敗したダウンロードの再開
Mobile Link, 134
自動検証
Mobile Link ファイルベースのダウンロード, 217
シャットダウン
Mobile Link サーバ, 31
Mobile Link 停止ユーティリティ (mlstop), 692
シャドウテーブル
download_delete_cursor スクリプトの作成, 265
重複
分割, 113
終了
Mobile Link サーバ, 31
受信者
展開済みリモートタスクへの追加, 178
照合順
Mobile Link 同期, 758
[詳細テーブル] ウィンドウ枠
Mobile Link プロファイラ, 197
冗長性
Mobile Link サーバオプション (mlsrv16) -v, 85
Mobile Link のパフォーマンス, 146
[使用率グラフ] ウィンドウ枠
Mobile Link プロファイラ, 199
シーケンス
Mobile Link 同期におけるプライマリキーの一意性, 21

す

スイッチ
Mobile Link サーバ (mlsrv16), 43
Mobile Link ユーザ認証ユーティリティ (mluser), 694
スキーマ
Mobile Link リモートテーブルと関係する統合テーブル, 2
スクリプト
.NET 接続スクリプトの追加と削除, 649
.NET テーブルスクリプトの追加と削除, 650
Java 接続スクリプトの追加と削除, 651
Java テーブルスクリプトの追加と削除, 652
Mobile Link イベント, 269
Mobile Link イベントの概要, 269
Mobile Link での統合データベースへの追加, 256
Mobile Link で必要なスクリプト, 255
Mobile Link の説明, 229
SQL 接続スクリプトの追加と削除, 648
SQL テーブルスクリプトの追加と削除, 664
空間データの同期, 8
グローバルスクリプトバージョン, 253
サポートされている DBMS スクリプトの作成方法, 5
接続スクリプト, 233
テーブルスクリプト, 234
バージョン, 252
ローをアップロードするスクリプトの作成, 260
ローをダウンロードするスクリプトの作成, 262
スクリプトの削除
Mobile Link、説明, 256
スクリプトの種類
Mobile Link, 233
スクリプトの直接挿入
Mobile Link, 259

スクリプトの追加
Mobile Link、説明, 256
スクリプトの追加または削除
Mobile Link, 257
スクリプトのパラメータ
Mobile Link の説明, 234
スクリプトパラメータ
last_download, 109
last_table_download, 109
スクリプトバージョン
Mobile Link 同期, 252
グローバル, 253
削除, 255
追加, 254
予約された名前, 253
スクリプトバージョン作成ウィザード
使用, 254
スクリプトバージョンの削除
Mobile Link, 255
スクリプトバージョンの消去 (*参照* スクリプトバージョンの削除)
スクリプトバージョンの追加
Mobile Link, 254
スクリプトを追加または削除するためのシステムプロシージャ
Mobile Link サーバ, 644
ストアドプロシージャ
Mobile Link, 644
Mobile Link ストアドプロシージャのソースコード, 259
データのダウンロードに使用, 137
同期スクリプトの追加と削除に使用, 256
ストアドプロシージャコールからの結果セットのダウンロード
同期の方法, 137
ストリームパラメータ
(*参照* プロトコルオプション)
スナップショットアイソレーション
Mobile Link, 140
Mobile Link mlsrv16 -dsd オプションによる無効化, 59
Mobile Link mlsrv16 -esu オプションによるアップロードでの有効化, 61
SQL Server 用の Mobile Link mlsrv16 -dt オプション, 59
スナップショットを使った同期
説明, 112
スプレッドシート
Mobile Link との同期, 623
スレッド
Mobile Link のパフォーマンス, 143
Mobile Link のワーカスレッドとパフォーマンス, 143
スレッド化
(*参照* スレッド)
自動調整, 144

せ

生成された Mobile LInk リプレイ API ユーティリティ (mlgenreplayapi)
構文, 704
制約エラー (*参照* 競合)
セキュリティ
Mobile Link ユーザ認証ユーティリティ (mluser), 694
世代番号
Mobile Link ファイルベースのダウンロード, 219
セッション全体の変数
IBM DB2 Mobile Link 統合データベース, 13
Oracle Mobile Link 統合データベース, 20
セッションプロパティ
Mobile Link プロファイラ, 203
接続
Mobile Link mlsrv16 -c オプション, 52
Mobile Link mlsrv16 -x オプション, 92
接続スクリプト
.NET スクリプトの削除, 649
.NET スクリプトの追加, 649
Java スクリプトの削除, 651
Java スクリプトの追加, 651
ml_global, 253
Mobile Link スクリプトのアルファベット順リスト, 269
SQL スクリプトの削除, 648
SQL スクリプトの追加, 648
Sybase Central を使用した追加, 257
説明, 233
定義, 233
接続スクリプトの作成ウィザード
使用, 257
接続パラメータ
Mobile Link mlsrv16 -x オプション, 92
接続プロパティ
Mobile Link mlsrv16 -x オプション, 92

接続文字列
Mobile Link mlsrv16, 52
接続レベルスクリプト
定義, 233
切断分割
Mobile Link, 114
定義, 113
設定
Mobile Link .NET 同期論理, 533
Mobile Link、Java 同期論理, 459
Mobile Link 統合データベース, 1, 3
Mobile Link ファイルベースのダウンロード, 213
設定スクリプト
Mobile Link システムデータベース, 4
Mobile Link 統合データベース, 3
選択的な共有 (*参照* 分割)
全方向同期 (*参照* 双方向同期)

そ

挿入
Mobile Link のスクリプト, 259
双方向同期
説明, 116
必要なスクリプト, 255
ソフトシャットダウン
Mobile Link 停止ユーティリティ (mlstop), 692
ソート順
文字と Mobile Link 同期, 758

た

タイムアウト
-tc オプション (mlsrv16), 80
-tf オプション (mlsrv16), 81
タイムスタンプ
Mobile Link ダウンロード, 110
タイムスタンプベースのダウンロード
説明, 108
タイムスタンプベースの同期
download_cursor スクリプト, 109
説明, 108
ダイレクトアップロードでの競合の処理
Mobile Link ダイレクトローハンドリング, 629
ダイレクトアップロードの処理
Mobile Link ダイレクトローハンドリング, 627
ダイレクトダウンロードの処理
Mobile Link ダイレクトローハンドリング, 633
ダイレクト同期イベント
説明, 624
ダイレクトローアクセス (*参照* ダイレクトローハンドリング)
ダイレクトローハンドリング
handle_DownloadData 接続イベント, 371
handle_UploadData 接続イベント, 383
アップロード, 627
開発のヒント, 626
クイックスタート, 625
説明, 623
ダウンロード, 633
ダイレクトローハンドリングの設定
説明, 625
ダウンロード
Mobile Link 失敗したダウンロード, 134
Mobile Link でのファイルベース, 212
Mobile Link トランザクション, 273
Mobile Link パフォーマンス, 147
Mobile Link ローをダウンロードするスクリプト, 262
タイムスタンプベース, 108
ダウンロードイベント
Mobile Link 同期, 280
ダウンロード確認
説明, 137
ダウンロード専用の同期
説明, 116
必要なスクリプト, 255
ダウンロードタイムスタンプ
Mobile Link での生成, 110
Mobile Link の説明, 109
ダウンロード中のイベント
説明, 280
ローをダウンロードするスクリプトの作成, 262
ダウンロードトランザクション
Mobile Link, 273
ダウンロードの失敗
Mobile Link 再起動可能なダウンロード, 134
ダウンロードのみ
(*参照* ダウンロード専用)
ダウンロードのみの同期
(*参照* ダウンロード専用の同期)
ダウンロードファイル
Mobile Link ファイルベースのダウンロード用に作成, 215
ダウンロードファイルの作成

Mobile Link ファイルベースのダウンロード, 215
タスク
(*参照* リモートタスク)
多対多関係
同期, 115
分割, 115
断片化
(*参照* 分割)

ち

[チャート] ウィンドウ枠
Mobile Link プロファイラ, 200
中央データベース
Mobile Link 統合データベース, 1
重複のある分割
Mobile Link, 115

つ

追加
Mobile Link .NET 接続スクリプト, 649
Mobile Link .NET テーブルスクリプト, 650
Mobile Link Java 接続スクリプト, 651
Mobile Link Java テーブルスクリプト, 652
Mobile Link SQL 接続スクリプト, 648
Mobile Link SQL テーブルスクリプト, 664
Mobile Link のプロパティ, 660
Mobile Link のユーザ名, 694
Sybase Central を使用した同期スクリプトの追加, 257
通信
Mobile Link mlsrv16 -c オプション, 52
Mobile Link mlsrv16 -x オプション, 92
ツール
Mobile Link プロファイラのマーキーツール, 202

て

停止
Mobile Link サーバ, 31
Mobile Link 停止ユーティリティ (mlstop), 692
SQL Anywhere クライアントの削除のアップロード, 133
停止ユーティリティ (mlstop)
構文, 692
テキストファイル
Mobile Link との同期, 623
デバッグ
.NET 同期論理, 542
Java クラスを使用した Mobile Link 同期, 464
Mobile Link サーバのログ, 31
Mobile Link 接続, 42
デフォルト値の選択方法
Mobile Link グローバルオートインクリメント, 121
デフォルトのグローバルオートインクリメント
Mobile Link でのせんげん, 120
デフォルトの独立性レベル
Mobile Link, 140
展開済みタスク (*参照* 展開済みリモートタスク)
展開済みリモートタスク
開始, 177
キャンセル, 176
再アクティブ化, 178
受信者の追加, 178
操作, 175
定義, 154
データ型
Mobile Link .NET と SQL, 537
Mobile Link IBM DB2 LUW, 718
Mobile Link Java と SQL, 462
Mobile Link Microsoft SQL Server, 727
Mobile Link MySQL, 734
Mobile Link Oracle, 740
Mobile Link と SAP HANA のマッピング, 751, 755
Mobile Link 統合データベースのマッピング, 708
Mobile Link マッピング ASE, 708
データ型マッピング
Mobile Link 統合データベース, 708
データ交換 (*参照* 同期)
データスクリプト
download_cursor, 329
download_delete_cursor, 332
handle_DownloadData, 371
handle_UploadData, 383
upload_delete, 433
upload_fetch, 435
upload_fetch_column_conflict, 437
upload_insert, 439
upload_new_row_insert, 441
upload_old_row_insert, 443
upload_update, 455
説明, 281

データソース名
Oracle ドライバのオプション, 761
データの一貫性
(*参照* 同期)
データのダウンロード
Mobile Link でのファイルベースのダウンロード, 212
データの調整 (*参照* 同期)
データの不整合
Mobile Link での競合処理, 124
データフロー (Mobile Link) (*参照* ダイレクトローハンドリング)
データベース
Mobile Link 統合データベース, 1
データベーススキーマ
Mobile Link リモートテーブルと関係する統合テーブル, 2
データベース接続
Mobile Link のパフォーマンス, 151
Mobile Link のパフォーマンス、最大数の設定, 145
データベースツールインタフェース
(*参照* DBTools インタフェース)
データベースの作成
統合, 3
データベースワーカスレッド
Mobile Link, 148
Mobile Link のパフォーマンス, 143
自動調整, 144
データマッピング
説明, 708
データベースの集中管理
SIRT, 179
テーブル
Mobile Link リモートテーブルと関係する統合テーブル, 2
分割, 113
テーブルから全ローを削除
Mobile Link, 266
テーブルスクリプト
.NET スクリプトの削除, 650
.NET スクリプトの追加, 650
Java スクリプトの削除, 652
Java スクリプトの追加, 652
Mobile Link, 231
Mobile Link スクリプトのアルファベット順リスト, 269
SQL スクリプトの削除, 664
SQL スクリプトの追加, 664
Sybase Central を使用した追加, 258
説明, 234
定義, 233
テーブルスクリプト追加ウィザード
使用, 258
テーブルの分割
例, 113
テーブル領域の容量
IBM DB2 Mobile Link 統合データベース, 13
テーブルレベルスクリプト
定義, 233
デーモン
デーモンとしての Mobile Link の実行, 33

と

同期
(*参照* 同期)
.NET での Mobile Link スクリプトの作成, 533
Java での Mobile Link スクリプトの作成, 459
mlsrv16 のプロトコルオプション, 92
Mobile Link ASE のデータ型, 708
Mobile Link IBM DB2 LUW のデータ型, 718
Mobile Link Microsoft SQL Server のデータ型, 727
Mobile Link MySQL のデータ型, 734
MobiLink SAP HANA データ型, 751, 755
Mobile Link サーバの実行, 29
Mobile Link システムプロシージャ, 644
Mobile Link におけるデータ型マッピング, 708
Mobile Link の Oracle データ型, 740
Mobile Link 文字セット, 758
Mobile Link 文字セット変換, 759
Mobile Link ユーティリティ, 692
イベントの概要, 269
競合解決, 124
空間データ, 6
再起動可能なダウンロード, 134
スクリプトのアルファベット順リスト, 269
スクリプトの作成, 229
スナップショット, 112
多対多関係, 115
統合データベース, 1
パフォーマンスに関するヒント, 142
プロセス, 232
プロファイラの接続パラメータ, 192
方法, 107

ローの削除, 265
ローのダウンロード, 262
同期イベント
ASE begin_connection_autocommit 接続イベント, 301
Mobile Link アップロード, 277
Mobile Link ダウンロード, 280
Mobile Link 同期の説明, 269
イベントスクリプトのアルファベット順リスト, 269
説明, 269
同期エラー
Mobile Link の処理, 267
トラブルシューティング, 60
同期間隔
Mobile Link エージェント, 166
同期サブスクリプション
(*参照* サブスクリプション)
同期サーバ (*参照* Mobile Link サーバ)
同期スクリプト
.NET, 533
.NET メソッド, 538
.NET 用に実装, 533
DBMS 依存性, 5
download_cursor, 263
handle_error イベント, 267
Java, 459
Java メソッド, 464
Java 用の実装, 459
Mobile Link イベント, 269
report_error, 268
Sybase Central を使用した追加, 257
サポートされている DBMS スクリプトの作成方法, 5
実行, 232
種類, 233
スクリプトバージョン, 252
ストアドプロシージャを使用した追加と削除, 256
接続スクリプト, 233
説明, 229
追加と削除, 256
テーブルスクリプト, 234
パラメータ, 234
例, 231
ローをアップロードするスクリプトの作成, 260
ローをダウンロードするスクリプトの作成, 262
同期スクリプトの追加
ストアドプロシージャの使用, 256
同期ストリームライブラリ
32 ビットの UNIX での Mobile Link サーバ配備, 771
32 ビットの Windows での Mobile Link サーバ配備, 766
64 ビットの UNIX での Mobile Link サーバ配備, 774
64 ビットの Windows での Mobile Link サーバ配備, 768
同期の方法
失敗したダウンロード, 134
スナップショットベースの同期, 112
説明, 107
タイムスタンプベースの同期, 108
ダウンロードするためのストアドプロシージャ, 137
プライマリキープール, 122
分割, 113
ローのアップロード, 260
ローの削除, 132
同期パラメータ
HTTPS 同期, 92
HTTP 同期, 92
TCP/IP 同期, 92
同期プロパティ
Mobile Link プロファイラ, 203
同期ユーザ
Mobile Link ユーザ認証ユーティリティ (mluser), 694
同期論理
Mobile Link, 229
統計
Mobile Link, 208
統計のカスタマイズ
Mobile Link プロファイラ, 204
統計のプロパティ
Mobile Link, 208
統合データベース
DBMS 依存性, 5
Mobile Link 統合データベースとして使用する IBM DB2 LUW, 12
Mobile Link 統合データベースとして使用する MySQL, 16

Mobile Link 統合データベースとして使用する Oracle, 19
Mobile Link 統合データベースとして使用する SQL Anywhere, 26
Mobile Link 統合データベースとして使用する SQL Server, 14
Mobile Link 統合データベースとしての ASE, 10
Mobile Link 統合データベースの作成, 3
Mobile Link 独立性レベル, 140
Mobile Link としての SAP Sybase IQ, 27
Mobile Link としての SAP HANA, 24
Mobile Link リモートテーブルと関係するテーブル, 2
SQL Anywhere 以外のデータベース, 5
データ型の Mobile Link マッピング, 708
同期スクリプトの追加, 256
統合データベース以外のデータソースの同期
説明, 623
統合データベースの作成
Mobile Link、説明, 3
同時実行性
Mobile Link のパフォーマンス, 142
動的キャッシュサイズ決定
Mobile Link サーバ, 40
登録
Mobile Link スクリプトとしてのメソッド, 644
独立性レベル
Mobile Link, 140
ドメイン設定ファイル
Mobile Link, 545
ドライバ
Mobile Link でサポートされるドライバ, 760
トラブルシューティング
Mobile Link 再起動可能なダウンロード, 134
Mobile Link サーバ起動時, 42
Mobile Link サーバのログ, 31
Mobile Link リモートデータ損失, 110
失敗したダウンロードの処理, 134
同期エラー, 60
トランザクション
Mobile Link, 269
Mobile Link .NET 同期論理, 537
Mobile Link Java 同期論理, 462
トランザクション単位のアップロード (*参照* トランザクションレベルのアップロード)
Mobile Link のパフォーマンス, 144

な

夏時間
Mobile Link, 111
名前付きスクリプトパラメータ
ml_add_column システムプロシージャ, 646
Mobile Link の説明, 234
名前付きパラメータ
last_download, 109
last_table_download, 109
Mobile Link の説明, 234
空間データの同期, 7
名前付きローパラメータ
Mobile Link スクリプトの説明, 235
統合データベースへのカラム情報の追加, 646

に

認証
Mobile Link ユーザ認証ユーティリティ (mluser), 694
認証パラメータ
Mobile Link, 251
Mobile Link スクリプト, 235

ね

ネットワークパラメータ
Mobile Link mlsrv16 -x オプション, 92
ネットワークプロトコル
HTTPS を使用した Mobile Link mlsrv16, 97
HTTP 使用した Mobile Link mlsrv16, 95
Mobile Link サーバ, 92
TCP/IP を使用した Mobile Link mlsrv16 -x オプション, 92
TLS over TCP/IP を使用した Mobile Link mlsrv16 -x オプション, 93

は

配備, v
Mobile Link アプリケーション, 765
Mobile Link サーバ, 765
Mobile Link のアプリケーションとデータベース, 765
Mobile Link の概要, 765
Mobile Link のパフォーマンス, 142
SQL Anywhere Mobile Link クライアント, 776
Ultra Light Mobile Link クライアント, 779
配備の概要
Mobile Link, 765

配布可能なダウンロード
Mobile Link ファイルベースのダウンロード, 212
配列サイズ
Oracle ドライバのオプション, 761
パススルーモード
(*参照* SQL パススルー)
パスワード
Mobile Link ユーザ認証ユーティリティ (mluser), 694
Oracle ドライバでの暗号化, 761
パスワードの暗号化
Oracle ドライバのオプション, 761
パッケージ化されたダウンロード
Mobile Link ファイルベースのダウンロード, 212
パフォーマンス
Mobile Link, 142
Mobile Link、BLOB の同期, 145
Mobile Link mlsrv16 -sm オプション, 79
Mobile Link スレッド, 143
Mobile Link データベース接続, 151
Mobile Link データベースワーカスレッド, 143
Mobile Link 同期ロジック, 147
Mobile Link のインクリメンタルアップロード, 144
Mobile Link の競合, 142
Mobile Link の最大データベース接続, 145
Mobile Link のスクリプトの実行, 146
Mobile Link のダウンロード, 147
Mobile Link のチューニング, 148
Mobile Link の同期の優先度, 147
Mobile Link の同時実行性, 142
Mobile Link のトランザクション単位のアップロード, 144
Mobile Link のメモリ, 145
Mobile Link のロギングの冗長性, 146
Mobile Link のローのアップロード, 147
データベースワーカスレッドの自動調整, 144
パフォーマンスのチューニング
Mobile Link, 148
パラメータ
同期スクリプト, 234
バージョン
Mobile Link 同期スクリプト, 252
スクリプトバージョンの削除, 255
スクリプトバージョンの追加, 254
ハードシャットダウン
Mobile Link 停止ユーティリティ (mlstop), 692

ひ

必要なスクリプト
Mobile Link, 255
非ブロッキングダウンロード確認
nonblocking_download_ack 接続イベント, 400
publication_nonblocking_download_ack 接続イベント, 406
説明, 137
非リレーショナルデータベース
Mobile Link との同期, 623
ヒント
Mobile Link のパフォーマンス, 142
同期の方法, 107

ふ

ファイル
Mobile Link ファイルベースのダウンロード, 212
ファイル定義データベース
作成, 213
説明, 213
ファイル定義データベースの作成
Mobile Link, 213
ファイルベースのダウンロード
説明, 212
例, 220
ファーム (*参照* Mobile Link サーバファーム)
フェイルオーバ
Mobile Link サーバファーム, 39
負荷分散
Mobile Link サーバファーム, 39
複合キー
Mobile Link のユニークなプライマリキー, 118
不整合
Mobile Link での競合処理, 124
フック
(*参照* イベントフック)
プライベートアセンブリ
Mobile Link での実装, 544
プライマリキー
Mobile Link での一意性の方法, 117
Mobile Link の説明, 107
Oracle のシーケンス, 21
プライマリキープール
Mobile Link のユニークなプライマリキー, 122

Mobile Link の例, 124
Mobile Link 用にデフォルトのグローバルオートインクリメントを使用したユニークな値の生成, 119
古いローパラメータ
Mobile Link スクリプト, 235
プレフィクス
Mobile Link 名前付きパラメータ, 235
プロジェクト
グループの追加, 167
プロシージャ
Mobile Link, 644
プロシージャコール
SQL Server Mobile Link 統合データベース, 15
プロシージャは結果を返す
Oracle ドライバのオプション, 761
ブロッキングダウンロード確認
説明, 137
プロトコル
(*参照* ネットワークプロトコル)
HTTPS を使用した Mobile Link mlsrv16, 97
HTTP を使用した Mobile Link mlsrv16, 95
Mobile Link サーバ, 92
TCP/IP を使用した Mobile Link mlsrv16 -x オプション, 92
TLS over TCP/IP を使用した Mobile Link mlsrv16 -x オプション, 93
プロパティを追加または削除するためのシステムプロシージャ
Mobile Link サーバ, 646
プロファイラ
Mobile Link プロファイラ, 191
Mobile Link プロファイラ (mlprof), 193
プロファイリング
Mobile Link での同期, 191
プロファイリングセッション
以前のセッションの削除, 195
以前のセッションを開く, 195
ウォッチの作成, 205
開始, 193
終了, 195
プロファイリングデータベース
Mobile Link, 196
表示, 205
ブロードキャストダウンロード
Mobile Link ファイルベースのダウンロード, 212
分割
Mobile Link、切断, 114
Mobile Link の説明, 113
定義, 113
文ベースのアップロード
競合検出, 125
文ベースのスクリプト
ローのアップロード, 260

へ

変換
文字セットの ODBC ドライバによる変換, 760

ほ

ボトルネック
Mobile Link のパフォーマンス, 148
ポーリング間隔
Mobile Link エージェント, 166

ま

マッピング
Mobile Link 統合データベースのデータ型, 708
マルチスレッドネットワーク処理
Mobile Link サーバオプション (mlsrv16), 80
Mobile Link のパフォーマンス, 143
マーキーツール
Mobile Link プロファイラ [概要] ウィンドウ枠, 202

め

メソッド
Mobile Link .NET 同期論理, 538
Mobile Link Java 同期論理, 464
メソッドの登録
Java 用 Mobile Link サーバ API, 464
メモリの使用
Mobile Link サーバ, 40

も

文字セット
Mobile Link 同期, 758
文字セット間の変換
Mobile Link 同期, 759
文字セットの考慮事項
Mobile Link, 758
文字セット変換
Mobile Link 同期中, 759
ODBC ドライバによる変換, 760

戻り値
.NET 同期, 538
Java 同期, 464
モニタ
Mobile Link のパフォーマンス, 151
モニタリング
Mobile Link での同期, 191

ゆ

優先同期
Mobile Link のパフォーマンス, 147
ユニーク
Mobile Link のプライマリキー, 117
ユニークなキー
Mobile Link, 117
ユニークなプライマリキー
Mobile Link, 117
Mobile Link 用にキープールを使用した生成, 122
Mobile Link 用にグローバルオートインクリメント, 119
UUID を使用して Mobile Link 用に生成, 118
複合キーを使用して Mobile Link 用に生成, 118
ユニークなプライマリキーの管理
UUID, 118
グローバルオートインクリメント, 119
説明, 117
複合キー, 118
プライマリキープール, 122
ユーザ定義起動クラス
Mobile Link .NET 同期論理, 539
Mobile Link Java 同期論理, 467
ユーザ定義のパラメータ
Mobile Link, 250
ユーザ定義のプロシージャ
IBM DB2 Mobile Link 統合データベース, 13
ユーザ認証ユーティリティ (mluser)
構文, 694
ユーザパラメータ
Mobile Link, 250
ユーザ名
Mobile Link ユーザ認証ユーティリティ (mluser), 694
ユーティリティ
Mobile Link, 692
Mobile Link 停止 (mlstop) 構文, 692
Mobile Link ユーザ認証 (mluser) 構文, 694

り

リセット
Mobile Link の最終ダウンロード時刻, 111
リモート管理 (*参照* リモートデータベースの集中管理)
リモートスキーマ名
インポート, 164
追加, 164
定義, 155
リモートタスク
copy file コマンド, 181
create database コマンド, 181
delete file コマンド, 182
download file コマンド, 182
drop database コマンド, 182
execute SQL コマンド, 183
prompt コマンド, 184
rename file コマンド, 184
run program コマンド, 184
synchronize コマンド, 185
upload file コマンド, 185
エクスポート, 175
コマンド, 180
コマンドの追加, 186
作成, 172
ステータス, 188
ステータスレポート, 171
説明, 169
定義, 154
展開, 174
編集, 173
変数, 187
ロジック, 172
リモートタスクの集中管理 (*参照* リモートデータベースの集中管理)
リモートタスクの属性
高優先度, 171
コマンド, 170
再試行間隔, 170
最大再試行回数, 170
最大実行時間, 170
条件, 170
スキーマ変更, 171
説明, 169
トリガのメカニズム, 169
名前, 169
ランダムな遅延時間, 171

リモートスキーマ名, 170
リモートデータベース
Mobile Link リモートテーブルと関係する統合テーブル, 2
データ型の Mobile Link マッピング, 708
リモートデータベースと統合データベース間での Mobile Link データマッピング
説明, 708
リモートデータベースの集中管理
copy file コマンド, 181
create database コマンド, 181
delete file コマンド, 182
download file コマンド, 182
drop database コマンド, 182
execute SQL コマンド, 183
Mobile Link エージェント ID の定義, 154
Mobile Link エージェントの定義, 153
Mobile Link プロジェクトの定義, 153
prompt コマンド, 184
rename コマンド, 184
run program コマンド, 184
Sybase Central, 164
synchronize コマンド, 185
upload file コマンド, 185
エージェントデータベースの定義, 154
エージェントの停止, 163
概念, 153
コマンド, 180
コマンドの定義, 154
サーバ起動リモートタスクの定義, 155
システムプロシージャ, 189
ステータス, 188
ステータス情報の定義, 154
設定, 155
説明, 152
展開済みタスクのキャンセル, 176
展開済みタスクの操作, 175
展開済みタスクへの受信者の追加, 178
展開済みリモートタスクの定義, 154
リモートスキーマ名の定義, 155
リモートタスク, 169
リモートタスクのエクスポート, 175
リモートタスクの開始, 177
リモートタスクの再アクティブ化, 178
リモートタスクの作成, 172
リモートタスクの定義, 154
リモートタスクの展開, 174
リモートタスクの編集, 173
リモートデータベースの定義, 155
リモートデータベースの配備
説明, 765
リモートテーブル
Mobile Link でのローの削除, 265

る

ルール
(*参照* 転送ルール)

れ

例
Java 同期論理, 468
レプリケーション
(*参照* Mobile Link)

ろ

ロギング
Mobile Link サーバの動作, 31
Mobile Link のパフォーマンス, 146
ログ
(*参照* ログファイル)
ログファイル
Mobile Link サーバ, 31
Mobile Link サーバの表示, 32
ログファイルビューア
Mobile Link サーバログ, 32
ロックのエスカレーション
IBM DB2 Mobile Link 統合データベース, 13
論理削除
download_delete_cursor スクリプトの作成, 265
ロー
Mobile Link リモートデータベースでの削除, 265
分割, 113
ローのアップロード
.NET の同期の方法, 544
Mobile Link のパフォーマンス, 147
スクリプトの作成, 260
ローの削除
Mobile Link の方法, 132
Mobile Link リモートデータベース, 265
ローのダウンロード
同期スクリプト, 262
ローの分割
Mobile Link、リモートデータベース間, 113
ローパラメータ

Mobile Link スクリプト, 235

わ

ワーカスレッド
Mobile Link, 148
Mobile Link のパフォーマンス, 143
自動調整, 144