iAnywhere SOLUTIONS™
A SYBASE COMPANY

Ultra Light 静的型 Java ユーザーズ・ガイド

パート番号：DC50036-01-0902-01
改訂：2005 年 3 月

目次

はじめに

このマニュアルの内容

このマニュアルでは、Ultra Light 静的型 Java API について説明します。『*Ultra Light データベース・ユーザーズ・ガイド*』の内容を補足説明します。

対象読者

このマニュアルでは、Ultra Light データベースを使用した Java プログラムの開発を行うアプリケーション開発者を対象としています。リレーショナル・データベースと Adaptive Server Anywhere の知識を持っていることを前提とします。

SQL Anywhere Studio のマニュアル

このマニュアルは、SQL Anywhere のマニュアル・セットの一部です。この項では、マニュアル・セットに含まれる各マニュアルと使用法について説明します。

SQL Anywhere Studio のマニュアル

SQL Anywhere Studio のマニュアルは、各マニュアルを 1 つの大きなヘルプ・ファイルにまとめたオンライン形式、マニュアル別の PDF ファイル、および有料の製本版マニュアルで提供されます。SQL Anywhere Studio のマニュアルは、次の分冊マニュアルで構成されています。

- **『SQL Anywhere Studio の紹介』** このマニュアルでは、SQL Anywhere Studio のデータベース管理と同期テクノロジの概要について説明します。また、SQL Anywhere Studio を構成する各部分について説明するチュートリアルも含まれています。
- **『SQL Anywhere Studio 新機能ガイド』** このマニュアルは、SQL Anywhere Studio のこれまでのリリースのユーザを対象としています。ここでは、製品の今回のリリースと以前のリリースで導入された新機能をリストし、アップグレード手順を説明しています。
- **『Adaptive Server Anywhere データベース管理ガイド』** このマニュアルでは、データベースおよびデータベース・サーバの実行、管理、設定について説明しています。
- **『Adaptive Server Anywhere SQL ユーザーズ・ガイド』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『Adaptive Server Anywhere SQL リファレンス・マニュアル』** このマニュアルは、Adaptive Server Anywhere で使用する SQL 言語の完全なリファレンスです。また、Adaptive Server Anywhere のシステム・テーブルとシステム・プロシージャについても説明しています。
- **『Adaptive Server Anywhere プログラミング・ガイド』** このマニュアルでは、C、C++、Java プログラミング言語を使用してデータベース・アプリケーションを構築、配備する方法につい

て説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。また、Adaptive Server Anywhere ADO.NET データ・プロバイダについても説明します。

- **『Adaptive Server Anywhere SNMP Extension Agent ユーザーズ・ガイド』** このマニュアルでは、SNMP 管理アプリケーションとともに使用するように Adaptive Server Anywhere SNMP Extension Agent を設定して、Adaptive Server Anywhere データベースを管理する方法を説明します。

- **『Adaptive Server Anywhere エラー・メッセージ』** このマニュアルでは、Adaptive Server Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。

- **『SQL Anywhere Studio セキュリティ・ガイド』** このマニュアルでは、Adaptive Server Anywhere データベースのセキュリティ機能について説明します。Adaptive Server Anywhere 7.0 は、米国政府から TCSEC (Trusted Computer System Evaluation Criteria) の C2 セキュリティ評価を授与されています。このマニュアルには、Adaptive Server Anywhere の現在のバージョンを、C2 基準を満たした環境と同等の方法で実行することを望んでいるユーザにとって役に立つ情報が含まれています。

- **『Mobile Link 管理ガイド』** このマニュアルでは、モバイル・コンピューティング用の Mobile Link データ同期システムについてあらゆる角度から説明します。このシステムによって、Oracle、Sybase、Microsoft、IBM の単一データベースと、Adaptive Server Anywhere や Ultra Light の複数データベースの間でのデータ共有が可能になります。

- **『Mobile Link クライアント』** このマニュアルでは、Adaptive Server Anywhere リモート・データベースと Ultra Light リモート・データベースの設定を行い、これらを同期させる方法について説明します。

- **『Mobile Link サーバ起動同期ユーザーズ・ガイド』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期の開始を可能にする Mobile Link の機能です。

- **『Mobile Link チュートリアル』** このマニュアルには、Mobile Link アプリケーションの設定と実行を行う方法を説明するチュートリアルがいくつか用意されています。
- **『QAnywhere ユーザーズ・ガイド』** このマニュアルでは、Mobile Link QAnywhere について説明します。Mobile Link QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアントだけでなく、モバイル・クライアントや無線クライアント用のメッセージング・アプリケーションの開発と展開を可能にするメッセージング・プラットフォームです。
- **『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』** このマニュアルでは、Mobile Link 同期サーバから、または Adaptive Server Anywhere リモート・データ・アクセスによって、Adaptive Server Anywhere 以外の統合データベースにアクセスするための ODBC ドライバの設定方法について説明します。
- **『SQL Remote ユーザーズ・ガイド』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて、あらゆる角度から説明します。このシステムによって、Adaptive Server Anywhere または Adaptive Server Enterprise の単一データベースと Adaptive Server Anywhere の複数データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere Studio ヘルプ』** このマニュアルには、Sybase Central や Interactive SQL、その他のグラフィカル・ツールに関するコンテキスト別のヘルプが含まれています。これは、製本版マニュアル・セットには含まれていません。
- **『Ultra Light データベース・ユーザーズ・ガイド』** このマニュアルは、Ultra Light 開発者を対象としています。ここでは、Ultra Light データベース・システムの概要について説明します。また、すべての Ultra Light プログラミング・インタフェースに共通する情報を提供します。
- **Ultra Light のインタフェースに関するマニュアル** 各 Ultra Light プログラミング・インタフェースには、それぞれに対応するマニュアルを用意しています。これらのインタフェースは、RAD (

ラピッド・アプリケーション開発) 用の Ultra Light コンポーネントとして提供されているものと、C、C++、Java 開発用の静的インタフェースとして提供されているものがあります。

このマニュアル・セットの他に、PowerDesigner と InfoMaker には、独自のオンライン・マニュアル (英語版) がそれぞれ用意されています。

マニュアルの形式

SQL Anywhere Studio のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere Studio の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

  Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] ー [プログラム] ー [SQL Anywhere 9] ー [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

  UNIX オペレーティング・システムでオンライン・マニュアルにアクセスするには、SQL Anywhere のインストール・ディレクトリに保存されている HTML マニュアルを参照してください。

- **PDF 版マニュアル** SQL Anywhere の各マニュアルは、Adobe Acrobat Reader で表示できる PDF ファイルで提供されています。

  PDF 版マニュアルは、オンライン・マニュアルまたは Windows の [スタート] メニューから利用できます。

- **製本版マニュアル** 製本版マニュアルをご希望の方は、ご購入いただいた販売代理店または弊社営業担当までご連絡ください。

表記の規則

この項では、このマニュアルで使用されている書体およびグラフィック表現の規則について説明します。

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

- **キーワード** SQL キーワードはすべて次の例に示す ALTER TABLE のように大文字で表記します。

  ALTER TABLE [*owner*.]*table-name*

- **プレースホルダ** 適切な識別子または式で置き換えられる項目は、次の例に示す *owner* や *table-name* のように表記します。

  ALTER TABLE [*owner*.]*table-name*

- **繰り返し項目** 繰り返し項目のリストは、次の例に示す *column-constraint* のように、リストの要素の後ろに省略記号 (ピリオド 3 つ …) を付けて表します。

  ADD *column-definition* [*column-constraint*, …]

  複数の要素を指定できます。複数の要素を指定する場合は、各要素間をカンマで区切る必要があります。

- **オプション部分** 文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

  この例では、角カッコで囲まれた *savepoint-name* がオプション部分です。角カッコは入力しないでください。

- **オプション** 項目リストから 1 つだけ選択するか、何も選択しなくてもよい場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。

  [**ASC** | **DESC**]

  この例では、ASC と DESC のどちらか 1 つを選択しても、どちらも選択しなくてもかまいません。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコで囲み、縦棒で区切ります。

  [**QUOTES** { **ON** | **OFF** }]

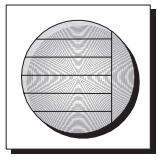
  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

グラフィック・アイコン

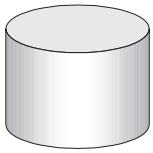
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

- Sybase Adaptive Server Anywhere などのデータベース・サーバ



- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- レプリケーションまたは同期のミドルウェア。ソフトウェアのこれらの部分は、データベース間のデータ共有を支援します。たとえば、Mobile Link 同期サーバ、SQL Remote Message Agent などがあげられます。

- プログラミング・インタフェース

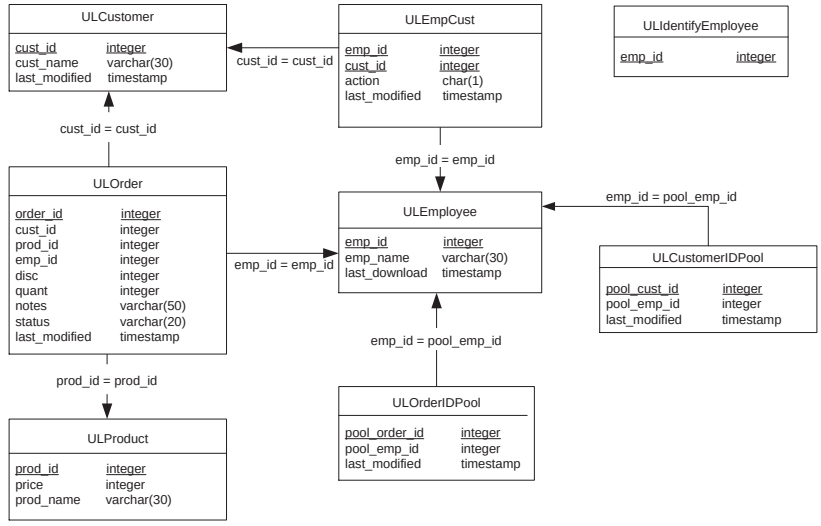
CustDB サンプル・データベース

Mobile Link と Ultra Light のマニュアルでは、多くの例で Ultra Light のサンプル・データベースが使用されています。

Ultra Light サンプル・データベースのリファレンス・データベースは、*custdb.db* という名前のファイルに保存され、SQL Anywhere ディレクトリのサブディレクトリ *Samples¥UltraLite¥CustDB* に置かれています。全面的にこのデータベースを使用して構築したアプリケーションも提供されています。

サンプル・データベースは、あるハードウェア販売会社の販売管理データベースです。データベースには、この販売会社の顧客、製品、営業戦力に関する情報が入っています。

次の図は、CustDB データベース内のテーブルと、各テーブル間の関係を示しています。



詳細情報の検索／フィードバックの提供

詳細情報の検索

詳しい情報やリソース (コード交換など) については、iAnywhere Developer Network (http://www.ianywhere.com/developer/) を参照してください。

ご質問がある場合や支援が必要な場合は、次に示す iAnywhere Solutions ニュースグループのいずれかにメッセージをお寄せください。

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere Studio バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng9 -v** と入力して確認できます。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります (ニュースグループにおけるサービスは英語でのみの提供となります)。以下のニュースグループがあります。

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

ニュースグループに関するお断り

iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。

iAnywhere Solutions のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

マニュアルに関するご意見、ご提案は、SQL Anywhere ドキュメンテーション・チームの iasdoc@ianywhere.com 宛てに電子メールでお寄せください。このアドレスに送信された電子メールに返信はいたしませんが、お寄せいただいたご意見、ご提案は必ず読ませていただきます。

マニュアルまたはソフトウェアについてのフィードバックは、上記のニュースグループを通してお寄せいただいてもかまいません。

第 1 章

静的型 Java API の概要

この章の内容

この章では、Ultra Light データベースの静的型 Java API について説明します。ここでは、Ultra Light データベース・システムとその開発モデルを理解していることを前提にしています。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light へようこそ」を参照してください。

システムの稼働条件とサポートされるプラットフォーム

サポートされるターゲット・プラットフォームは、Sun Personal Java 1.2 です。

アプリケーション開発にはサポートされる JDK が必要です。また、Adaptive Server Anywhere リファレンス・データベースも必要です。

詳細については、『SQL Anywhere Studio の紹介』> 「Ultra Light 開発プラットフォーム」および『SQL Anywhere Studio の紹介』> 「Ultra Light ターゲット・プラットフォーム」を参照してください。

静的型 Java アプリケーションの開発

静的型 Java Ultra Light アプリケーション開発では、JDBC に似たプログラミング・インタフェースを使用します。これらのアプリケーションを開発するには、Java プログラミング言語に精通していることが必要です。

静的型 Java Ultra Light アプリケーションの開発プロセスは、次のとおりです。

1. データベースを設計します。

   Ultra Light データベースにインクルードするテーブルとインデックスを含む Adaptive Server Anywhere リファレンス・データベースを準備します。

2. SQL 文をデータベースに追加します。

   アプリケーションに使用する SQL 文をリファレンス・データベースに追加する必要があります。

3. アプリケーションのクラスを生成します。

   Ultra Light ジェネレータを使用して、アプリケーションに必要なクラスを生成します。

4. アプリケーションを作成します。

   このアプリケーション・コードのデータ・アクセス機能は、JDBC と他の関数呼び出しを使用します。

   このインタフェースについては、『Ultra Light 静的型 Java ユーザーズ・ガイド』>「Ultra Light 静的型 Java API リファレンス」を参照してください。

5. .java ファイルをコンパイルします。

   生成した .java ファイルは、他の .java ファイルと同じようにコンパイルできます。

開発プロセスの詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「Ultra Light 静的型 Java アプリケーションの構築」を参照してください。

静的型 Java API の利点と制約

Ultra Light には、静的型開発モデル (静的型 Java インタフェースはこのうちの 1 つ) と Ultra Light コンポーネントの 2 つ以外にも、いくつかのプログラミング・インタフェースが用意されています。Java ベースのコンポーネント (Native Ultra Light for Java) もこのうちの 1 つです。

静的型 Java API には、次のような利点があります。

- **Pure Java ソリューション** 静的型 Java API 用の Ultra Light ランタイム・ライブラリは、pure Java アプリケーションです。Native Ultra Light for Java コンポーネントとは異なり、同じ C++ ベースの Ultra Light ランタイム・ライブラリを他の Ultra Light インタフェースとして共有します。Native Ultra Light for Java コンポーネントでは、Ultra Light ランタイムへのアクセスはネイティブ・メソッドによって提供されます。
- **さまざまな SQL をサポート** 静的型 Java API を使用すると、コンポーネント・ベースのインタフェースを使用する場合よりも、さまざまな種類の SQL を開発アプリケーションに使用できます。

静的型 Java API には、次のような制約があります。

- **複雑な開発モデル** Ultra Light データベースのスキーマを保持するリファレンス・データベースを使用し、さらに開発アプリケーションに合わせたクラスを生成する必要がある場合、静的型 Java API による開発プロセスは複雑なものになります。Native Ultra Light for Java などの Ultra Light コンポーネントを使用すると、開発プロセスは極めて簡単になります。
- **設計時に SQL を指定する必要がある** コンパイル時に定義された SQL 文しかアプリケーションに組み込むことができません。Ultra Light コンポーネントでは、SQL 文をいつでも動的に組み込むことができます。

開発モデルは、個々のプロジェクトの要件やプログラマのスキルと経験を考慮して選定してください。

第 2 章

チュートリアル：Java を使用したアプリケーションの構築

この章の内容

この章は、静的型 Java API を使用して Ultra Light アプリケーションを開発するプロセスについて理解していただくためのチュートリアルです。最初のセクションでは、非常に簡単なアプリケーションを構築する方法を解説します。2 番目のセクションでは、アプリケーションに同期を追加する方法を解説します。

Ultra Light データベースの開発プロセスと補足情報の概要については、「Ultra Light 静的型 Java アプリケーションの構築」69 ページを参照してください。

Ultra Light 静的型 Java アプリケーションの開発については、「静的型 Java API を使用したデータ・アクセス」27 ページを参照してください。

概要

このチュートリアルでは、Ultra Light Java を使用して非常に簡単なアプリケーションを構築する方法について説明します。このアプリケーションは Sun JDK を使って開発したコマンド・ライン・アプリケーションであり、*Ultra Light 9.0 Sample* データベースの ULProduct テーブルに格納されているデータに対してクエリを実行します。

このチュートリアルでは、Java ソース・ファイルを作成し、リファレンス・データベース内にプロジェクトを作成し、これらのソースを使用してアプリケーションを構築、実行します。レッスンの前半では、同期をサポートしないバージョンのアプリケーションについて説明します。同期の追加は、レッスンの後半で行います。

このチュートリアルを行うには、必ず Java Development Kit をインストールしておいてください。

概要

最初のレッスンでは、次のタスクを行うアプリケーションを記述し構築します。

1. 単一テーブルからなる Ultra Light データベースに接続します。このテーブルは、Ultra Light サンプル・データベースの ULProduct テーブルのサブセットです。
2. テーブルにローを挿入します。初期データは通常、統合データベースと同期することで Ultra Light アプリケーションに追加されます。同期については、この章の後の方で追加します。
3. テーブルのローを標準出力に書き出します。

アプリケーションを構築するには、次の手順を実行します。

1. Adaptive Server Anywhere リファレンス・データベースを作成します。

   ここでは Ultra Light サンプル・データベース (CustDB) を使用します。

2. リファレンス・データベースに、アプリケーションで使用する SQL 文を追加します。

3. Ultra Light ジェネレータを実行して、Ultra Light データベースに書き出す Java コードと追加のソース・ファイルを生成します。

   Ultra Light ジェネレータは、アプリケーションで使用できる形式で、SQL 文の入った *.java* ファイルと、クエリを実行するコードの入った *.java* ファイルを書き出します。

4. アプリケーションの論理を実装するソース・コードを記述します。

   ここでは、ソース・コードは *Sample.java* という名前の単一ファイルです。

5. アプリケーションをコンパイルし、実行します。

次のレッスンでは、アプリケーションに同期を追加します。

ファイルを格納するディレクトリの作成

このチュートリアルでは、ソース・ファイルや実行可能ファイルを含む一連のファイルを作成します。これらのファイルを格納するディレクトリを作成してください。また、Ultra Light サンプル・データベースのコピーを作成して、実際の作業はそのコピーで行い、オリジナルのサンプル・データベースは他のプロジェクトでも使用できるように保管しておいてください。

このチュートリアルで使用するファイルのコピーは、SQL Anywhere ディレクトリのサブディレクトリ *Samples¥UltraLite¥JavaTutorial* にあります。

❖ チュートリアル・ディレクトリを準備するには、次の手順に従います。

1　作成するファイルを格納するディレクトリを作成します。以降のチュートリアルでは、このディレクトリを *c:¥JavaTutorial* とします。

2 Ultra Light 9.0 サンプル・データベースのバックアップ・コピーを作成し、チュートリアル・ディレクトリに保存します。Ultra Light 9.0 サンプル・データベースは *custdb.db* ファイルです。このファイルは、SQL Anywhere インストール・ディレクトリのサブディレクトリ *Samples¥UltraLite¥CustDB* にあります。このチュートリアルでは、オリジナルの Ultra Light 9.0 サンプル・データベースを使用します。チュートリアル終了後には、元のバージョンのサンプル・データベースを *APITutorial* ディレクトリから元の位置へコピーしなおします。

レッスン 1：リファレンス・データベースに SQL 文を追加する

このチュートリアルで使うリファレンス・データーベースは、Ultra Light 9.0 サンプル・データベースです。後の手順で、統合データベースと同じデータベースを同期に使用します。用途が別なので、実際のプロジェクトでは、リファレンス・データベースと統合データベースの役割に合った別々のデータベースを使うことができます。

ul_add_statement ストアド・プロシージャを使用して、リファレンス・データベースに SQL 文を追加します。この簡単なアプリケーションでは、次の文を使用します。

- **Insert**　INSERT 文は、データの初期コピーを ULProduct テーブルに追加します。この文は、アプリケーションに同期を追加するときには必要ありません。
- **Select**　SELECT 文は、ULProduct テーブルに問い合わせます。

SQL 文を追加するときは、この文を Ultra Light プロジェクトに関連付けてください。ここで使用するプロジェクト名は Product です。文の名前も追加してください。名前は大文字を使用する規則になっています。

❖ リファレンス・データベースに SQL 文を追加するには、次の手順に従います。

1. Sybase Central を起動し、Adaptive Server Anywhere プラグインを使用して Ultra Light 9.0 サンプル・データ・ソースに接続します。
   a. [スタート] ー [プログラム] ー [SQL Anywhere 9] ー [Sybase Central] を選択します。Sybase Central が表示されます。
   b. [ファイル] メニューから [ツール] ー [接続] を選択します。[新しい接続] ダイアログが表示されます。

c. Adaptive Server Anywhere プラグインを選択して [OK] をクリックします。[接続] ダイアログが表示されます。

d. [ID] タブで、[ODBC データ・ソース名] を選択し、**UltraLite 9.0 Sample** と入力します。

2 データベースにプロジェクトを追加します。

a. Sybase Central で、custdb データベースを開きます。

b. [Ultra Light プロジェクト] フォルダを開きます。

フォルダにはすでに、サンプル・アプリケーションで使用する custapi プロジェクトが入っています。新しいプロジェクトを作成してください。

c. [ファイル] － [新規] － [Ultra Light プロジェクト] を選択します。

d. プロジェクト名に **Product** と入力して、[完了] をクリックします。

3 Product プロジェクトに INSERT 文を追加します。

a. [Product] をダブルクリックして、プロジェクトを開きます。

b. [ファイル] － [新規] － [Ultra Light 文] を選択します。

c. 文の名前に **InsertProduct** と入力します。[次へ] をクリックします。

d. 次の文を入力します。

```
INSERT INTO ULProduct ( prod_id, price,
prod_name)
 VALUES (?,?,?)
```

SQL 文の中の疑問符は、ULProduct テーブルに挿入される値のプレースホルダです。これらの値は、実行時に入力します。

e. [完了] をクリックして操作を完了します。

Sybase Central でのこの操作は、次のストアド・プロシージャ・コールを実行するのと同等の操作です。

```
call ul_add_statement('Product',
     'InsertProduct',
     'INSERT INTO ULProduct( prod_id, price,
prod_name)
      VALUES (?,?,?) ' )
```

ul_add_statement の最初の引数はプロジェクト名、2 番目は文の名前、3 番目は SQL 文そのものです。

4　Product プロジェクトに SELECT 文を追加します。

a. Product プロジェクトから、[ファイル] － [新規] － [Ultra Light 文] を選択します。

b. 文の名前に **SelectProduct** と入力します。[次へ] をクリックします。

c. 次の文を入力します。

```
SELECT prod_id, prod_name, price
FROM ULProduct
```

d. [完了] をクリックして操作を完了します。

Sybase Central でのこの操作は、次のストアド・プロシージャ・コールを実行するのと同等の操作です。

```
call ul_add_statement('Product',
     'SelectProduct',
     'SELECT prod_id, prod_name, price FROM
ULProduct')
```

5　Sybase Central を閉じます。

これでデータベースに SQL 文が追加され、Ultra Light データベースを生成する準備ができました。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「ul_add_project システム・プロシージャ」および『Ultra Light データベース・ユーザーズ・ガイド』>「ul_add_statement システム・プロシージャ」を参照してください。

レッスン 2：Ultra Light ジェネレータの実行

Ultra Light ジェネレータは、2 つの Java ファイルを書き出します。1 つ目のファイルには、インタフェース定義として SQL 文が保管されています。ファイル名は *ISampleSQL.java* です。このインタフェース定義はメインのアプリケーション・コードで使用できます。2 つ目のファイルには、クエリとデータベースを実装するコードが格納されています。ファイル名は *SampleDB.java* です。

❖ Ultra Light データベース・コードを生成するには、次の手順に従います。

1 コマンド・プロンプトを開き、*JavaTutorial* ディレクトリに移動します。

2 Ultra Light ジェネレータを、次の引数で実行します (すべて 1 行に入力)。

```
ulgen -a -t java -c "dsn=UltraLite 9.0 Sample"
-j Product -s ISampleSQL -f SampleDB
```

引数には次の意味があります。

- **-a** SQL 文字列の名前を大文字で生成します。**InsertProduct** 文と **SelectProduct** 文は、**INSERT_PRODUCT** と **SELECT_PRODUCT** になります。
- **-t** 生成されたコードの言語です。C コードの代わりに Java コードを生成します。
- **-c** データベースに接続するための接続文字列です。
- **-j** Ultra Light プロジェクト名。この名前は、データベースに SQL 文を追加したときに入力したプロジェクト名と一致します。ジェネレータが生成するのは、このプロジェクトに関連する文のコードのみです。
- **-s** SQL 文を文字列として含むインタフェースの名前です。

- **-f** 生成されたデータベース・コードとクエリ実行コードを格納するファイルの名前です。

レッスン 3：アプリケーション・コードの記述

次のコード・リストは、非常に簡単な Ultra Light のアプリケーションを示します。

このコードを新規ファイルにコピーして *c:¥JavaTutorial* ディレクトリに *Sample.java* として保存することも、新規ファイルを開いて内容を入力することもできます。

```
// (1) Import required packages
 import java.sql.*;
 import ianywhere.ultralite.jdbc.*;
 import ianywhere.ultralite.runtime.*;

// (2) Class implements the interface containing SQL
statements
 public class Sample implements ISampleSQL
 {
   public static void main( String[] args )
   {
     try{
         // (3) Connect to the database
         java.util.Properties p = new
             java.util.Properties();
         p.put( "persist", "file" );
         SampleDB db = new SampleDB( p );
         Connection conn = db.connect();
         // (4) Initialize the database with data
         PreparedStatement pstmt1 =
             conn.prepareStatement( INSERT_PRODUCT );
         pstmt1.setInt(1, 1);
         pstmt1.setInt(2, 400);
         pstmt1.setString(3, "4x8 Drywall x100");
         int rows1=pstmt1.executeUpdate();
         pstmt1.setInt(1, 2);
         pstmt1.setInt(2, 3000);
         pstmt1.setString(3, "8' 2x4 Studs x1000");
         int rows2=pstmt1.executeUpdate();
        // (5) Query the data and write out the results
         Statement stmt = conn.createStatement();
         ResultSet result = stmt.executeQuery(
              SELECT_PRODUCT );
         while( result.next() ) {
             int id = result.getInt( 1 );
             String name = result.getString( 2 );
```

```
                int price = result.getInt( 3 );
                System.out.println ( name +
                    " ¥tId=" + id +
                    " ¥tPrice=" + price );
            }
            // (6) Close the connection to end
            conn.close();
        } catch (SQLException e) {
            Support.printException( e );
        }
    }
}
```

サンプル・プログラムの説明

単純すぎて便利とは言えませんが、この例のコードには、データベース・アクセスで使用するすべての Java プログラムに不可欠な要素が入っています。以下に、このサンプル・プログラムの主な要素を解説します。自分で Ultra Light アプリケーションを作成するときには、この手順をガイドとして使用してください。

各手順の番号は、ソース・コード内のコメントの番号に対応しています。

1. 必要なパッケージをインポートします。

   サンプル・プログラムでは JDBC インタフェースとクラスを使用しているため、そのパッケージをインポートします。また Ultra Light ランタイム・クラスと、SQL 文の文字列を含む生成されたインタフェースも必要です。

2. クラスを定義します。

   アプリケーションで使用する SQL 文は、インタフェースとして別々のファイルに格納されます。クラスでは、そのプロジェクトの SQL 文が使用できるように、インタフェースを実装すると宣言してください。クラス名は、データベースに文を追加したときに入力した文の名前に基づいて決定されます。

3. データベースに接続します。

接続は、データベース・クラスのインスタンスを使用して確立されます。データベース名は、生成された Java クラス (この場合は **SampleDB**) の名前と一致させてください。`persist` Properties オブジェクトの `file` 値は、そのデータベースが永続的であることを示します。

4. サンプル・データを挿入します。

   実際のアプリケーションでは、一般的にサンプル・データは挿入しません。代わりに、同期によってデータの初期コピーを取得します。開発の初期段階では、直接データを挿入することで作業が簡単になります。

   - **prepareStatement()** メソッドを使用して、**PreparedStatement** オブジェクトを作成します。
   - SQL コマンドを実行するには、**Statement** または **PreparedStatement** オブジェクトを作成します。**Statement** オブジェクトを使用するとパラメータのない単純な SQL コマンドが実行され、**PreparedStatement** オブジェクトを使用するとパラメータのある SQL コマンドが実行されます。サンプル・プログラムではまず、挿入コマンドを実行するための **PreparedStatement** オブジェクトを作成します。

     ```
     PreparedStatement pstmt1 =
     conn.prepareStatement( INSERT_PRODUCT );
     ```

     prepareStatement メソッドは引数として SQL 文字列を取ります。この SQL 文字列は、生成されたインタフェースから取り込まれます。

5. Statement オブジェクトを使用して、select SQL コマンドを実行します。

   - **createStatement()** メソッドを使用して、**Statement** オブジェクトを作成します。

     PreparedStatement オブジェクトとは異なり、**Statement** オブジェクトの作成時に SQL 文を入力する必要はありません。したがって、単一の **Statement** オブジェクトを使用して 2 つ以上の SQL 文を実行することができます。

     ```
     Statement stmt = conn.createStatement();
     ```

- SQL クエリを実行します。

  executeQuery() メソッドを使用して、選択クエリを実行します。select 文は、**ResultSet** オブジェクトを返します。

- ループを実行して、クエリ結果を順次取得します。

  ResultSet オブジェクトは、カーソルが最初のローの直前を指すようにしています。**next()** メソッドが呼び出されるたびに、カーソルは 1 ローずつ移動していきます。データのあるローにカーソルが移動すると、**next()** メソッドは正の値を返し、最後のローを越えると負の値を返します。

  ```
  while(result.next()) {
   ...
  }
  ```

- **get*xxx*()** メソッドを使用して、クエリ結果を取得します。

  これらのメソッドには、引数としてカラム番号を入力します。サンプル・プログラムでは、**getInt()** メソッドを使用して製品 ID と価格をそれぞれ 1 つ目と 2 つ目のカラムから取得し、**getString()** メソッドを使用して 3 つ目のカラムから製品名を取得します。

  ```
  int id = result.getInt( 1 );
  int price = result.getInt( 2 );
  String name = result.getString( 3 );
  ```

6. プログラムを終了します。

   - **Connection.close()** メソッドを使用してデータベースへの接続を終了します。

     ```
     conn.close();
     ```

レッスン 4：アプリケーションの構築と実行

前の項で、サンプル・コードを使用してソース・ファイル *Sample.java* を作成しました。これで、Ultra Light アプリケーションを構築する準備が整いました。

❖ アプリケーションを構築するには、次の手順に従います。

1. Adaptive Server Anywhere パーソナル・データベース・サーバを起動します。

   データベース・サーバを起動すると、Ultra Light ジェネレータがリファレンス・データベースにアクセスできるようになります。次のように、[スタート] メニューからデータベース・サーバを起動します。

   [スタート] ー [プログラム] ー [SQL Anywhere 9] ー [Ultra Light] ー [Ultra Light のパーソナル・サーバのサンプル] の順に選択してください。

2. Java ソース・ファイルをコンパイルします。

   classpath に、次のロケーションを含めます。

   - 現在のディレクトリ (classpath ではドットを使用)。
   - Java ランタイム・クラス。JDK 1.2 の場合は、クラスパスに *jre¥lib¥rt.jar* ファイルを含めます。JDK 1.1 の場合は、Java のインストール環境から *classes.zip* ファイルを含めます。
   - Ultra Light ランタイム・クラス。これらのクラスは、次のロケーションにあります。

     ```
     %ASANY9%¥UltraLite¥java¥lib¥ulrt.jar
     ```

     %ASANY9% は、SQL Anywhere のディレクトリです。

   次のように、*javac* コンパイラを実行します。

   ```
   javac *.java
   ```

これで、アプリケーションを実行する準備ができました。

❖ アプリケーションを実行するには、次の手順に従います。

1　コマンド・プロンプトを開き、*Javatutorial* ディレクトリに移動します。

2　classpath に、前の手順と同じクラスを含めます。

3　次のコマンドを入力してアプリケーションを実行します。

```
java Sample
```

2 項目のリストが画面に出力され、アプリケーションが終了します。

これで、初めての Ultra Light Java アプリケーションの構築と実行が終了しました。次の手順では、アプリケーションに同期を追加します。

レッスン 5：アプリケーションへの同期の追加

自分のプログラムが正常に機能していることをテストしたら、ULProduct テーブルへのデータ入力の自動化を制限するコードを削除します。これらの文を **JdbcConnection.synchronize()** 関数の呼び出しに置き換え、リモート・データベースと統合データベースを同期します。このプロセスによってテーブルにデータが入力されるので、引き続き選択クエリを実行することができます。

同期を追加すると、実際にコードが単純化されます。*Sample.java* の初期バージョンには次のコードが含まれています。これらのコードにより、Ultra Light データベースにデータが挿入されます。

```
PreparedStatement pstmt1 = conn.prepareStatement(
ADD_PRODUCT_1 );
pstmt1.setInt(1, 1);
pstmt1.setInt(2, 400);
pstmt1.setString(3, "4x8 Drywall x100");
int rows1=pstmt1.executeUpdate();
pstmt1.setInt(1, 2);
pstmt1.setInt(2, 3000);
pstmt1.setString(3, "8' 2x4 Studs x1000");
int rows2=pstmt1.executeUpdate();
```

このコードは、アプリケーションにデータの初期セットを入力するために含まれています。実際のアプリケーションでは、ソース・コードからデータの初期コピーを挿入するのではなく、同期を実行します。

❖ アプリケーションに同期を追加するには、次の手順に従います。

1. レッスン 4 で生成したデータベース (udb ファイル) を削除します。

   このレッスンでは、データは同期によって追加されるため、クリーンな Ultra Light データベースが必要です。

2. ハードエンコードされた挿入を、同期の呼び出しで置き換えます。

   a. コードを挿入する上記の命令を削除します。

b. 代わりに次のコードを追加します。

```
UlSynchOptions synch_opts = new
UlSynchOptions();
synch_opts.setUserName( "50" );
synch_opts.setPassword( "pwd50" );
synch_opts.setScriptVersion( "ul_default" );
synch_opts.setStream( new UlSocketStream() );
synch_opts.setStreamParms( "host=localhost" );
( (JdbcConnection)conn ).synchronize(
synch_opts );
```

ULSocketStream 引数はアプリケーションに対して、Mobile Link ユーザ名として 50 を使用して、TCP/IP 経由で現在のマシン (**localhost**) の Mobile Link 同期サーバに同期するよう指示します。

3 アプリケーションのコンパイルとリンクを行います。

CLASSPATH に現在のディレクトリ、Ultra Light ランタイム・クラス、Java ランタイム・クラスを含め、次のコマンドを入力します。

```
javac *.java
```

4 サンプル・データベースに対して動作する Mobile Link 同期サーバを起動します。

JavaTutorial ディレクトリのコマンド・プロンプトで、次のコマンドを入力します。

```
start dbmlsrv9 -c "dsn=UltraLite 9.0 Sample"
```

5 アプリケーションを実行します。

JavaTutorial ディレクトリのコマンド・プロンプトで、次のコマンドを入力します。

```
java Sample
```

アプリケーションが接続し、同期してデータを取得し、情報をコマンド・ラインに書き出します。この出力は、次のようになります。

```
Connecting to server:port = localhost(a.b.c.d):2439
 4x8 Drywall x100         Id=1    Price=400
 8' 2x4 Studs x1000       Id=2    Price=3000
 Drywall Screws 10lb      Id=3    Price=40
 Joint Compound 100lb     Id=4    Price=75
 Joint Tape x25x500       Id=5    Price=100
 Putty Knife x25          Id=6    Price=400
 8' 2x10 Supports x 200   Id=7    Price=3000
 400 Grit Sandpaper       Id=8    Price=75
 Screwmaster Drill        Id=9    Price=40
 200 Grit Sandpaper       Id=10   Price=100
```

このレッスンでは、簡単な Ultra Light アプリケーションに同期を追加しました。

JdbcConnection.synchronize() 関数の詳細については、「synchronize メソッド」81 ページを参照してください。

レッスン 6：行った変更の取り消し

チュートリアルを終了するには、Mobile Link 同期サーバを停止し、Ultra Light 9.0 サンプル・データベースをリストアしてください。

❖ チュートリアルを終了するには、次の手順に従います。

1. Mobile Link 同期サーバを終了します。
2. Ultra Light 9.0 サンプル・データベースをリストアします。
   - SQL Anywhere ディレクトリの *Samples¥UltraLite¥custdb* サブディレクトリにある *custdb.db* ファイルと *custdb.log* ファイルを削除します。
   - *Javatutorial* ディレクトリから、*custdb.db* ファイルを *Samples¥UltraLite¥custdb* ディレクトリにコピーします。
3. Ultra Light データベースを削除します。
   - Ultra Light データベースは JAR ファイルと同じディレクトリにあり、*.udb* という拡張子が付いています。次にアプリケーションを実行すると、新しいデータベースが初期化されます。

第 3 章

静的型 Java API を使用したデータ・アクセス

この章の内容

この章では、静的型 Java API に固有の Ultra Light 開発プロセスについて詳細に説明します。Java を使用して Ultra Light アプリケーションを作成する方法と、Ultra Light アプリケーションの構築と配備について説明します。

概要

Ultra Light アプリケーションは、データベース・アクセスのために JDBC を使用して Java 言語で作成できます。

Java 用 Ultra Light 開発プロセスは、C/C++ 静的開発モデルのプロセスと似ています。詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light 静的インタフェースの使用」を参照してください。

この章では、Ultra Light 静的型 Java アプリケーション特有のアプリケーション開発面のみを説明します。Java と JDBC についての基礎知識があることを前提とします。

Ultra Light 静的型 Java サンプル・アプリケーション

この項では、Ultra Light 静的型 Java バージョンの CustDB サンプル・アプリケーションをコンパイルして実行する方法について説明します。

サンプル・アプリケーションは、SQL Anywhere ディレクトリのサブディレクトリ *Samples¥UltraLite¥CustDB¥java* に格納されています。

アプレット・バージョンのサンプルでは、Sun アプレットビューアが使用され、ファイル *custdb.html* を表示します。このファイルには、簡単な <APPLET> タグが入っています。

アプレットビューアのセキュリティ上の制限のため、アプレットは、ファイル・システムから実行するのではなく Web サーバからダウンロードする必要があります。これは、ソケット接続を許可して同期に成功するためです。

アプリケーション・バージョンの CustDB は、データをファイルに永続的に保持しますが、アプレット・バージョンの場合は永続性は使用されません。

非常によく似た機能を持つ C/C++ バージョンのアプリケーションの概要については、『Ultra Light データベース・ユーザーズ・ガイド』>「チュートリアル：CustDB サンプル Ultra Light アプリケーション」を参照してください。

Ultra Light 静的型 Java サンプル・ファイル

Ultra Light 静的型 Java サンプル・アプリケーションのコードは、SQL Anywhere ディレクトリの *Samples¥UltraLite¥CustDB¥java* サブディレクトリに格納されています。

このディレクトリにあるファイルは、次のとおりです。

- **データ・アクセス・コード**　*CustDB.java* には、Ultra Light 専用のデータ・アクセス論理が保持されています。SQL 文は、*SQL.sql* に格納されています。

- **ユーザ・インタフェース・コード** *DialogDelOrder.java*、*Dialogs.java*、*DialogNewOrder.java*、*DialogUserID.java* には、ユーザ・インタフェース機能が保持されています。
- **readme.txt** リリースによって異なる、サンプルについての詳細な情報が記載されたテキスト・ファイルです。
- **サブディレクトリ** サンプルを実行できるサブディレクトリは 2 つあります。ディレクトリ *java11* が Java 1 用であり、*java13* が Java 2 用です。JDK バージョン 1.1.x を使用している場合には java11 を現在のディレクトリにし、JDK 1.2.x 以降を使用している場合には java13 を現在のディレクトリにしてください。これらのサブディレクトリには、サンプルを実行するバッチ・ファイルがあります。いずれのディレクトリでも、バッチ・ファイルは JAVA_HOME 環境変数に依存します。この環境変数は JDK を含むディレクトリに設定してください。次に例を示します。

  ```
  SET JAVA_HOME=c:¥jdk1.3.1
  ```

- **アプリケーション構築用のバッチ・ファイル** *build.bat* と *clean.bat* は、それぞれアプリケーションをコンパイルしたり、ソース・ファイル以外のすべてのファイルを削除したりします。
- **サンプルをアプリケーションとして実行するためのファイル** *Application.java* には、サンプルを Java アプリケーションとして実行するために必要な命令が入っています。また、*run.bat* はサンプル・アプリケーションを実行します。
- **サンプルをアプレットとして実行するためのファイル** *Applet.java* ファイルには、サンプルを Java アプレットとして実行するために必要な命令が入っています。また、*avweb.bat* は、Web ページとして *custdb.html* を使用し、アプレットビューアを使ってサンプル・アプレットを実行します。

サンプルをアプレットとして実行するには、Web サーバをインストールして起動します。アプレットは、アプレットビューア・ユーティリティか Web ブラウザを使用して実行できます。詳細については、*Samples¥UltraLite¥CustDB¥Java¥readme.txt* ファイルを参照してください。

Ultra Light 静的型 Java サンプル・アプリケーションの構築

この項では、Sun Java 1 または 2 の環境で Ultra Light 静的型 Java サンプル・アプリケーションを構築する方法について説明します。

❖ Ultra Light 静的型 Java サンプルを構築するには、次の手順に従います。

1　JDK が正しいことを確認します。

サンプル・アプリケーションを構築するには JDK 1.1 または JDK 1.3 が必要です。また、JDK ツールをパスに指定してください。

2　コマンド・プロンプトを開きます。

3　SQL Anywhere ディレクトリの *Samples¥UltraLite¥CustDB¥java¥java13* サブディレクトリに変更します。Java 1 を使用している場合には、*java11* ディレクトリに変更します。

4　サンプルを構築します。

- JAVA_HOME 環境変数を設定します。次に例を示します。

```
SET JAVA_HOME=c:¥jdk1.3.1
```

- コマンド・プロンプトから次のコマンドを入力します。

```
build
```

構築プロシージャは、次の操作を実行します。

- SQL 文を Ultra Light サンプル・データベースにロードする。

  この手順では、Interactive SQL と *SQL.sql* ファイルを使用します。また、Ultra Light 9.0 サンプル・データ・ソースに依存します。

- Java データベース・クラス **custdb.Database** を生成する。

この手順では、Ultra Light ジェネレータと Ultra Light 9.0 サンプル・データ・ソースを使用します。

- Java ファイルをコンパイルする。

  この手順では、JDK コンパイラ (*javac*) と *jar* ユーティリティを使用します。

Ultra Light 静的型 Java サンプル・アプリケーションの実行

サンプル・アプリケーションは、Java アプリケーションまたはアプレットとして実行できます。いずれの場合も、サンプル・アプリケーションを実行しているマシン上で動作する Mobile Link 同期サーバを起動して、サンプルを実行するための準備をしてください。

❖ サンプルを実行するための準備をするには、次の手順に従います。

- Ultra Light サンプル・データベース上で動作する Mobile Link 同期サーバを起動します。

  [スタート] ー [プログラム] ー [SQL Anywhere 9] ー [Mobile Link] ー [同期サーバのサンプル] を選択します。

❖ サンプルをアプリケーションとして実行するには、次の手順に従います。

1. *Samples¥UltraLite¥CustDB¥java¥java13* ディレクトリ (Java 1 を使用している場合には、*java11* ディレクトリ) のコマンド・プロンプトを開きます。
2. サンプルを実行します。

   - JAVA_HOME 環境変数を設定します。次に例を示します。

     ```
     SET JAVA_HOME=c:¥jdk1.3.1
     ```

   - 次のコマンドを入力します。

```
run
```

アプリケーションが起動して、[Enter ID] ダイアログが表示されます。

3　従業員 ID を入力します。

従業員 ID として 50 を入力し、[OK] をクリックします。

[UltraLite Customer Demonstration] ウィンドウが表示されます。以前にサンプルをアプリケーションまたはアプレットとして実行したことがある場合は、データベースにデータがあります。

4　データベースにデータがない場合は、同期をとります。

[Actions] メニューから、[Synchronize] を選択します。アプリケーションが同期して、ウィンドウに注文が表示されます。

これで、データベース内のデータに対する操作を実行できます。

サンプル・データベースとサンプルで示された Ultra Light の機能については、『Ultra Light データベース・ユーザーズ・ガイド』>「チュートリアル：CustDB サンプル Ultra Light アプリケーション」を参照してください。

❖ アプレットビューアを使用してサンプルをアプレットとして実行するには、次の手順に従います。

1　Web サーバを起動して、適切なサブディレクトリがそのサーバのデフォルト・ディレクトリとして設定されているのか、または仮想ディレクトリの 1 つとして設定されているのかを確認します。

2　*samples¥UltraLite¥CustDB¥java¥java13* ディレクトリ (Java 1 を使用している場合には、*java11* ディレクトリ) でコマンド・プロンプトを開きます。

3　次のコマンドを入力します。

```
avapplet
```

- アプレットが起動して、従業員 ID を入力するフィールドが表示されます。

4 従業員 ID を入力します。

従業員 ID として 50 を入力し、[OK] をクリックします。

[UltraLite Customer Demonstration] ウィンドウが表示されます。初めてサンプルを実行するときは、データベースにデータはありません。以前にサンプルをアプリケーションまたはアプレットとして実行したことがある場合は、データベースにデータがあります。

5 アプリケーションを同期します。

[Actions] メニューから、[Synchronize] を選択します。アプリケーションが同期して、ウィンドウに注文が表示されます。

これで、データベース内のデータに対する操作を実行できます。

❖ Web ブラウザを使用して、サンプルをアプレットとして実行するには、次の手順に従います。

1 Web サーバを起動して、適切なサブディレクトリがそのサーバのデフォルト・ディレクトリとして設定されているのか、または仮想ディレクトリの 1 つとして設定されているのかを確認します。

2 Web ブラウザを起動して、*Samples¥UltraLite¥CustDB¥java¥custdb.htm* ファイルの URL を入力します。

- アプレットが起動して、従業員 ID を入力するフィールドが表示されます。

3 従業員 ID を入力します。

従業員 ID として 50 を入力し、[OK] をクリックします。

[UltraLite Customer Demonstration] ウィンドウが表示されます。初めてサンプルを実行するときは、データベースにデータはありません。以前にサンプルをアプリケーションまたはアプレットとして実行したことがある場合は、データベースにデータがあります。

4　アプリケーションを同期します。

[Actions] メニューから、[Synchronize] を選択します。アプリケーションが同期して、ウィンドウに注文が表示されます。

サンプル・データベースとサンプルで示された Ultra Light の機能については、『Ultra Light データベース・ユーザーズ・ガイド』>「チュートリアル：CustDB サンプル Ultra Light アプリケーション」を参照してください。

サンプルのリセット

コンパイルしたファイル、サンプル・データベース、生成したコードをすべて削除するには、*clean.bat* ファイルを実行します。

Ultra Light データベースへの接続と設定

この項では、Ultra Light データベースに接続する方法について説明します。データベースへの接続に推奨される Ultra Light メソッドについて説明します。また、標準の JDBC 接続モデルを使用して接続する方法も説明します。

Ultra Light データベースへの接続には、ユーザ ID やパスワードは不要です。詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light のユーザ認証」を参照してください。

Ultra Light 静的型 Java データベースは、「**永続的**」(アプリケーションを閉じるとファイルに格納される) にも、「**一時的**」(アプリケーションを閉じると消える) にもなります。デフォルトでは、一時的です。

Ultra Light データベースに接続するときは、データベースの永続性を設定します。この項では、Ultra Light データベースを設定する方法について説明します。

Ultra Light JdbcDatabase.connect メソッドの使用

生成された Ultra Light データベース・コードは、**JdbcDatabase** を拡張したクラスの形式になります。この JdbcDatabase には、接続を確立する **connect** メソッドが含まれています。

次の例は、**SampleDB** という名前の生成されたデータベース・クラスの一般的なコードを示します。

```
try {
     SampleDB db = new SampleDB();
     java.sql.Connection conn = db.connect();
} catch( SQLException e ){
// error processing here
}
```

生成されたデータベースのクラスは、`-f` オプションを使用して Ultra Light ジェネレータのコマンド・ラインに指定されます。

永続的なデータベースを使用する場合は、接続に **Properties** オブジェクトとして特性を指定します。次の例は、一般的なコードを示します。

```
java.util.Properties p = new java.utils.Properties();
p.put( "persist", "file" );
p.put( "persistfile", "c:¥¥dbdir¥¥database.udb" );
SampleDB db = new SampleDB( p );
java.sql.Connection conn = db.connect( );
```

Properties は、データベース・コンストラクタで使用されます。接続間のデータベースの永続性モデルは変更できません。

2 つのプロパティによって、このデータベースが永続的であることと、ファイル *c:¥dbdir¥database.udb* にこのデータベースが格納されることが指定されます。

URL に指定できるプロパティの詳細については、「Ultra Light JDBC URL」39 ページ を参照してください。

詳細については、「Ultra Light 静的型 Java データベースの設定」41 ページおよび「生成されたデータベース・クラス」85 ページを参照してください。

JDBC ドライバのロードと登録

前の項で説明した Ultra Light **JdbcDatabase.connect()** メソッドは、Ultra Light データベースに接続する最も簡単な方法です。ただし、標準の JDBC を使って接続を確立することもできます。この項では、その方法について説明します。

Ultra Light アプリケーションは、JDBC ドライバを使用してデータベースに接続します。このドライバは、Ultra Light ランタイム・クラス (*ulrt.jar*) に入っています。使用するアプリケーションに JDBC ドライバをロードし登録してから、データベースに接続してください。ドライバをロードするには、**Class.forName()** メソッドを使用します。このメソッドは、ドライバのパッケージ名を引数として取ります。

```
Class.forName( "ianywhere.ultralite.jdbc.JdbcDriver");
```

JDBC ドライバは、ロード時に自動的に登録されます。

複数のドライバのロード

一般的には、それぞれのアプリケーションに登録されるドライバは 1 つだけです。しかし、1 つのアプリケーションに複数のドライバを登録できます。前述のメソッドと同じものを使用して各ドライバをロードします。**DriverManager** によって、データベースに接続するときにどのドライバを使用するかが決定されます。

getDriver メソッド

DriverManager.getDriver メソッドは、指定した URL の **Driver** を返します。

エラー処理

ドライバが見つからない場合の処理を行うには、次のように **ClassNotFoundException** を取得します。

```
try{
     Class.forName(
         "ianywhere.ultralite.jdbc.JdbcDriver");
 } catch(ClassNotFoundException e){
     System.out.println( "Exception: " + e.getMessage()
);
     e.printStackTrace();
}
```

JDBC を使用したデータベースへの接続

ドライバが宣言されると、標準の JDBC **DriverManager.getConnection** メソッドを使用してデータベースに接続できます。

getConnection のプロトタイプ

JDBC **DriverManager.getConnection** メソッドには、プロトタイプがいくつかあります。それらのプロトタイプは、次の引数を取ります。

```
DriverManager.getConnection( String url, Properties
info )
DriverManager.getConnection( String url )
```

Ultra Light ドライバでは、それぞれのプロトタイプがサポートされます。引数については、以降の項で説明します。

ドライバ・マネージャ

DriverManager クラスは、現在ロードされている **Driver** クラスのリストを管理します。リスト内の各ドライバに対して URL に接続可能かどうかを問い合わせます。接続可能なドライバが見つかると、**DriverManager** はそのドライバを使用してデータベースに接続しようとします。

エラー処理

接続を確立できない場合の処理を行うには、次のように **SQLException** を取得します。

```
try{
  Class.forName(
      "ianywhere.ultralite.jdbc.JdbcDriver");
  Connection conn = DriverManager.getConnection(
      "jdbc:ultralite:asademo" );
  } catch(SQLException e){
  System.out.println( "Exception: " + e.getMessage() );
  e.printStackTrace();
  }
```

Ultra Light JDBC URL

URL は、Ultra Light データベースへの接続に使用する **DriverManager.getConnection** メソッドの必須引数です。

接続方法の概要については、「JDBC を使用したデータベースへの接続」38 ページを参照してください。

Ultra Light JDBC URL の構文は、次のとおりです。

```
jdbc:ultralite:[database:persist:persistfile][;option=v
alue...]
```

すべてのコンポーネントで、大文字と小文字を区別します。各コンポーネントの意味は次のとおりです。

- **jdbc**　ドライバを JDBC ドライバとして識別します。このコンポーネントは必須です。
- **ultralite**　ドライバを Ultra Light ドライバとして識別します。このコンポーネントは必須です。
- **database**　データベースのクラス名。このコンポーネントは必須で、完全に修飾された名前でなければなりません。データベース・クラスがパッケージ内にある場合は、パッケージ名も含めて指定します。

  たとえば、`jdbc:ultralite:MyProject` という URL からは、`MyProject` というクラスがロードされます。

Java クラスは、それを定義している *.java* ファイルと名前を共有します。したがって、このコンポーネントは、Ultra Light ジェネレータの出力ファイル・パラメータと同じになります。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light ジェネレータ」を参照してください。

- **persist** データベースを永続的にするかどうかを指定します。デフォルトでは、一時的です。

  詳細については、「Ultra Light 静的型 Java データベースの設定」41 ページを参照してください。

- **persistfile** 永続的なデータベースの場合に、ファイル名を指定します。

  詳細については、「Ultra Light 静的型 Java データベースの設定」41 ページを参照してください。Ultra Light 静的型 Java プロパティは、C/C++ アプリケーションのプロパティと非常に似ています。これらのプロパティ名は、**persistfile** が **file_name** に対応することを除き、アンダースコア文字がないことだけが異なります。『Ultra Light C/C++ ユーザーズ・ガイド』>「UL_STORE_PARMS マクロ」を参照してください。

- **options** 次のオプションが提供されます。

  - **uid** ユーザ ID。
  - **pwd** ユーザ ID のパスワード。

また、Properties オブジェクトを使用して接続することもできます。次のプロパティを指定できます。各プロパティは、上記の明示的な URL 構文でも同じ意味を持ちます。

- database
- persist
- persistfile
- user

- password

Properties オブジェクトを使用した接続情報の格納

Properties オブジェクトを使用して、接続情報を格納できます。このオブジェクトは URL とともに **getConnection** の引数として指定することもできます。

接続方法の概要については、「JDBC を使用したデータベースへの接続」38 ページを参照してください。

「Ultra Light JDBC URL」39 ページ で説明した次の URL のコンポーネントを、URL の一部か、**Properties** オブジェクトのメンバとして指定できます。

- **persist**
- **persistfile**
- `jdbc:ultralite` コンポーネントは、必ず URL 内に指定します。

データベースを暗号化する場合は、**key** プロパティを指定します。詳細については、「Ultra Light データベースの暗号化」49 ページを参照してください。

複数のデータベースへの接続

Ultra Light 静的型 Java アプリケーションは、複数のデータベースに接続できます。複数のデータベースに接続するには、connection オブジェクトを複数作成するだけです。

詳細については、「JDBC を使用したデータベースへの接続」38 ページを参照してください。

Ultra Light 静的型 Java データベースの設定

Ultra Light 静的型 Java データベースについて、次のような設定ができます。

- データベースが一時的か永続的か。
- データベースが永続的な場合は、ファイル名を指定できる。
- データベースが一時的な場合は、初期化するデータベースに URL を指定できる。
- 暗号化キーを設定できる。

こうした動作は、データベースの URL に特別な値を指定するか、データベースの作成時に Properties オブジェクトを指定することで設定できます。暗号化キーは、URL では設定できませんが、Properties オブジェクトでは必ず設定します。

詳細については、「Ultra Light JdbcDatabase.connect メソッドの使用」36 ページおよび「Properties オブジェクトを使用した接続情報の格納」41 ページを参照してください。

一時的なデータベースと永続的なデータベース

デフォルトでは、Ultra Light 静的型 Java データベースは一時的です。一時的なデータベースは、データベース・オブジェクトがインスタンス化されると初期化され、アプリケーションが閉じると消去されます。次回アプリケーションが起動するときに、データベースを再度初期化する必要があります。

Ultra Light 静的型 Java データベースを永続的にするには、データベースをファイルに保管します。これを行うには、JDBC URL の **persist** 要素と **persistfile** 要素を指定するか、データベースの **connect** メソッドに Properties オブジェクト **persist** と **persistfile** を提供します。

一時的なデータベースの初期化

ほとんどの Ultra Light アプリケーションのデータベースは、最初の同期呼び出しのときに初期化されます。一時的なデータベースを使用する Ultra Light 静的型 Java アプリケーションの場合は、データベースを初期化するもう 1 つの方法があります。初期データベースとして使用される Ultra Light データベースの URL は、URL の **persistfile** コンポーネントに提供されます。

データベースの設定

URL のデータベース設定コンポーネントは、次のようになります。

- **persist**　次の値のいずれかを設定します。

- **none**　データベースを一時的なものにするときに使用します。データベースは、アプリケーションの実行中はメモリに格納されますが、アプリケーションが終了すると消えます。
- **file**　データベースをファイルとして保管するときに使用します。デフォルトのファイル名は *database.udb* です。*database* はデータベースのクラス名を表します。

デフォルト設定は **none** です。

- **persistfile**　このコンポーネントの意味は、**persist** の設定によって異なります。
  - **persist** コンポーネントの値が **none** の場合、**persistfile** コンポーネントは、データベースの初期化に使用される Ultra Light データベース・ファイルの URL になります。

    URL からのスキーマとデータの両方がアプリケーション・データベースの初期化に使用されますが、両者の間にはその他の関係はありません。アプリケーションが行ったすべての変更は一時的なデータベースのみに適用され、初期化データベースには適用されません。

    次に、JDBC URL の例を示します。

```
jdbc:ultralite:transient:none:http://www.address.com/
transient.udb
```

初期化データベースは、データベースの作成、同期、終了に永続的なフォームの URL を使用するアプリケーションで、準備できます。

- **persist** コンポーネントの値が **file** の場合、**persistfile** コンポーネントは、永続的な Ultra Light データベースのファイル名になります。ファイル名の拡張子はどのようなものでもかまいません (*.udb* など)。

Ultra Light 静的型 Java アプリケーションに SQL 文を含める

この項では、Ultra Light アプリケーションに SQL 文を追加する方法について説明します。

Ultra Light アプリケーションで使用可能または使用不可能な SQL の機能については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light の SQL サポートの概要」を参照してください。

アプリケーションで使用する SQL 文は、リファレンス・データベースに追加します。Ultra Light ジェネレータは、これらの SQL 文を **public static final strings** として定義するインタフェースを書き出します。インタフェースを実装して SQL 文を識別子で参照するか、またはインタフェースから直接 SQL 文を参照して、アプリケーションで文を呼び出します。

アプリケーション用の SQL 文の定義

Ultra Light アプリケーションに含める SQL 文と Ultra Light データベース自体の構造を定義するには、使用するアプリケーション用にリファレンス・データベースに SQL 文を追加します。

リファレンス・データベースについては、『Ultra Light データベース・ユーザーズ・ガイド』> 「リファレンス・データベースの準備」を参照してください。

プロジェクトの定義

リファレンス・データベースに保管される SQL 文は、それぞれ「**プロジェクト**」と対応します。プロジェクトは名前であり、リファレンス・データベースで定義されます。これにより、1 つのアプリケーションの SQL 文がグループ化されます。複数のプロジェクトを定義することにより、1 つのリファレンス・データベースに複数のアプリケーションの SQL 文を保管できます。

プロジェクトの作成については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light プロジェクトの作成」を参照してください。

プロジェクトへの文の追加

Ultra Light アプリケーションで使用するデータ・アクセス文は、プロジェクトに追加する必要があります。

データベースへの SQL 文の追加については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light プロジェクトに SQL 文を追加する」を参照してください。

アプリケーションへのユーザ認証の追加

Ultra Light のユーザ認証の詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light のユーザ認証」を参照してください。

ユーザの認証機能をオンにして Ultra Light データベースを作成すると、認証されたユーザが 1 人作成されます。ユーザ ID は **DBA** であり、パスワードは **SQL** です。Ultra Light では、16 文字より短いユーザ ID とパスワードを使用して、一度に最大 4 人のユーザを定義できます。各ユーザは正しく認証されると、データベースにアクセスできます。

Ultra Light におけるユーザ ID とパスワードの大文字と小文字の区別は、リファレンス・データベースによって決定されます。リファレンス・データベースの大文字と小文字を区別しない場合は (デフォルト)、Ultra Light データベースも大文字と小文字を区別しません (ユーザ認証を含む)。

ユーザ認証の有効化

ユーザの認証機能を有効にするには、Ultra Light データベースへの接続時に Ultra Light の有効なユーザ ID とパスワードをアプリケーションから提供してもらう必要があります。ユーザ認証を明示的に有効にしないと、Ultra Light はユーザ認証を行いません。

❖ ユーザ認証を有効にするには、次の手順に従います。

- **JdbcSupport.enableUserAuthentication** メソッドを呼び出してから新しいデータベース・オブジェクトを作成します。次に例を示します。

```
JdbcSupport.enableUserAuthentication();
java.util.Properties p = new java.util.Properties();
p.put( "persist", "file" );
SampleDB db = new SampleDB( p );
```

ユーザ認証を有効にしたら、使用するアプリケーションにユーザ管理コードを追加してください。詳細については、「ユーザ ID とパスワードの管理」47 ページを参照してください。

ユーザ ID とパスワードの管理

ユーザ認証の例

次のサンプル・コードは、Ultra Light Java アプリケーションのユーザ管理と認証を実行します。

完全なサンプルは、SQL Anywhere ディレクトリの *Samples¥UltraLite¥javaauth* サブディレクトリにあります。次のコードは、*Samples¥UltraLite¥javaauth¥Sample.java* を基にしています。

```
JdbcSupport.enableUserAuthentication();
// Create database environment
java.util.Properties p = new java.util.Properties();
p.put( "persist", "file" );
SampleDB db = new SampleDB( p );

// Get new user ID and password
try{
   conn = db.connect( "dba", "sql" );
   // Set user ID and password
   // a real application would prompt the user.
   uid = "50";
   pwd = "pwd50";

   db.grant( uid, pwd );
   db.revoke( "dba" );
   conn.close();
}
catch( SQLException e ){
   // dba connection failed - prompt for user ID and
password
   uid = "50";
   pwd = "pwd50";
}

// Connect
conn = db.connect( uid, pwd );
```

このコードでは、次のタスクを実行します。

1. データベース・オブジェクトを開く。
2. デフォルトのユーザ ID とパスワードを使用して接続する。
3. 接続に成功したら、新しいユーザを追加する。
4. Ultra Light データベースからデフォルト・ユーザを削除する。
5. 切断する。更新されたユーザ ID とパスワードがデータベースに追加される。
6. 更新されたユーザ ID とパスワードを使用して接続する。

データベース保管の設定と管理

Ultra Light 永続ストレージについて、次のような設定ができます。

- Ultra Light データベース・エンジンがキャッシュとして使用するメモリ容量
- データベースの暗号化
- ファイル・システム領域の事前割り付け
- データベースのファイル名
- データベース・ページ・サイズ

Ultra Light データベースの暗号化

デフォルトでは、Ultra Light データベースは、ディスク上でも永続的なメモリ内でも暗号化されません。データベース・ストア内のテキスト・カラムとバイナリ・カラムは、16 進エディタなどの表示ツールを使用すると、きちんと読むことができます。セキュリティを強化するために、次の 2 つのオプションが用意されています。

- **難読化**　データベースを読みにくくすることで、表示ツールでデータベース内のデータを直接表示しようとする単純な行為に対するセキュリティが提供されます。巧妙で容赦ないデータへのアクセスが試みられた場合は、このオプションでは防止できません。難読化は、パフォーマンスにほとんど影響しません。

  詳細については、「Ultra Light データベースの難読化」50 ページを参照してください。

- **高度な暗号化**　Ultra Light データベース・ファイルは、AES 128 ビット・アルゴリズムを使用して高度に暗号化できます。このアルゴリズムは、Adaptive Server Anywhere データベースを暗号化するために使用しているアルゴリズムと同じです。高度な暗号化を使用すると、巧妙で容赦ないデータへのアクセス試行に対抗するセキュリティが提供されますが、パフォーマンスに大きな影響を与えます。

警告
高度に暗号化されているデータベースの暗号化キーをなくしたり忘れてしまったりした場合は、データベースにアクセスできません。このような状況では、テクニカル・サポートでもデータベースにアクセスできません。アクセスできなくなったデータベースは、廃棄して、新しくデータベースを作成する必要があります。

詳細については、「Ultra Light データベースの暗号化」50 ページおよび「データベースの暗号化キーの変更」52 ページを参照してください。

Ultra Light データベースの難読化

❖ Ultra Light データベースを難読化するには、次の手順に従います。

- 次の行をコードに追加してから、データベースを作成します (つまり、最初にデータベースに接続する前に行います)。

```
UlDatabase.setDefaultObfuscation( true );
```

Ultra Light データベースの暗号化

Ultra Light データベースは、最初の接続時に作成されます。Ultra Light データベースを暗号化するには、暗号化キーを指定してから最初の接続を行います。最初の接続時に、指定したキーを使用することでデータベースが暗号化されます。以降の接続では、指定したキーと暗号化キーが照合され、一致しない場合は接続が失敗します。

❖ Ultra Light データベースを高度に暗号化するには、次の手順に従います。

1 最初にデータベース・オブジェクトを作成する前に、key という名前のプロパティを設定します。

次のコードは、コマンド・ラインから暗号化キーを読み込みます。

```
InputStreamReader isr = new InputStreamReader(
System.in );
BufferedReader br = new BufferedReader( isr );
String key = null ;
System.out.print( "Enter encryption key:" );
key = br.readLine() ;
System.out.println( "The key is: " + key );

// Connect to the database
java.util.Properties p = new java.util.Properties();
p.setProperty( "persist", "file" );
p.setProperty( "key", key );
SampleDB db = new SampleDB( p );
```

ここで、SampleDB は、Ultra Light ジェネレータの `-f` コマンド・ライン・オプションで指定されたデータベース・ファイル名です。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light ジェネレータ」および『Ultra Light 静的型 Java ユーザーズ・ガイド』>「Properties オブジェクトを使用した接続情報の格納」を参照してください。

2　プロパティを使用してデータベース・オブジェクトを作成します。

次に例を示します。

```
Connection conn = db.connect();
```

初回の接続後、間違ったキーを指定してデータベースにアクセスしようとすると、「暗号化キーが不正であるか、または見つかりません。」という SQLException が発生します。

暗号化を例示する Java サンプル・アプリケーションが、ディレクトリ *¥Samples¥UltraLite¥JavaSecurity* に保存されています。暗号化コードは、*¥Samples¥UltraLite¥JavaSecurity¥Sample.java* に保存されています。

このサンプル・コードの一部を次に示します。

```
// Obtain the encryption key
InputStreamReader isr = new InputStreamReader(
System.in );
BufferedReader br = new BufferedReader( isr );
String key = null ;
System.out.print( "Enter encryption key:" );
key = br.readLine() ;
System.out.println( "The key is: " + key );

java.util.Properties p = new java.util.Properties();
p.setProperty( "persist", "file" );
p.setProperty( "key", key );
SampleDB db = new SampleDB( p );
Connection conn = db.connect();
```

データベースの暗号化キーの変更

データベースの暗号化キーは変更できます。既存のキーを使用してアプリケーションをデータベースに接続してから、変更を行ってください。

警告
キーを変更すると、データベースのすべてのローは古いキーを使用して復号化され、新しいキーを使用して再度暗号化されます。この操作は回復不能です。操作の途中でアプリケーションが中断されると、データベースは無効になり、アクセスできなくなります。新しいデータベースを作成してください。

❖ Ultra Light データベースの暗号化キーを変更するには、次の手順に従います。

- 引数として新しいキーを指定して、データベース・オブジェクトで **changeEncryptionKey** を呼び出します。

  ```
  db.changeEncryptionKey( "new key" );
  ```

  詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「changeEncryptionKey メソッド」を参照してください。

Ultra Light データベースの断片化解除

Ultra Light の記憶領域は、空き領域を効率的に再使用するように設計されているので、通常の状況では明示的な断片化解除は必要ありません。この項では、Ultra Light データベースの断片化解除を明示的に実行するテクニックについて説明します。このテクニックは、領域要件が極端に厳しいアプリケーションで使用します。

Ultra Light には、データベースの一部分の断片化解除を行う断片化解除ステップ関数が用意されています。データベース全体の断片化解除を 1 回で行うには、**ul_true** が返されるまで断片化解除ステップ関数をループで呼び出します。この処理は負荷が高くなる可能性があります。また、エラーを検出するために SQLCODE をチェックしてください (ここでのエラーは、通常はファイル I/O エラーになります)。

明示的な断片化解除は、アプリケーションの制御下で、そのアイドル時間に徐々に行われます。それぞれのステップは、小さな処理です。

詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「JdbcDefragIterator クラス」を参照してください。

❖ Ultra Light データベースの断片化を解除するには、次の手順に従います。

1　**Connection** を **JdbcConnection** オブジェクトにキャストします。次に例を示します。

```
...
Connection conn = db.connect();
JdbcConnection jconn = (JdbcConnection)conn ;
```

Ultra Light には、データベースの一部分の断片化解除を行う断片化解除ステップ関数が用意されています。データベース全体の断片化解除を 1 回で行うには、**ul_true** が返されるまで断片化解除ステップ関数をループで呼び出します。この処理は負荷が高くなる可能性があります。また、エラーを検出するために SQLCODE をチェックしてください (ここでのエラーは、通常はファイル I/O エラーになります)。

2　**getDefragIterator()** を呼び出して、**JdbcDefragIterator** オブジェクトを取得します。次に例を示します。

```
JdbcDefragIterator defrag =
jconn.getDefragIterator();
```

3 アイドル時間中に、**ulStoreDefragStep()** を呼び出して、データベースの一部の断片化を解除します。

```
defrag.ulStoreDefragStep();
```

アプリケーションへの同期の追加

多くの Ultra Light アプリケーションにとって、同期は重要な機能です。この項では、アプリケーションに同期の機能を追加する方法を説明します。

Ultra Light アプリケーションを統合データベースの最新状態と同期する論理は、アプリケーション自体にはありません。統合データベースに格納されている同期スクリプトは、Mobile Link 同期サーバと Ultra Light ランタイム・ライブラリとともに、変更のアップロード時に変更をどのように処理するかを制御し、ダウンロードする変更はどれかを決定します。

概要

同期ごとの詳細は、同期パラメータのセットによって制御されます。これらのパラメータは、同期用関数呼び出しの引数として指定されるオブジェクトにまとめられます。このメソッドの概要は、どの開発モデルでも同じです。

❖ アプリケーションに同期を追加するには、次の手順に従います。

1. 同期パラメータが格納されたオブジェクトを初期化します。

   詳細については、「同期パラメータの初期化」56 ページを参照してください。

2. アプリケーションのパラメータ値を割り当てます。

   詳細については、『Mobile Link クライアント』>「Ultra Light 同期クライアントのネットワーク・プロトコルのオプション」を参照してください。

3. 同期関数を呼び出し、構造体またはオブジェクトを引数として指定します。

   詳細については、「同期を呼び出す」58 ページを参照してください。

同期するときに、コミットされていない変更がないことを確認してください。詳細については、「同期の前に変更をコミットする」60 ページを参照してください。

同期パラメータ

同期の定義は、同期パラメータのセットを介して制御されます。これらのパラメータの詳細については、『Mobile Link クライアント』>「Ultra Light 同期クライアントのネットワーク・プロトコルのオプション」を参照してください。

同期パラメータの初期化

同期パラメータは、Java オブジェクトに格納されます。Mobile Link 同期サーバの URL、使用するスクリプト・バージョン、Mobile Link ユーザ名などの同期の詳細はすべて、**UlSynchOptions** オブジェクトに格納されています。

同期パラメータの完全なリストについては、「同期パラメータ」90 ページを参照してください。

❖ 同期パラメータを初期化するには、次の手順に従います。

1. **UlSynchOptions** オブジェクトを作成します。次に例を示します。

```
UlSynchOptions opts = new UlSynchOptions();
```

2. 必須パラメータを設定します。

   UlSynchOptions() オブジェクトには、フィールドを設定して取得するための一連のメソッドがあります。メソッドのリストについては、『Mobile Link クライアント』>「同期パラメータ」を参照してください。これらのメソッドを使って必要な同期パラメータを設定してから同期を開始してください。次に例を示します。

```
opts.setUserName( "50" );
opts.setScriptVersion( "default" );
opts.setStream( new UlSocketStream() );
```

同期パラメータの設定

Ultra Light Java アプリケーションの同期ストリームはオブジェクトであり、コンストラクタによって設定されます。使用可能なストリームは次のとおりです。

- **UlSocketStream**　TCP/IP 同期。
- **UlSecureSocketStream**　Certicom 楕円曲線暗号化トランスポート・レイヤ・セキュリティを適用した TCP/IP 同期。
- **UlSecureRSASocketStream**　Certicom RSA トランスポート・レイヤ・セキュリティを適用した TCP/IP 同期。
- **UlHTTPStream**　HTTP 同期。
- **UlHTTPSStream**　HTTPS 同期。

次の行は、ストリームを TCP/IP に設定します。

```
synch_opts.setStream( new UlSocketStream() );
```

詳細については、『Mobile Link クライアント』> 「同期パラメータ」を参照してください。

別途ライセンスを取得できるオプションが必要

UlHTTPSStream、**UlSecureSocketStream**、**UlSecureRSASocketStream** を使用するには、Certicom テクノロジが必要です。このテクノロジを使用するには、別途ライセンスを取得できる SQL Anywhere Studio 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。このオプションの詳細については、『SQL Anywhere Studio の紹介』> 「SQL Anywhere Studio へようこそ」を参照してください。

各パラメータの詳細については、『Mobile Link クライアント』> 「Ultra Light 同期クライアントのネットワーク・プロトコルのオプション」を参照してください。

同期パラメータを初期化して、アプリケーションで必要な値に設定すると、**JdbcConnection.synchronize()** メソッドを使用して同期を開始することができます。

メソッドは、**UlSynchOptions** オブジェクトを引数として取ります。同期に必要な一連の呼び出しは次のとおりです。

```
UlSynchOptions opts = new UlSynchOptions();
opts.setUserName( "50" );
opts.setScriptVersion( "default" );
opts.setStream( new UlSocketStream() );
opts.setStreamParms( "host=123.45.678.90" );
conn.synchronize( opts );
```

同期を呼び出す

同期を呼び出す方法は、ターゲット・プラットフォームと同期ストリームによって細かく異なります。

同期処理が機能するのは、Ultra Light アプリケーションを実行するデバイスが同期サーバと通信できる場合だけです。プラットフォームによっては、デバイスを、クレードルに置くかまたはケーブルを使用してサーバ・コンピュータに接続して、物理的に接続する必要があります。同期が実行できない場合には、アプリケーションにエラー処理コードを追加する必要があります。

❖ 同期を呼び出すには、次の手順に従います (TCP/IP、HTTP、または HTTPS ストリームの場合)。

新しい **ULSynchInfo** オブジェクトを構成して同期パラメータを初期化し、**JdbcConnection.synchronize()** を呼び出して同期を行います。『Ultra Light 静的型 Java ユーザーズ・ガイド』>「アプリケーションへの同期の追加」を参照してください。

同期呼び出しでは、その同期固有の情報が記述されたパラメータのセットを保持している構造体が必要です。使用される特定のパラメータは、ストリームによって異なります。

トランスポート・レイヤ・セキュリティ

同期時のセキュリティを強化するためにトランスポート・レイヤ・セキュリティを使用すると、Ultra Light アプリケーションと統合データベースの間でやりとりするメッセージを暗号化できます。

暗号化テクノロジについては、『Mobile Link 管理ガイド』> 「Mobile Link トランスポート・レイヤ・セキュリティ」を参照してください。

Ultra Light Java クライアント・アプリケーションからのトランスポート・レイヤ・セキュリティは、独立した同期ストリームを使用します。この同期ストリームを使うときは、Ultra Light クライアントの他に Mobile Link 同期サーバを設定します。

クライアントの変更

クライアントでは、**UlSecureSocketStream** または **UlSecureRSASocketStream** 同期ストリームを選択して、一連のストリーム・パラメータを指定する必要があります。ストリーム・パラメータには、セキュリティを制御するパラメータが含まれています。

次のようにパラメータを設定します。

```
UlSynchOptions opts = new UlSynchOptions();
opts.setStream(new UlSecureSocketStream() );
opts.setStreamParms( "host=myserver;"
    + "port=2439;"
    + "certificate_company=Sybase Inc.;"
    + "certificate_unit="MEC;"
    + "certificate_name=Mobilink");
// set other options here
conn.synchronize( opts );
```

ストリーム・パラメータの詳細については、『Mobile Link クライアント』> 「UlSecureSocketStream 同期パラメータ」を参照してください。

Mobile Link サーバの設定

Java アプリケーションのセキュリティ機能のある同期ストリームは独立したストリームなので、Mobile Link 同期サーバにこのストリームを確実に受信させます。そのためには、**java_certicom_tls** または **java_rsa_tls** 同期ストリームを提供して、クライアントでの選択項目と一致させてください。

次に、コマンド・ラインの例を示します。

```
dbmlsrv9 -x
java_certicom_tls(certificate=mycertificate.crt;port=12
34)
```

java_certicom_tls と **java_rsa_tls** ストリームのセキュリティ・パラメータは、次のとおりです。

- **certificate** サーバの識別情報が保管されている証明書ファイルの名前です。証明書ファイルには、サーバの証明書、証明書の署名チェーンを構成するすべての認証局の証明書、サーバのプライベート・キーが含まれている必要があります。

  certificate パラメータのデフォルトは、**java_certicom_tls** の *sample.crt* と **java_rsa_tls** の *rsaserver.crt* です。この値は Mobile Link のデフォルト識別情報です。これらのファイルは、SQL Anywhere Studio とともに Mobile Link サーバと同じディレクトリに配布されます。

- **certificate_password** 証明書ファイルのプライベート・キーを暗号化するときに使用するパスワードです。

  デフォルトは、*sample.crt* と *rsaserver.crt* にあるプライベート・キーのパスワード **test** です。

同期の前に変更をコミットする

Ultra Light データベースは、同期のときに変更をコミットしないでおくことはできません。Ultra Light データベースを同期しようとした時点で、コミットされていないトランザクションが接続にあると、同期は失敗し、例外がスローされ、SQLE_UNCOMMITTED_TRANSACTIONS エラーが設定されます。このエラー・コードは、Mobile Link 同期サーバ・ログにも表示されます。

ダウンロード専用同期の詳細については、『Mobile Link クライアント』> 「Download Only 同期パラメータ」を参照してください。

アプリケーションへの初期データの追加

Ultra Light アプリケーションは、一般的に、使用前にデータが必要です。同期でアプリケーションにデータをダウンロードできます。アプリケーションが最初に実行されたとき、他のアクションが行われる前に必要なデータがすべてダウンロードされるように、アプリケーションに論理を追加できます。

開発のヒント
アプリケーションを段階別に開発すると、エラーが発見しやすくなります。プロトタイプの開発中に、テストとデモンストレーションを目的としたデータを得るため、一時的に INSERT 文をアプリケーションに記述します。プロトタイプが正常に動作するようになったら、同期を有効にして、一時的に使用した INSERT 文を削除します。

同期の開発に関するヒントについては、『Mobile Link 管理ガイド』>「開発のヒント」を参照してください。

同期のモニタとキャンセル

この項では、Ultra Light のアプリケーションからの同期をモニタしたりキャンセルしたりする方法について説明します。

- 同期の進行状況のモニタや、同期のキャンセルに使用する API。
- 使用するアプリケーションに追加可能なインタフェースを実装する進行状況インジケータ・コンポーネント。

同期のモニタ

同期をモニタするには、**UlSynchObserver** インタフェースを実装するクラスを作成します。このインタフェースには 1 つのメソッドがあります。

```
void updateSynchronizationStatus( UlSynchStatus status )
```

- **UlSynchOptions** クラスを使用して **UlSynchObserver** オブジェクトを登録します。
- **synchronize()** メソッドを呼び出して同期を行います。
- Ultra Light が、同期のステータスが変更するたびに observer クラスの **updateSynchronizationStatus** メソッドを呼び出します。次の項では同期のステータスについて説明します。

次の例は、同期命令の一般的なシーケンスです。この例では、**MyObserver** クラスによって **UlSynchObserver** インタフェースが実装されます。

```
UlSynchObserver observer = new MyObserver ();
UlSynchOptions opts = new UlSynchOptions();
// set options
opts.setUserName( "mluser" );
opts.setPassword( "mlpwd" );
opts.setStream( new UlSocketStream() );
opts.setStreamParms( "localhost" );
opts.setObserver( observer );
opts.setUserData( myDataObject );
// synchronize
conn.synchronize( opts );
```

同期ステータス情報の処理

UlSynchObserver を実装するクラスでは、**UlSynchStatus** オブジェクトが同期ステータス情報を保持します。**updateSynchronizationStatus** メソッドが呼び出されるたびに、Ultra Light はこのオブジェクトに同期ステータス情報を書き込みます。

UlSynchStatus オブジェクトには、次のメソッドがあります。

```
int getState()
int getTableCount()
int getTableIndex()
Object getUserData()
UlSynchOptions getSynchOptions()
UlSqlStmt getStatement()
int getErrorCode()
boolean isOKToContinue()
void cancelSynchronization()
```

- **getState** 以下のステータスのいずれかを表します。
  - **STARTING** 同期アクションはまだ開始されていません。
  - **CONNECTING** 同期ストリームは構築されていますが、まだ開かれていません。
  - **SENDING_HEADER** 同期ストリームはすでに開かれており、ヘッダが送信されようとしています。
  - **SENDING_TABLE** テーブルが送信されています。

- **SENDING_DATA** スキーマ情報またはデータが送信されています。
- **FINISHING_UPLOAD** アップロード処理が完了し、コミットが実行されています。
- **RECEIVING_UPLOAD_ACK** アップロードが完了したという確認が受信されています。
- **RECEIVING_TABLE** テーブルが受信されています。
- **SENDING_DATA** スキーマ情報またはデータが受信されています。
- **COMMITTING_DOWNLOAD** ダウンロード処理が完了し、コミットが実行されています。
- **SENDING_DOWNLOAD_ACK** ダウンロードが完了したという確認が送信されています。
- **DISCONNECTING** 同期ストリームが閉じようとしています。
- **DONE** 同期は正常に完了しました。
- **ERROR** 同期は完了しましたが、エラーが発生しました。

  同期処理の概要については、『Mobile Link 管理ガイド』>「同期処理」を参照してください。

- **getTableCount** 同期中のテーブルの数を返します。テーブルごとに送信と受信のフェーズがあります。したがって、この数は同期されるテーブルの数より多い場合があります。
- **getTableIndex** アップロード処理またはダウンロード処理される現在のテーブルは 0 から開始されます。この数値は、すべてのテーブルが同期されるのではない場合には、値を省略することがあります。
- **getSynchOptions UlSynchOptions** オブジェクトを返します。
- **sent.inserts** これまでにアップロードされた挿入済みローの数。

- **sent.updates** これまでにアップロードされた更新済みローの数。
- **sent.deletes** これまでにアップロードされた削除済みローの数。
- **sent.bytes** これまでにアップロードされたバイト数。
- **received.inserts** これまでにダウンロードされた挿入済みローの数。
- **received.updates** これまでにダウンロードされた更新済みローの数。
- **received.deletes** これまでにダウンロードされた削除済みローの数。
- **received.bytes** これまでにダウンロードされたバイト数。
- **cancelSynchronization** 同期を中断するには、このメンバを true に設定します。SQL 例外の SQLE_INTERRUPTED が設定され、通信エラーが発生したかのように同期が停止します。observer は、適切なクリーンアップを実行するように、*常に* DONE または ERROR のステータスで呼び出されます。
- **getUserData** ユーザ・データ・オブジェクトを返します。
- **getStatement** 同期を呼び出した文を返します。文は Ultra Light 内部の文であり、このメソッドを実際に使用することはほとんどありませんが、完了のために含まれています。
- **getErrorCode** 同期ステータスが ERROR に設定されているときに、診断エラー・コードを返すメソッドです。
- **isOKToContinue cancelSynchronization** が呼び出されると、**false** に設定されます。それ以外の場合は **true** です。

例

次の例は、ごく簡単な observer 関数を示しています。

```
void updateSynchronizationStatus( UlSynchStatus status
)
 {
     int state = status.getState();
     System.out.println( "Sync status: " + state );
     if( state == UlSynchStatus.SENDING_TABLE ||
        state == UlSynchStatus.RECEIVING_TABLE ){
           System.out.println( "send/receive table " +
           ( status.getTableIndex() + 1 ) +
             " of " + status.getTableCount() );
     }
 }
```

CustDB の例

observer 関数の例は CustDB サンプル・アプリケーションに含まれています。CustDB を使った実装では、同期の進捗状況を示すダイアログが表示されます。ユーザはそのダイアログで同期をキャンセルすることができます。ユーザ・インタフェース・コンポーネントは observer 関数のプラットフォームを指定します。

CustDB サンプル・コードは、SQL Anywhere ディレクトリのサブディレクトリ *Samples¥UltraLite¥CustDB* にあります。observer 関数は *CustDB* ディレクトリのプラットフォーム固有のサブディレクトリに保管されています。

進行状況ビューアの使用

Ultra Light ランタイム・ライブラリには、進行状況ビューア・クラスが 2 つ組み込まれています。これらによって、同期モニタが実装され、エンド・ユーザに対して同期キャンセル機能が提供されます。進行状況ビューアのクラスは次のとおりです。

- **ianywhere.ultralite.ui.SynchProgressViewer**　重量 AWT バージョン
- **ianywhere.ultralite.ui.JSynchProgressViewer**　Swing スレッド・モデルを考慮した、ビューアの Swing バージョン

2 つのクラスはまったく同じように使用されます。ビューアには、モーダル・ダイアログまたはモードレス・ダイアログが表示されます。ダイアログには、一連のメッセージや進行状況を示すバーが表示されます。メッセージとバーは、両方とも同期の最中に更新されます。ビューアには、[キャンセル] ボタンも表示されます。ユーザが [

キャンセル] ボタンをクリックすると、同期は停止して SQL 例外 SQLE_INTERRUPTED がスローされます。

スレッドの問題

Java アプリケーションでは、「**イベント・スレッド**」と呼ばれる単一のスレッド上ですべてのイベントが発生します。また、ユーザ・インタフェース・オブジェクトも、すべてこのイベント・スレッドで作成されます。そのときにアプリケーションが別のスレッドで動作しているかどうかは問題ではありません。1 つのアプリケーションには 1 つのイベント・スレッドしか存在しません。

イベント・スレッドではブロックしないでください。このため、イベント・スレッドでの処理は短時間で済ませるようにします。そうしないと表示のずれが発生します。モーダル・ダイアログで **show()** メソッドを呼び出しても、イベント・スレッドの実行は中断されます。このため、イベント・スレッド上では **synchronize()** メソッドを呼び出さないでください。

モーダル・ビューアの表示

次の例は、モーダル・ビューアのインスタンスがどのように呼び出されるかを示したコードの抜粋です。この **import** 文では AWT バージョンが使用されています。

```
import ianywhere.ultralite.ui.SynchProgressViewer;
// create a frame to display a dialog
java.awt.Frame frame = ...;
// get UltraLite connection
Connection conn = ...;
// set synchronization options
UlSynchOptions options = new UlSynchOptions();
options.setUserName( "my_user" );
...
// create the viewer
SynchProgressViewer viewer = new SynchProgressViewer(
frame );
viewer.synchronize( frame, options );
// execution stops here until synchronization is
complete
```

この方法で呼び出されると、ビューアは次の処理を実行します。

1. ビューア自体を同期 observer として登録します。
2. スレッドを生成して同期を実行します。
3. ビューア自体を表示し、現在のスレッドをブロックします。

4. 同期が完了すると、observer のコールバックがダイアログを破棄するので、スレッドは処理を続行します。

モードレス・ビューアの表示

次の例は、モードレス・ビューアのインスタンスがどのように呼び出されるかを示したコードの抜粋です。

```
SynchProgressViewer viewer = new SynchProgressViewer(
frame, false );
options.setObserver( viewer );
conn.synchronize( options );
```

このとき、ビューアがブロックされないよう、同期がイベント・スレッド以外のスレッド上で発生していることを確認してください。

注意

- すべてのメッセージは **SynchProgressViewerResources** リソース・バンドルから送られます。
- ビューアは、**UISynchObserver** インタフェースを実装すると同期処理にフックできます。
- CustDB サンプル・アプリケーションには進行状況ビューアが含まれています。CustDB サンプル・コードは、SQL Anywhere ディレクトリの *UltraLite¥samples¥CustDB¥java* サブディレクトリに入っています。

Ultra Light 静的型 Java の開発上の注意

この項では、Ultra Light 静的型 Java アプリケーションの開発上の注意について説明します。

Ultra Light 静的型 Java アプレットの作成

JDBC プログラムをアプレットとして作成する場合、アプリケーションはアプレットのロード元マシンとだけ同期できます。マシンのアドレスは、通常、HTML と同じです。

HTML ページにアプレットを含める

次の例は、Ultra Light アプレットの作成に使用されるサンプルの HTML ページを示します。

```
<html>
     <head>
     </head>
     <body bgcolor="FFFF00">
          <applet code="CustDbApplet.class" width=440
           height=188
           archive="custdb.zip,ulrt.jar" >
          </applet>
     </body>
 </html>
```

アプレット・タグが指定するものは、次のとおりです。

- アプレットが起動するクラス

  ```
  code="CustDbApplet.class"
  ```

- アプレットを表示する Web ブラウザのウィンドウのサイズ

  ```
  width=440 height=188
  ```

- アプレットの実行に必要な zip ファイル

  ```
  archive="custdb.zip,ulrt.jar"
  ```

  この場合、Ultra Light CustDB サンプル・アプリケーションを実行するには、*custdb.zip* ファイルと Ultra Light ランタイム zip ファイルが必要です。

Ultra Light 静的型 Java アプリケーションの構築

この項の項目は、次のとおりです。

Ultra Light 静的型 Java クラスの生成

リファレンス・データベースを準備し、アプリケーションに Ultra Light プロジェクトを定義し、SQL 文を追加してデータ・アクセス機能を定義すると、ジェネレータに必要な情報はすべてリファレンス・データベースに含まれます。

Ultra Light ジェネレータの概要については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light データ・アクセス・コードの生成」を参照してください。コマンド・ライン・オプションについては、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light ジェネレータ」を参照してください。

ジェネレータの出力は、ユーザが選択した名前を持つ Java ソース・ファイルになります。データベースの設計とアプリケーションが必要とするデータベースの機能性により、このファイルのサイズと内容はかなり異なることがあります。

Ultra Light ジェネレータの出力をカスタマイズする方法は、アプリケーションの特性によっていくつかあります。

概要

リファレンス・データベースに対して Ultra Light ジェネレータを実行して、クラスを生成します。

❖ **Ultra Light ジェネレータを実行するには、次の手順に従います。**

- コマンド・プロンプトで次のコマンドを入力します。

```
ulgen -c "connection-string" options
```

options は、プロジェクトの詳細により異なります。

一般的なコマンド・ラインの組み合わせ

Java コード生成時に、いくつかのオプションを指定できます。

- **-t java** Java コードを生成します。ジェネレータは、C/C++ 開発用のツールと同じなので、このオプションはすべての Java の使用に必要です。
- **-i** Java コンパイラには、内部クラスを正しくサポートしないものがあるので、ジェネレータはデフォルトでは、内部クラスを含む Java コードを生成しません。内部クラスをサポートするコンパイラを利用する場合は、このオプションを使用してください。
- **-p** 一般的に、生成されるクラスはパッケージに含めます。パッケージには、アプリケーションからの他のクラスが含まれることがあります。このスイッチを使用すると、生成されるファイルにクラスのパッケージ名を含めるようにジェネレータに指示できます。
- **-s** SQL 文を実行するためのコードの他に、SQL 文自体をインタフェースとして生成します。このオプションを指定しないと、文字列はデータベース・クラス自体のメンバとして書き出されます。
- **-a** SQL 文字列の名前を大文字にします。-a オプションを選択すると、生成したファイルで使われる各 SQL 文の識別子は、データベースに追加したときその文に指定した名前から抽出されます。Java の一般的な規則では、大文字を使用して定数を表します。SQL 文字列の名前は Java コードの定数なので、このオプションを使用して、一般的な規則に従った文字列識別子を生成してください。

例

- 次のコマンド (すべて 1 行に入力) では、**CustDemo** プロジェクトの SQL 文を表すコードと必要なデータベース・スキーマを生成して、ファイル *uldemo.java* に出力します。

```
ulgen -c "dsn=Ultralite 9.0 Sample;uid=DBA;pwd=SQL"
-a -t java -s IStatements CustDemo uldemo.java
```

Ultra Light 静的型 Java アプリケーションのコンパイル

❖ 生成されたファイルをコンパイルするには、次の手順に従います。

1　クラス・パスを設定します。

Ultra Light 静的型 Java アプリケーションをコンパイルするときは、Java コンパイラは次のクラスにアクセスできる必要があります。

- Java ランタイム・クラス
- Ultra Light ランタイム・クラス
- ターゲットのクラス (通常、現在のディレクトリにある)

次のクラス・パスは、これらのクラスへのアクセスを指定するものです。

```
%JAVA_HOME%¥jre¥lib¥rt.jar;%ASANY9%¥ultralite¥java¥
lib¥ulrt.jar;.
```

JAVA_HOME は Java のインストール・ディレクトリを示しています。また、ASANY9 は SQL Anywhere のインストール・ディレクトリを示しています。

JDK 1.1 開発の場合、*ulrt.jar* は *UltraLite¥java* ディレクトリの *jdk11¥lib* サブディレクトリにあります。

2　クラスをコンパイルします。

手順 1 に従ってクラス・パスを設定したら、*javac* を使用して、次のコマンドを (すべて 1 行に) 入力します。

```
javac file.java
```

コンパイラによって *file.java* のクラス・ファイルが作成されます。

コンパイル手順では、複数のクラス・ファイルが作成されます。生成された *.class* ファイルはすべて配備に含めてください。

Java アプリケーションの配備

Ultra Light アプリケーションの構成は、次のとおりです。

- アプリケーションを実装するために作成したクラス・ファイル
- 生成されたクラス・ファイル
- Java コア・クラス (*rt.jar*)
- Ultra Light ランタイム JAR ファイル (*ulrt.jar*)

Ultra Light アプリケーションは、適切であればどんな方法でも配備できます。これらのクラス・ファイルを 1 つの JAR ファイルにまとめて配備を簡単にすることもできます。

作成した Ultra Light アプリケーションは、最初に起動した時に、自動的にデータベースを初期化します。最初は、データベースにデータは格納されていません。データは、アプリケーションで INSERT 文を使用して明示的に追加できます。または、同期を介して統合データベースからインポートできます。明示的な INSERT 文は、プロトタイプを開発する場合に特に便利です。

第 4 章

Ultra Light 静的型 Java API リファレンス

この章の内容

この章では、Ultra Light 静的型 Java API について説明します。JDBC と異なる内容だけを記載しています。

Ultra Light API リファレンス

この項では、Ultra Light が提供する JDBC インタフェースの拡張機能について説明します。また、Ultra Light でサポートされていない JDBC 機能についても説明します。

Ultra Light での JDBC 機能

JDBC Ultra Light アプリケーション開発に特有の機能と制限事項は、次のとおりです。

Ultra Light 静的型 Java API は JDBC 2.0 の次の **ResultSet** メソッドが追加され、JDBC 1.2 でモデル化されています。

- absolute( )
- afterLast( )
- beforeFirst( )
- first( )
- isAfterLast( )
- isBeforeFirst( )
- isFirst( )
- isLast( )
- last( )
- previous( )
- relative( )

Ultra Light 開発モデルとの互換性がなく、Ultra Light によってサポートされていない機能は、次のとおりです。

- メタデータへのアクセス (システム・テーブルへのアクセス) は一部のみサポートされています。したがって、**DatabaseMetaData** インタフェースは使用できません。メタデータへのアクセスは、カラム数とカラムのタイプに限られています。
- Java オブジェクトは、データベースに保管できません。
- ストアド・プロシージャやストアド関数はサポートされていません。
- 静的 SQL 文だけがサポートされています。これらの文は、データベースに追加して Ultra Light ジェネレータが生成できるようにしてください。

サポートされていない JDBC メソッド

Ultra Light は、次の JDBC 1.2 メソッドをサポートしていません。次のメソッドを使用すると、ベンダのコードとともに **SQLException** が通知され、Ultra Light がサポートしない機能であることが告げられます。

Connection インタフェース

- getCatalog
- getMetaData
- getTransactionIsolation
- setCatalog
- setTransactionIsolation

ResultSet インタフェース

- getMetaData

Statement インタフェース

- cancel
- getMaxFieldSize
- getMaxRows

- setMaxFieldSize
- setMaxRows

JdbcConnection クラス

パッケージ ianywhere.ultralite.jdbc

説明 Ultra Light データベースの接続を表します。ほとんどのメソッドは、JDBC 接続クラスから継承されます。サポートされていないメソッドは `unsupported feature` 例外をスローします。

countUploadRows メソッド

プロトタイプ long **countUploadRows(** int *mask*, long *threshold* **)**

説明 次回の同期でアップロードする必要のあるロー数を返します。

同期が必要かどうかを判定するときに使用する関数です。

パラメータ **mask** チェック対象のパブリケーションのセット。値 0 はデータベース全体を示します。このセットはマスクとして提供されます。たとえば、次のマスクはパブリケーション PUB1 と PUB2 に対応します。

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

パブリケーション・マスクの詳細については、「publication 同期パラメータ」104 ページを参照してください。

threshold カウントするローの最大数を決定する値。呼び出しにかかる時間を制限します。値 0 は制限がないことを示します。値 1 は、同期の必要なローがあるかどうかを判別する場合に使用します。

検査結果 アップロードするローの数

スロー java.sql.SQLException

getDatabaseID メソッド

プロトタイプ　public int **getDatabaseID( )** throws SQLException

説明　getDatabaseID() は、グローバル・オートインクリメントに使用する現在のデータベース ID を返します。getDatabaseID() は、setDatabaseID への最終呼び出しによって設定された値を返します。また、ID が設定されていない場合は、-1 を返します。

getDefragIterator メソッド

プロトタイプ　JdbcDefragIterator **getDefragIterator( )**

説明　断片化解除反復子を初期化して返します。

パラメータ　**user_name**　Mobile Link ユーザ名。「user_name 同期パラメータ」116 ページを参照してください。

password　user_name に関連付けられたパスワード。「password 同期パラメータ」101 ページを参照してください。

script_version　スクリプト・バージョン。「version 同期パラメータ」117 ページを参照してください。

stream_defn　同期で使用されるストリーム。「stream 同期パラメータ」109 ページを参照してください。

parms　同期で使用されるユーザ指定のパラメータ

「stream_parms 同期パラメータ」112 ページを参照してください。

検査結果　断片化解除反復子

スロー　java.sql.SQLException

参照　「Ultra Light データベースの断片化解除」53 ページ

getLastDownloadTimeDate メソッド

プロトタイプ java.util.Date **getLastDownloadTimeDate(** int *mask* **)**

説明 特定の文の結果セットに対する変更が最後にダウンロードされた時刻を返します。

パラメータ **mask** 最終ダウンロード時間を取得するパブリケーションのセット。値 0 はデータベース全体を示します。このセットはマスクとして提供されます。たとえば、次のマスクはパブリケーション PUB1 と PUB2 に対応します。

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

パブリケーション・マスクの詳細については、「publication 同期パラメータ」104 ページを参照してください。

検査結果

- 文が最後にダウンロードされた時刻

getLastDownloadTimeLong メソッド

プロトタイプ long **getLastDownloadTimeLong(** int *mask* **)**

説明 特定の文の結果セットに対する変更が最後にダウンロードされた時刻を返します。

パラメータ **mask** 最終ダウンロード時間を取得するパブリケーションのセット。値 0 はデータベース全体を示します。このセットはマスクとして提供されます。たとえば、次のマスクはパブリケーション PUB1 と PUB2 に対応します。

```
UL_PUB_PUB1 | UL_PUB_PUB2
```

パブリケーション・マスクの詳細については、「publication 同期パラメータ」104 ページを参照してください。

検査結果

- 文が最後にダウンロードされた時刻

getLastIdentity メソッド

プロトタイプ　long **getLastIdentity( )**

説明　直前に使用した identity の値を返します。この関数は、次の SQL 文と同義です。

```
SELECT @@identity
```

この関数は、グローバル・オートインクリメント・カラムで使うと特に便利です。

検査結果　直前に使用した identity の値

参照　『Mobile Link クライアント』>「最後に割り当てられた値の割り出し」

『Mobile Link クライアント』>「グローバル・データベース識別子の設定」

globalAutoincUsage メソッド

プロトタイプ　short **globalAutoincUsage( )**

説明　データベースに含まれているすべてのテーブルのグローバル・オートインクリメント・カウンタの最大比率 (%) を返します。この値は、データベース ID を設定するかどうか決めるときに使用すると便利です。

検査結果　グローバル・オートインクリメントの値の使用済み比率 (%)

スロー　java.sql.SQLException

参照　『Mobile Link クライアント』>「デフォルトのグローバル・オートインクリメント・カラムの宣言」

「setDatabaseID メソッド」80 ページ

grant メソッド

プロトタイプ　void **grant(** String *user*, String *password* **)**

説明　ユーザ名とパスワードに、Ultra Light データベースに接続するパーミッションを付与します。このメソッドを有効にするには、**JdbcSupport.enableUserAuthentication** でユーザ認証が有効にされている必要があります。

grant メソッドは、明示的な JdbcDatabase オブジェクトを持たないアプリケーションに使用する JdbcConnection に提供されています。

パラメータ　**user**　接続時にユーザ名として入力する文字列

password　接続時にパスワードとして入力する文字列

検査結果　void

スロー　java.sql.SQLException

revoke メソッド

プロトタイプ　void **revoke(** String *user* **)**

説明　ユーザ名から Ultra Light データベースに接続するパーミッションを取り消します。このメソッドを有効にするには、**JdbcSupport.enableUserAuthentication** でユーザ認証が有効にされている必要があります。

grant メソッドは、明示的な JdbcDatabase オブジェクトを持たないアプリケーションに使用する JdbcConnection に提供されています。

パラメータ　**user**　データベースに接続できないようにするユーザ名

検査結果　void

スロー　java.sql.SQLException

setDatabaseID メソッド

プロトタイプ　void **setDatabaseID(** int *value* **)**

説明　データベース ID を設定します。

パラメータ　**value**　グローバル・データベース識別子として使用する整数値

スロー　java.sql.SQLException

参照　「globalAutoincUsage メソッド」79 ページ

synchronize メソッド

プロトタイプ

```
void synchronize(
        java.lang.String user_name,
        java.lang.String password,
        java.lang.String script_version,
        UlStream stream_defn,
        java.lang.String parms )

void synchronize(
            ianywhere.ultralite.runtime UlSynchOptions opts
    )
```

説明　データを Mobile Link 同期サーバと同期させます。

パラメータ

user_name　Mobile Link ユーザ名。「user_name 同期パラメータ」116 ページを参照してください。

password　user_name に関連付けられたパスワード。「password 同期パラメータ」101 ページを参照してください。

script_version　スクリプト・バージョン。「version 同期パラメータ」117 ページを参照してください。

stream_defn　同期で使用されるストリーム。「stream 同期パラメータ」109 ページを参照してください。

parms　同期で使用されるユーザ指定のパラメータ

「stream_parms 同期パラメータ」112 ページを参照してください。

opts　ULSynchOptions オブジェクト。「同期パラメータ」90 ページを参照してください。

スロー　java.sql.SQLException

startSynchronizationDelete メソッド

プロトタイプ void **startSynchronizationDelete( )**

説明 Mobile Link 同期で行われた削除のロギングを再起動します。

スロー java.sql.SQLException

参照 『ASA SQL リファレンス・マニュアル』> 「START SYNCHRONIZATION DELETE 文 [Mobile Link]」

stopSynchronizationDelete メソッド

プロトタイプ void **stopSynchronizationDelete( )**

説明 Mobile Link 同期で行われた削除のロギングを停止します。

スロー java.sql.SQLException

参照 『ASA SQL リファレンス・マニュアル』> 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」

JdbcDatabase クラス

パッケージ ianywhere.ultralite.jdbc

説明 **JdbcDatabase** はデータベースの難読化にのみ直接使用されます。生成されたデータベース・クラスは **JdbcDatabase** を継承し、Ultra Light データベースを表すオブジェクトを提供します。ほとんどの JdbcDatabase メソッドは、生成されたデータベース・クラスから使用されます。

詳細については、「生成されたデータベース・クラス」85 ページを参照してください。

changeEncryptionKey メソッド

プロトタイプ　Connection **changeEncryptionKey( )**

説明　Ultra Light データベースの暗号化キーを変更します。

検査結果　JDBC 接続オブジェクト

スロー　java.sql.SQLException

参照　「Ultra Light データベースの暗号化」49 ページ

close メソッド

プロトタイプ　void **close( )**

説明　Ultra Light データベースへのすべての接続を閉じます。このメソッドを実行してからでないと、Ultra Light データベースを削除できません。

検査結果　void

スロー　java.sql.SQLException

connect メソッド

プロトタイプ　Connection **connect( )**

Connection **connect(** String *user*, String *password* **)**

Connection **connect(** String *user*, String *password*, Properties *info* **)**

説明　Ultra Light データベースに接続します。ユーザ名とパスワードは、**JdbcSupport.enableUserAuthentication** でユーザ認証が有効にされている場合にのみチェックされます。

パラメータ　**user**　データベースに接続できるユーザ名

password　接続時にパスワードとして入力する文字列

info ユーザ名とパスワードを保持する **Properties** オブジェクト

検査結果 JDBC 接続オブジェクト

スロー java.sql.SQLException

drop メソッド

プロトタイプ void **drop( )**

説明 Ultra Light データベース・ファイルを削除します。このメソッドは注意して使用してください。このメソッドは、JdbcDatabase.close() メソッドを呼び出してからでないと実行できません。

検査結果 void

スロー java.sql.SQLException

参照 「close メソッド」83 ページ

grant メソッド

プロトタイプ void **grant(** String *user*, String *password* **)**

説明 ユーザ名とパスワードに、Ultra Light データベースに接続するパーミッションを付与します。このメソッドを有効にするには、**JdbcSupport.enableUserAuthentication** でユーザ認証が有効にされている必要があります。

パラメータ **user** 接続時にユーザ名として入力する文字列

password 接続時にパスワードとして入力する文字列

検査結果 void

スロー java.sql.SQLException

revoke メソッド

プロトタイプ　void **revoke(** String *user* **)**

説明　ユーザ名から Ultra Light データベースに接続するパーミッションを取り消します。このメソッドを有効にするには、**JdbcSupport.enableUserAuthentication** でユーザ認証が有効にされている必要があります。

パラメータ　**user**　データベースに接続できないようにするユーザ名

検査結果　void

スロー　java.sql.SQLException

setDefaultObfuscation メソッド

プロトタイプ　**setDefaultObfuscation (true | false)**

説明　データベースを読みにくくします。

参照　「Ultra Light データベースの難読化」50 ページ

生成されたデータベース・クラス

説明　生成されたデータベース・クラスは、**JdbcDatabase** を継承します。このデータベース・クラスは、Ultra Light データベースを表すオブジェクトです。JdbcDatabase メソッドは、通常は生成されたデータベース・クラスで使用されます。

コンストラクタ　new *database-name*(Properties *props*)

database-name は、生成されたデータベース・クラス名です。クラス名は、Ultra Light ジェネレータの `-f` コマンド・ライン・オプションを使用して指定できます。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light ジェネレータ」を参照してください。

パラメータ　**props**　次の項目の一部またはすべてを含む Properties オブジェクト

- persist
- persistfile
- key

詳細については、「Properties オブジェクトを使用した接続情報の格納」41 ページを参照してください。

JdbcDefragIterator クラス

パッケージ　ianywhere.ultralite.jdbc

説明　データベース・ストアの断片化を明示的に解除するためのオブジェクトを提供します。

ulStoreDefragStep メソッド

プロトタイプ　boolean **ulStoreDefragStep(** UlConnection *conn* **)**

説明　Ultra Light データベースの一部の断片化を解除します。

パラメータ　**conn**　JdbcConnection オブジェクトである現在の接続

検査結果　**true**　処理が成功した場合

false　処理が失敗した場合

スロー　java.sql.SQLException

参照　『ASA SQL リファレンス・マニュアル』> 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」

JdbcSupport クラス

パッケージ　ianywhere.ultralite.jdbc

説明　Ultra Light 機能を有効にするメソッドを提供する静的クラス

enableUserAuthentication メソッド

プロトタイプ　void **enableUserAuthentication( )**

説明　Ultra Light データベースに接続するときにユーザ認証が要求されるように、Ultra Light データベースを設定します。このメソッドを呼び出してから、データベース・オブジェクトを作成します。

パラメータ　なし。

検査結果　Void

スロー　java.sql.SQLException

参照　「ユーザ認証の例」47 ページ

disableUserAuthentication メソッド

プロトタイプ　void **disableUserAuthentication( )**

説明　Ultra Light データベースに接続するときにユーザ認証が要求されないように、Ultra Light データベースを設定します。このメソッドを呼び出してから、データベース・オブジェクトを作成します。

パラメータ　なし。

検査結果　Void

スロー　java.sql.SQLException

参照　「enableUserAuthentication メソッド」87 ページ

第 5 章

同期パラメータ・リファレンス

この章の内容　この章では、同期パラメータのリファレンス情報を提供します。

同期パラメータ

同期パラメータは、ianywhere.ultralite.runtime.UlSynchOptions オブジェクトのフィールドです。フィールドはプライベート変数です。フィールドの値を取得し設定するメソッドを供給します。UlSynchOptions オブジェクトは、ianywhere.ultralite.jdbc.JdbcConnection.synchronize を呼び出す引数として供給します。

フィールド	アクセス・メソッド	参照場所
auth_status	getAuthStatus setAuthStatus	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「auth_status パラメータ」
auth_value	getAuthValue getAuthValue	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「auth_value 同期パラメータ」
download_only	getDownloadOnly setDownloadOnly	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「download_only 同期パラメータ」
ignored_rows	getIgnoredRows setIgnoredRows	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「ignored_rows 同期パラメータ」
new_password	getNewPassword setNewPassword	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「new_password 同期パラメータ」
observer	getObserver setObserver	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「observer 同期パラメータ」
password	getPassword setPassword	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「password 同期パラメータ」
ping	getPing setPing	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「ping 同期パラメータ」

フィールド	アクセス・メソッド	参照場所
publication	getSynchPublication setSynchPublication	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「publication 同期パラメータ」
stream	getStream setStream	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「stream 同期パラメータ」
stream_parms	getStreamParms setStreamParms	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「stream_parms 同期パラメータ」
upload_ok	getUploadOK setUploadOK	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「upload_ok 同期パラメータ」
upload_only	getUploadOnly setUploadOnly	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「upload_only 同期パラメータ」
user_data	getUserData setUserData	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「user_data 同期パラメータ」
user_name	getUserName setUserName	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「user_name 同期パラメータ」
version	getScriptVersion setScriptVersion	『Ultra Light 静的型 Java ユーザーズ・ガイド』>「version 同期パラメータ」

各同期パラメータの役割については、『Mobile Link クライアント』>「同期パラメータ」を参照してください。

auth_parms パラメータ

機能

カスタム・ユーザ認証スクリプトにパラメータを供給します。

アクセス・メソッド

String[ ] **getAuthParms( )**

void **setAuthParms(** String[ ] *auth_parms* **)**

使用法

次のようにパラメータを設定します。

```
ul_char * Params[ 3 ] = { UL_TEXT( "parm1" ),
                          UL_TEXT( "parm2" ),
                          UL_TEXT( "parm3" ) };
 // ...
 info.num_auth_parms = 3;
 info.auth_parms = Params;

params = new String[ num_params ];
 // set params values
 UlSynchOptions opts = new UlSynchOptions;
 opts.setAuthParms( params );
 opts.setAuthParmsNumber( num_params );
```

参照

- 『Mobile Link クライアント』> 「Authentication Parameters 同期パラメータ」
- 「num_auth_parms パラメータ」99 ページ
- 『Mobile Link 管理ガイド』> 「authenticate_parameters 接続イベント」
- 『Mobile Link 管理ガイド』> 「authenticate_user 接続イベント」

auth_status パラメータ

機能　Mobile Link のユーザ認証のステータスをレポートします。

アクセス・メソッド　short **getAuthStatus( )**

void **setAuthStatus(** short *auth_status* **)**

使用法　このパラメータにアクセスするには、次のように入力します。

```
UlSynchOptions opts = new UlSynchOptions;
 // set options here
 conn.synchronize( opts );
 returncode = opts.getAuthStatus();
```

指定可能な値　同期後は、パラメータには次の値の 1 つが入ります。統合データベースでカスタム **authenticate_user** 同期スクリプトが異なる値を返す場合は、『Mobile Link 管理ガイド』> 「authenticate_user 接続イベント」で指定されているルールに従って値が解釈されます。

定数	値	説明
UlDefnUL_AUTH_STATUS_UNKNOWN	0	認証ステータスが不明です。接続がまだ同期していない可能性があります。
UlDefnUL_AUTH_STATUS_VALID	1000	ユーザ ID とパスワードは、同期時には有効でした。
UlDefnUL_AUTH_STATUS_VALID_BUT_EXPIRES_SOON	2000	ユーザ ID とパスワードは、同期時には有効でしたが、まもなく有効期限が切れます。
UlDefnUL_AUTH_STATUS_EXPIRED	3000	認証に失敗しました。ユーザ ID またはパスワードの有効期限が切れています。
UlDefnUL_AUTH_STATUS_INVALID	4000	認証に失敗しました。不正なユーザ ID またはパスワードです。
UlDefnUL_AUTH_STATUS_IN_USE	5000	認証に失敗しました。ユーザ ID はすでに使用されています。

参照

- 『Mobile Link クライアント』> 「Authentication Status 同期パラメータ」
- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」

auth_value 同期パラメータ

機能

カスタム・ユーザ認証同期スクリプトからの戻り値をレポートします。

アクセス・メソッド

long **getAuthValue( )**

void **setAuthValue(** long *auth_value* **)**

デフォルト

デフォルトの Mobile Link ユーザ認証メカニズムによって設定される値については、『Mobile Link 管理ガイド』>「authenticate_user 接続イベント」で説明します。

使用法

このパラメータは読み込み専用です。

このパラメータにアクセスするには、次のように入力します。

```
UlSynchOptions opts = new UlSynchOptions;
// set other options here
conn.synchronize( opts );
returncode = opts.getAuthValue();
```

参照

- 『Mobile Link クライアント』>「Authentication Value 同期パラメータ」
- 『Mobile Link 管理ガイド』>「authenticate_user 接続イベント」
- 『Mobile Link 管理ガイド』>「authenticate_user_hashed 接続イベント」
- 「auth_status パラメータ」93 ページ

download_only 同期パラメータ

機能

この同期中は、Ultra Light データベースから変更をアップロードしません。

アクセス・メソッド

boolean **getDownloadOnly( )**

void **setDownloadOnly(** boolean *download_only* **)**

デフォルト

このパラメータはオプションのブール値であり、デフォルトは false です。

使用法

次のようにパラメータを設定します。

```
UlSynchOptions opts = new UlSynchOptions;
opts.setDownloadOnly( true );
 // set other options here
conn.synchronize( opts );
```

参照

- 『Mobile Link クライアント』> 「Download Only 同期パラメータ」
- 『Mobile Link クライアント』> 「Ultra Light データベースへの読み込み専用テーブルの組み込み」
- 「upload_only 同期パラメータ」114 ページ

ignored_rows 同期パラメータ

機能　同期中にスクリプトがないため、Mobile Link 同期サーバによってローが無視された場合、レポートを行います。

このパラメータは読み込み専用です。

アクセス・メソッド　boolean **getIgnoredRows( )**

void **setIgnoredRows(** boolean *ignored_rows* **)**

new_password 同期パラメータ

機能 ユーザ名に対する新しい Mobile Link パスワードを設定します。

アクセス・メソッド java.lang.String **getNewPassword( )**

void **setNewPassword(** java.lang.String *new_password* **)**

デフォルト デフォルト値はありません。

使用法 次のようにパラメータを設定します。

```
UlSynchOptions opts = new UlSynchOptions;
 opts.setUserName( "50" );
 opts.setPassword( "mypassword" );
 opts.setNewPassword( "mynewpassword" );
 // set other options here
 conn.synchronize( opts );
```

参照

- 『Mobile Link クライアント』> 「New Password 同期パラメータ」
- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」.

num_auth_parms パラメータ

機能　カスタム認証スクリプトに渡される認証パラメータ文字列の数。

アクセス・メソッド　byte **getAuthParmsNumber( )**

void **setAuthParmsNumber(** byte *value* **)**

デフォルト　カスタム認証スクリプトに渡されるパラメータはありません。

使用法　auth_parms とともにパラメータを使用して、カスタム認証スクリプトに情報を提供します。

詳細については、「auth_parms パラメータ」92 ページを参照してください。

参照

- 『Mobile Link クライアント』>「Number of Authentication Parameters パラメータ」
- 「auth_parms パラメータ」92 ページ
- 『Mobile Link 管理ガイド』>「authenticate_parameters 接続イベント」
- 『Mobile Link 管理ガイド』>「authenticate_user 接続イベント」

observer 同期パラメータ

機能 同期をモニタするコールバック関数へのポインタです。

アクセス・メソッド ianywhere.ultralite.runtime.UlSynchObserver **getObserver( )**

void **setObserver(** ianywhere.ultralite.runtime.UlSynchObserver *observer* **)**

参照

- 『Mobile Link クライアント』> 「Observer 同期パラメータ」
- 「同期のモニタとキャンセル」61 ページ
- 「user_data 同期パラメータ」115 ページ

password 同期パラメータ

機能

user_name に対する Mobile Link パスワードを指定する文字列です。このユーザ名とパスワードは他のデータベース・ユーザ ID やパスワードとは別のもので、アプリケーションを Mobile Link 同期サーバに対して識別し、認証するために使用されます。

アクセス・メソッド

java.lang.String **getPassword( )**

void **setPassword(** java.lang.String *password* **)**

デフォルト

デフォルト値はありません。

使用法

次のようにパラメータを設定します。

```
UlSynchOptions opts = new UlSynchOptions;
opts.setUserName( "50" );
opts.setPassword( "mypassword" );
// set other options here
conn.synchronize( opts );
```

参照

- 『Mobile Link クライアント』> 「Password 同期パラメータ」
- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」

ping 同期パラメータ

機能

Ultra Light クライアントと Mobile Link 同期サーバ間の通信を確認します。このパラメータが true に設定されている場合は、同期は行われません。

Mobile Link 同期サーバは、ping 要求を受信すると、統合データベースに接続し、ユーザを認証し、ユーザ認証ステータスと値をクライアントに送信します。

ping に成功した場合、Mobile Link サーバは情報メッセージを発行します。ping に失敗した場合は、エラー・メッセージを発行します。

Mobile Link サーバがコマンド・ライン・オプション -zu+ を指定して実行されていると、Mobile Link ユーザ名が ml_user システム・テーブルに見つからない場合は Mobile Link サーバがユーザを ml_user に追加します。

Mobile Link 同期サーバに次のスクリプトが存在する場合、ping 要求に対してこれらのスクリプトを実行できます。

- begin_connection
- authenticate_user
- authenticate_user_hashed
- end_connection

アクセス・メソッド

boolean **getPing( )**

void **setPing(** boolean *ping* **)**

デフォルト

このパラメータはオプションのブール値であり、デフォルトは false です。

使用法

次のようにパラメータを設定します。

```
UlSynchOptions opts = new UlSynchOptions;
opts.setUserName( "50" );
opts.setPing( true );
// set other options here
conn.synchronize( opts );
```

参照

- 『Mobile Link クライアント』>「Ping 同期パラメータ」
- 『Mobile Link クライアント』>「-pi オプション」

publication 同期パラメータ

機能

同期させるパブリケーションを指定します。

アクセス・メソッド

int **getSynchPublication( )**

void **setSynchPublication(** int *publication* **)**

デフォルト

パブリケーションを指定しない場合、すべてのデータが同期されます。

使用法

ulgen -v コマンド・ライン・オプションで指定したパブリケーションが Ultra Light ジェネレータによって UL_PUB_pubname という名前の大文字の定数として識別されます。pubname は、-v オプションに指定する名前です。

たとえば、次のコマンド・ラインは、定数 **sales**:UL_PUB_SALES で識別されるパブリケーションを生成します。

```
ulgen -v sales ...
```

同期時には、publication パラメータを「**パブリケーション・マスク**」、つまり、パブリケーション定数を OR で結合したリストに設定します。次に例を示します。

```
UlSynchOptions opts = new UlSynchOptions;
opts.setSynchPublication(
  projectname.UL_PUB_MYPUB1 |
  projectname.UL_PUB_MYPUB2 );
// set other options here
conn.synchronize( opts );
```

projectname は、Ultra Light ジェネレータによって生成されるメイン・プロジェクト・クラスの名前です。

特殊なパブリケーション・マスク **UL_SYNC_ALL** は、パブリケーションに含まれるかどうかに関係なく、データベース内のすべてのテーブルを表します。マスク **UL_SYNC_ALL_PUBS** は、データベース内のパブリケーションに含まれるすべてのテーブルを表します。

参照

- 『Mobile Link クライアント』> 「publication 同期パラメータ」

- 『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light ジェネレータ」
- 『Mobile Link クライアント』> 「別々に同期するためのデータ・セットの設計」

security 同期パラメータ

機能

Mobile Link 同期サーバとのメッセージ交換に Certicom 暗号化テクノロジを使用するように、Ultra Light クライアントを設定します。

別途ライセンスを取得できるオプションが必要
Certicom テクノロジを使用するには、別途ライセンスを取得できる SQL Anywhere Studio 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。このオプションの詳細については、『SQL Anywhere Studio の紹介』>「SQL Anywhere Studio へようこそ」を参照してください。

アクセス・メソッド

このパラメータは Java では使用できません。

Ultra Light Java アプリケーションからセキュリティ機能のある同期を使用するには、別個のストリームを選択します。詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』>「同期パラメータの初期化」を参照してください。

参照

- 『Mobile Link クライアント』>「Security 同期パラメータ」
- 『Mobile Link 管理ガイド』>「Mobile Link トランスポート・レイヤ・セキュリティ」

security_parms 同期パラメータ

機能

トランスポート・レイヤ・セキュリティを使用する場合に必要なパラメータを設定します。このパラメータは、**security** パラメータとともに使用してください。

詳細については、「security 同期パラメータ」106 ページを参照してください。

アクセス・メソッド

このパラメータは Java では使用できません。

Ultra Light Java アプリケーションからセキュリティ機能のある同期を使用するには、別個のストリームを選択します。詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』>「同期パラメータの初期化」を参照してください。

使用法

ULSecureCerticomTLSStream() と ULSecureRSATLSStream() セキュリティ・パラメータは、次のオプションのパラメータで構成される文字列を使用します。オプションのパラメータは、セミコロンで区切られた文字列で指定します。

- **certificate_company**　証明書に記されている組織フィールドがこの値と一致する場合にだけ、Ultra Light アプリケーションはサーバ証明書を受け入れます。デフォルトでは、このフィールドはチェックされていません。
- **certificate_unit**　証明書に記されている部署フィールドがこの値と一致する場合にだけ、Ultra Light アプリケーションはサーバ証明書を受け入れます。デフォルトでは、このフィールドはチェックされていません。
- **certificate_name**　証明書に記されている共通名フィールドがこの値と一致する場合にだけ、Ultra Light アプリケーションはサーバ証明書を受け入れます。デフォルトでは、このフィールドはチェックされていません。

次に例を示します。

```
ul_synch_info info;
...
info.stream = ULSocketStream();
info.security = ULSecureCerticomTLSStream();
info.security_parms =
     UL_TEXT( "certificate_company=Sybase" )
     UL_TEXT( ";" )
     UL_TEXT( "certificate_unit=Sales" );
```

security_parms パラメータは文字列です。デフォルトは null です。

セキュリティ機能のある同期を実行する場合は、Ultra Light ジェネレータの -r コマンド・ライン・オプションも使用してください。詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light ジェネレータ」を参照してください。

参照

- 『Mobile Link クライアント』> 「Security Parameters 同期パラメータ」

stream 同期パラメータ

機能

同期に使用するように Mobile Link 同期ストリームを設定します。

詳細については、「stream_parms 同期パラメータ」112 ページを参照してください。

アクセス・メソッド

ianywhere.ultralite.runtime.UlStream **getStream( )**

void **setStream(** ianywhere.ultralite.runtime.UlStream *stream* **)**

デフォルト

パラメータにはデフォルト値がないので、明示的に設定してください。

使用法

```
UlSynchOptions opts = new UlSynchOptions;
opts.setStream(new UlSocketStream() );
opts.setStreamParms( "host=myserver;port=2439" );
// set other options here
conn.synchronize( opts );
```

ストリームのタイプでパラメータが必要な場合は、**stream_parms** パラメータを使用してパラメータを渡します。パラメータが不要なときは、**stream_parms** パラメータを null に設定します。

stream 関数には以下の種類がありますが、すべてのターゲット・プラットフォームですべての関数を使用できるわけではありません。

ストリーム	説明
ActiveSync (静的型 Java には使用できません)	ActiveSync 同期 (Windows CE のみ) ストリーム・パラメータのリストについては、『Mobile Link クライアント』> 「ActiveSync プロトコル・オプション」を参照してください。

ストリーム	説明
UlHTTPStream()	HTTP を介して同期 HTTP ストリームは基本となるトランスポートとして TCP/IP を使用します。Ultra Light アプリケーションは Web ブラウザとして機能し、Mobile Link 同期サーバは Web サーバとして機能します。Ultra Light アプリケーションは、サーバへのデータ送信のために POST 要求を送り、サーバからのデータの読み込みのために GET 要求を送ります。 ストリーム・パラメータのリストについては、『Mobile Link クライアント』>「HTTP プロトコル・オプション」を参照してください。
UlHTTPSStream()	HTTPS 同期ストリームを介して同期 HTTPS ストリームは、基本のプロトコルとして SSL または TLS を使用します。これは、インターネット・プロトコル (HTTP と TCP/IP) 上で動作します。 HTTPS ストリームでは、Certicom が提供するテクノロジを使用する必要があります。Certicom テクノロジを使用するには、別途ライセンスを取得できる SQL Anywhere Studio 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。このオプションの詳細については、『SQL Anywhere Studio の紹介』>「SQL Anywhere Studio へようこそ」を参照してください。 ストリーム・パラメータのリストについては、『Mobile Link クライアント』>「HTTPS プロトコル・オプション」を参照してください。
UlSocketStream()	TCP/IP を介して同期 ストリーム・パラメータのリストについては、『Mobile Link クライアント』>「TCP/IP プロトコル・オプション」を参照してください。

ストリーム	説明
UlSecureSocketStream()	楕円曲線暗号化を使用したトランスポート・レイヤ・セキュリティでの TCP/IP または HTTP 同期 ストリーム・パラメータのリストについては、『Mobile Link クライアント』> 「UlSecureSocketStream 同期パラメータ」を参照してください。
UlSecureRSASocketStream()	RSA 暗号化を使用したトランスポート・レイヤ・セキュリティでの TCP/IP または HTTP 同期 ストリーム・パラメータのリストについては、『Mobile Link クライアント』> 「UlSecureRSASocketStream 同期パラメータ」を参照してください。

Java 同期ストリームの詳細については、『Ultra Light 静的型 Java ユーザーズ・ガイド』> 「同期パラメータの初期化」を参照してください。

参照

- 『Mobile Link クライアント』> 「Stream Type 同期パラメータ」

stream_parms 同期パラメータ

機能

同期ストリームを設定するネットワーク・プロトコルのオプションを設定します。

オプションの割り当てのリストで、セミコロンで区切られます。各割り当ては *keyword*=*value* の形式で、許可されるキーワード・セットは同期ストリームよって異なります。

各ストリームに使用できるオプションのリストについては、次の各項を参照してください。

- 『Mobile Link クライアント』> 「HTTP プロトコル・オプション」
- 『Mobile Link クライアント』> 「HTTPS プロトコル・オプション」
- 『Mobile Link クライアント』> 「TCP/IP プロトコル・オプション」
- 『Mobile Link クライアント』> 「UlSecureRSASocketStream 同期パラメータ」

アクセス・メソッド

java.lang.String **getStreamParms( )**

void **setStreamParms(** java.lang.String *stream_parms* **)**

デフォルト

これはオプションの文字列パラメータです。デフォルトは null です。

使用法

次のようにパラメータを設定します。

```
UlSynchOptions synch_options = new UlSynchOptions();
synch_opts.setStream( new UlSocketStream() );
synch_opts.setStreamParms( "host=myserver;port=2439" );
```

参照

- 『Mobile Link クライアント』> 「Stream Parameters 同期パラメータ」
- 『Mobile Link クライアント』> 「Ultra Light 同期クライアントのネットワーク・プロトコルのオプション」

upload_ok 同期パラメータ

機能

Mobile Link アップロードのステータスをレポートします。Mobile Link 同期サーバが、この情報をクライアントに提供します。

このパラメータは読み込み専用です。

アクセス・メソッド

boolean **getUploadOK( )**

void **setUploadOK(** boolean *upload_ok* **)**

使用法

同期後、**upload_ok** パラメータには、アップロードが成功した場合は **true** が入ります。それ以外の場合は **false** が入ります。

このパラメータにアクセスするには、次のように入力します。

```
UlSynchOptions opts = new UlSynchOptions;
// set options here
conn.synchronize( opts );
returncode = opts.getUploadOK();
```

参照

- 『Mobile Link クライアント』> 「Upload OK 同期パラメータ」

upload_only 同期パラメータ

機能

現在の同期中にダウンロードは発生しないことを示します。これにより、特に低速の通信リンクでは、通信時間を節約できます。true に設定すると、クライアントは Mobile Link 同期サーバからのアップロード確認を待ってから、同期セッションを正常終了します。

アクセス・メソッド

boolean **getUploadOnly( )**

void **setUploadOnly(** boolean *upload_only* **)**

デフォルト

このパラメータはオプションのブール値であり、デフォルトは false です。

使用法

次のように true に設定します。

```
UlSynchOptions opts = new UlSynchOptions;
opts.setUploadOnly( true );
```

参照

- 『Mobile Link クライアント』> 「Upload Only 同期パラメータ」
- 『Mobile Link クライアント』> 「優先度の高い変更の同期」
- 「download_only 同期パラメータ」96 ページ

user_data 同期パラメータ

機能　アプリケーション固有の情報を同期 observer で使用できるようにします。

アクセス・メソッド　java.lang.Object **getUserData( )**

void **setUserData(** java.lang.Object *user_data* **)**

使用法　アプリケーション固有の情報を同期 observer クラスにするには、同期 observer インタフェース **UISynchObserver** を実装する際に、オブジェクトを **setUserData** メソッドに提供します。

参照

- 『Mobile Link クライアント』> 「User Data 同期パラメータ」
- 「observer 同期パラメータ」100 ページ

user_name 同期パラメータ

機能

ユーザ名を指定する文字列であり、Mobile Link 同期サーバが Mobile Link クライアントをユニークに識別するために使用します。Mobile Link では、この値を使用して、ダウンロードする内容の決定、同期ステータスの記録、同期中の割り込みからの復帰を行います。

アクセス・メソッド

java.lang.String **getUserName( )**

void **setUserName(** java.lang.String *user_name* **)**

デフォルト

これは必須の文字列パラメータです。

使用法

次のようにパラメータを設定します。

```
UlSynchOptions synch_options = new UlSynchOptions();
synch_opts.setUserName( "mluser" );
```

参照

- 『Mobile Link クライアント』> 「User Name 同期パラメータ」
- 『Mobile Link クライアント』> 「Mobile Link ユーザの認証」
- 『Mobile Link クライアント』> 「Mobile Link ユーザ」

version 同期パラメータ

機能

統合データベースの同期スクリプトは、それぞれバージョン文字列でマーク付けされます。たとえば、異なる文字列バージョンによって特定される 2 種類の **download_cursor** スクリプトがあります。Ultra Light アプリケーションは、バージョン文字列により、同期スクリプトのセットから選択できます。

アクセス・メソッド

java.lang.String **getScriptVersion( )**

void **setScriptVersion(** java.lang.String *version* **)**

デフォルト

これは文字列パラメータです。デフォルトでは、Mobile Link のデフォルト・バージョン文字列です。

使用法

次のようにパラメータを設定します。

```
UlSynchOptions synch_options = new UlSynchOptions();
synch_opts.setVersion( "default" );
```

参照

- 『Mobile Link クライアント』>「Version 同期パラメータ」
- 『Mobile Link 管理ガイド』>「スクリプト・バージョン」

索引

I

J

K

L

N

O

P

R

S

U

あ

い

う

え

か

き

く

こ

さ

し

す

せ

た

ち

て

と

な

に

ね

は

ひ