BEA WebLogic Server

BEA WebLogic Express™

WebLogic jDriver for Informix
のインストールと使い方

BEA WebLogic Server 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社（以下、「BEA」といいます）の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii) 、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする（ただし、これらには限定されない）いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA、Jolt、Tuxedo、および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder、BEA Campaign Manager for WebLogic、BEA eLink、BEA Manager、BEA WebLogic Collaborate、BEA WebLogic Commerce Server、BEA WebLogic E-Business Platform、BEA WebLogic Enterprise、BEA WebLogic Integration、BEA WebLogic Personalization Server、BEA WebLogic Process Integrator、BEA WebLogic Server、E-Business Control Center、How Business Becomes E-Business、Liquid Data、Operating System for the Internet、および Portal FrameWork は、BEA Systems, Inc. の商標です。

その他の商標はすべて、関係各社がその権利を有します。

WebLogic jDriver for Informix のインストールと使い方

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 24 日	BEA WebLogic Server バージョン 6.1

目次

このマニュアルの内容

1. WebLogic jDriver for Informix のインストール

2. WebLogic jDriver for Informix の使い方

このマニュアルの内容

このマニュアルでは、BEA の Informix データベース管理システム用 Type 4 Java Database Connectivity（JDBC）ドライバである WebLogic jDriver for Informix のインストール方法と、このドライバを使用したアプリケーションの開発方法について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「WebLogic jDriver for Informix のインストール」
- 第 2 章「WebLogic jDriver for Informix の使い方」

対象読者

このマニュアルは、Sun Microsystems の Java 2 Platform, Enterprise Edition（J2EE）を使用して e- コマース アプリケーションを構築するアプリケーション開発者を対象としています。SQL、データベースの一般的な概念、および Java プログラミングに読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA ホームページの [製品のドキュメント] をクリックするか、WebLogic Server 製品ドキュメント ページ（http://edocs.beasys.co.jp/e-docs/wls61）を直接表示してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は Adobe の Web サイト（http://www.adobe.co.jp）で無料で入手できます。

関連情報

BEA の Web サイトでは、WebLogic Server の全マニュアルを提供しています。

サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@bea.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェアの名前とバージョン、およびドキュメントのタイトルと日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.beasys.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

表記法	適用
斜体の等幅テキスト	コード内の変数を示す。 例 : `String CustomerName;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文の中で複数の選択肢を示す。
[ ]	構文の中で任意指定の項目を示す。 例 : `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例 : `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
...	コマンドラインで以下のいずれかを示す。 ♦ 引数を複数回繰り返すことができる ♦ 任意指定の引数が省略されている ♦ パラメータや値などの情報を追加入力できる
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 WebLogic jDriver for Informix のインストール

この章では、BEA の Informix 用 Type 4 JDBC ドライバである WebLogic jDriver for Informix のインストール方法について説明します。

内容は以下のとおりです。

概要

WebLogic jDriver for Informix は、Java クライアントからリレーショナル データベースにアクセスするための業界標準である Java Database Connectivity（JDBC）API の Java 実装です。この実装を使用すると、Java クライアントから Informix データベース管理システム（DBMS）に直接アクセスできます。

始める前に

この節では、WebLogic Server バージョン 6.1 用のソフトウェアのアップグレードが必要な場合とその方法を説明します。

評価ライセンス

WebLogic jDriver のライセンス機能は、このバージョンの WebLogic jDriver for Informix をインストールしたディレクトリのライセンス ファイルに含まれています。次に例を示します。

c:\bea\license.bea

インストール手順

以下の手順は、WebLogic jDriver for Informix をインストールする方法です。

WebLogic Server のインストール

WebLogic jDriver for Informix は、WebLogic Server に付属して配布され、WebLogic Server のインストール時にインストールされます。**weblogic.jar** ファイルには、Informix クラスが入っています。インストールに特別な手順は必要ありません。

スタンドアロン WebLogic jDriver for Informix のインストール

スタンドアロンの WebLogic jDriver for Informix は、まもなく提供されます。

接続プールの使い方

WebLogic Server または WebLogic Express で WebLogic jDriver for Informix を使用している場合、WebLogic Server の起動時に Informix DBMS との接続を確立する接続プールを設定できます。接続はユーザ間で共有されるので、接続プールを使用すると、ユーザごとに新規のデータベース接続を開くオーバーヘッドをなくすことができます。

アプリケーションは、WebLogic Pool、JTS、または RMI ドライバなどの多層（Type 3）JDBC ドライバを使用して、WebLogic Server に接続します。WebLogic Server は、WebLogic jDriver for Informix とプールの中の 1 つの接続を使用して、アプリケーションの代わりに Informix データベースに接続します。

WebLogic Server での接続プールのコンフィグレーション

1. WebLogic jDriver for Informix クラスを、WebLogic Server の起動に使われる WebLogic クラスパスに入れます。詳細については、『管理者ガイド』の「WebLogic Server の起動と停止」を参照してください。
2. Administration Console を使用して、接続プールを設定します。接続プールの詳細については、『管理者ガイド』の「接 続プール」、またはオンライン ヘルプの「JDBC 接続プールのコンフィグレーション」の手順を参照してください。
3. WebLogic Server を起動します。

アプリケーションでの接続プールの使い方

接続プールを使用するには、まずデータベース接続を確立する必要があります。接続を確立する方法は、接続プールを使用するアプリケーションがクライアント サイド アプリケーションかサーバサイド アプリケーションかによって決まります。

クライアントサイド アプリケーション

クライアントサイド アプリケーションで接続プールを使用するには、WebLogic RMI ドライバを使用してデータベース接続を確立します。RMI ドライバの詳細については、『WebLogic JDBC プログラミング ガイド』の「WebLogic 多層 JDBC ドライバの使い方」を参照してください。

サーバサイド アプリケーション

サーバサイド アプリケーション（サーブレットなど）で接続プールを使用するには、WebLogic `pool` または `jts` ドライバを使用してデータベース接続を確立します。詳細については、『WebLogic HTTP サーブレット プログラマーズ ガイド』の「プログラミング タスク」を参照してください。

Informix データベースへの接続の確認

Informix データベースへの接続をチェックします。以下の情報が必要です。

- ユーザに関しては、有効なユーザ名とパスワード
- データベースに関しては、データベース名、ホスト名、およびポート番号

データベースに関して必要な情報がわからない場合は、次の節の「データベース、ホスト名、およびポートの調べ方」を参照してください。

必要な情報を集めたら、接続をテストできます。コマンド ラインで次のように入力します。

```
java utils.dbping INFORMIX4 user password db@host:port
```

このコマンド ラインの引数の定義は次のとおり。

- *user* は、このデータベースの有効なユーザの Informix ユーザ名
- *password* は、そのユーザのパスワード
- *db@host:port* ― 3 つの引数の組み合わせによって、Informix データベースにアクセスする方法がわかります。
  - *db* はデータベースの名前

- *host* は、Informix サーバが動作しているコンピュータの名前
- *port* は、Informix サーバが接続リクエストをリスンしている TCP/IP ポートの番号

コマンドの構文は、データベース、@（*at* 記号）、ホスト名、コロン、ポートの順です。

DBMS との接続を確認する手順については、『WebLogic JDBC プログラミング ガイド』の「JDBC 接続のテストとトラブルシューティング」の「接続テスト」を参照してください。

データベース、ホスト名、およびポートの調べ方

インストールされている Informix サーバに接続する前に、次の情報を集めておく必要があります。

- アクセスする予定のデータベースの名前
- Informix サーバが動作しているホスト コンピュータの名前
- Informix サーバが接続リクエストをリスンしているポートの TCP/IP アドレス

混乱を避けるために、サーバという言葉を、データベースが動作しているマシンを表す場合と、データベース インスタンス自体を表す場合の両方で使用しないでください。このドキュメントでは、以下の用語を使うことでこの問題を解決しています。

- ホスト名はマシンの名前を表します。
- データベース名は Informix インスタンスの名前を表します。

接続先の Informix サーバの情報を取得するには、`$INFORMIXDIR/etc/sqlhosts` の中で、SERVER カラムの下の該当するエントリを見つけます。このファイルのエントリによって、ホスト名と接続のサービス名（右端近くのカラム）がわかります。サービス名がわかれば、ポート番号がわかります。

/etc/services ファイル（Windows NT プラットフォームでは \Winnt\system32\drivers\etc\services）の中で、サービス名に関連付けられているポート番号を見つけます。

参考資料

この節では、参考となるドキュメントおよびサンプル コードを示します。

ドキュメント

- API リファレンス
- 『WebLogic JDBC プログラミング ガイド』

サンプル コード

WebLogic Server では、サンプル コードを用意しています。サンプル コードの場所は、WebLogic Server 配布キットの samples\examples\jdbc\informix4 ディレクトリです。

2 WebLogic jDriver for Informix の使い方

この節では、WebLogic jDriver for Informix を設定および使用する方法について説明します。内容は以下のとおりです。

WebLogic jDriver for Informix とは

WebLogic jdriver for Informix は、Type 4 の pure-Java 2 層ドライバです。通信フォーマット レベルで独自のベンダ プロトコルを使用してデータベースに接続するので、クライアントサイド ライブラリは必要ありません。したがって、Type 2 の 2 層ドライバと違い、ネイティブ コールを作成せず、Java だけで記述されています。

ただし、Type 4 ドライバと Type 2 ドライバには共通点が 1 つあります。両方とも 2 層ドライバなので、どちらのドライバを使うクライアントも、データベースと接続するためには、ドライバをメモリにコピーする必要があるという点です。

WebLogic jDriver for Informix は、ResultSet の同時実行をサポートしています。つまり、1 つの ResultSet の接続を閉じてからでなくとも、別の ResultSet を開いて処理を行うことができます。ただし、ドライバは、ResultSet の同時実行とクライアントサイド キャッシングを同時にサポートすることはできません。

WebLogic jDriver for Informix は、Informix OnLine バージョン 7.x および 9.x を、7.x データ型に加えて、9.x INT8 および SERIAL8 データ型に関してサポートしています。

型のマッピング

次の表は、以下のマッピング方法を示します。

- Informix の型を WebLogic jDriver for Informix の型にマップする
- WebLogic jDriver for Informix の型を Java の型にマップする

Informix	WebLogic jDriver for Informix	Java の型
Byte	Binary	java.io.InputStream を使用
Char	Char	java.lang.String
Date	Date	java.sql.Date
Datetime	Timestamp	java.sql.Timestamp
Decimal	Decimal	java.math.BigDecimal
Float	Decimal	java.math.BigDecimal
Integer	Integer	java.lang.Integer
Integer8	Long	java.lang.BigInt
Interval	InformixInterval	Informix のリテラル文字列
Money	Decimal	java.math.BigDecimal
NChar	Char	java.lang.String
NVarchar	Varchar	java.lang.String
Serial	Integer	java.lang.Integer

Informix	WebLogic jDriver for Informix	Java の型
Serial8	Long	java.lang.BigInt
Smallfloat	Decimal	java.math.BigDecimal
Smallint	Smallint	java.lang.Integer
Text	Longvarchar	java.io.InputStream を使用
Varchar	Varchar	java.lang.String

Informix DBMS への接続

この節では、Informix DBMS への接続手順のコーディングについて説明し、接続が確立される方法を示すサンプル コードを示します。

接続手順

以下の 3 段階の手順に従って、WebLogic jDriver for Informix を使用して Informix に接続するよう、アプリケーションを設定します。

1. 次の手順に従って、JDBC ドライバをロードして登録します。

   a. WebLogic jDriver for Informix JDBC ドライバクラスの完全クラス名を使って `Class.forName().newInstance()` を呼び出します。

   b. その結果を `java.sql.Driver` オブジェクトにキャストします。

   次に例を示します。

```
Driver myDriver = (Driver)
 Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
```

2. 接続を記述する java.util.Properties オブジェクトを作成します。このオブジェクトは、ユーザ名、パスワード、データベース名、サーバ名、およびポート番号などの情報が入った名前と値の組み合わせを格納します。次に例を示します。

```
Properties props = new Properties();
props.put("user",          "scott");
props.put("password",      "secret");
props.put("db",            "myDB");
props.put("server",        "myHost");
props.put("port",          "8659");
```

3. Driver.connect() メソッドを呼び出すことで、JDBC の操作で不可欠となる JDBC 接続オブジェクトを作成します。このメソッドは、パラメータとしてドライバの URL と手順 2 で作成した java.util.Properties オブジェクトを取ります。次に例を示します。

```
Connection conn =
  myDriver.connect("jdbc:weblogic:informix4", props);
```

手順 1 と 3 では、JDBC ドライバを記述します。手順 1 では、ドライバの完全パッケージ名を使用します。ドットを使って区切ります。手順 3 では、URL（コロンで区切ります）を使ってドライバを識別します。URL には、weblogic:jdbc:informix4 という文字列を入れなければなりません。このほかに、サーバのホスト名やデータベース名などの情報を入れてもかまいません。

接続のサンプル

次のサンプル コードは、Properties オブジェクトを使って myHost というサーバ上の myDB というデータベースに接続する方法を示します。

```
Properties props = new Properties();
props.put("user",          "scott");
props.put("password",      "secret");
props.put("db",            "myDB");
props.put("server",        "myHost");
props.put("port",          "8659");

Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
Connection conn =
  myDriver.connect("jdbc:weblogic:informix4", props);
```

次のサンプルのように、db、server、および port プロパティを server プロパティにまとめることができます。

```
Properties props = new Properties();
props.put("user",         "scott");
props.put("password",     "secret");
props.put("server",       "myDB@myHost:8659");

Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
Connection conn  =
  myDriver.connect("jdbc:weblogic:informix4", props);
```

URL 内または Properties オブジェクト内に情報を提供する方法はさまざまです。ドライバの URL 内に渡される情報は、Properties オブジェクトに含まれている必要はありません。

Connection または Properties オブジェクトに設定可能な Informix 固有のその他のプロパティ

この節では、接続 URL または Properties オブジェクトに設定可能なその他の Informix 固有のプロパティについて説明します。これらのプロパティを使用すると、Informix 固有の環境をより自在に制御できます。詳細については、Informix のドキュメントを参照してください。

`weblogic.informix4.login_timeout_secs=`*`seconds_to_wait`*

Informix にログインしようとする試みがタイムアウトした場合、WebLogic jDriver for Informix は SQLException を返します。デフォルトでは、タイムアウトまでの時間は 90 秒です。タイムアウト期間を変更するには、このプロパティを SQLException が返されるまでの秒数に設定します。

`weblogic.informix4.delimited_identifiers=y`

Informix 環境変数 `DELIMIDENT` を使用すると、ANSI SQL 区切り文字付き識別子を有効または無効にできます。デフォルトではオフ（n）です。

`weblogic.informix4.db_money=`*`currency`*

Informix 環境変数 `DBMONEY` を使用すると、通貨記号の表示を設定できます。現在、デフォルト値は `$.` ですが、このプロパティを使ってオーバーライドできます。

`weblogic.informix4.db_date=`*`dateformat`*

Informix 環境変数 DBDATE を使用すると、ユーザが日付の入力フォーマットを指定できるようになります。ユーザは、ログイン時に Informix DBDATE 環境変数を設定します。デフォルト値は Y4MD です。このドライバは、2 桁の年（Y2 を含むフォーマット）をサポートしていません。この変数を使って、ResultSet.getString() 文で取得した日付を正しくフォーマットすることはできません。代わりに、ResultSet.getDate() を使用して java.util.Date オブジェクトを取得してから、日付をフォーマットします。

次のサンプル コードは、これらのプロパティを URL で使用する方法を示します。

```
jdbc:weblogic:informix4:mydb@host:1493
?weblogic.informix4.delimited_identifiers=y
&weblogic.informix4.db_money=DM
&weblogic.informix4.db_date=Y4MD
```

注意： URL は必ず 1 行で入力します。前述のサンプルでは、読みやすくするために複数の行に分けてあります。

URL 用の特殊文字の ? と & が使用されています。

次のサンプル コードは、これらのプロパティを Properties オブジェクトで使用する方法を示します。

```
Properties props = new Properties();
props.put("user",                                    "scott");
props.put("password",                                "tiger");
props.put("weblogic.informix4.delimited_identifiers", "y");
props.put("weblogic.informix4.db_money",             "DM");

Connection conn = myDriver.connect
   (jdbc:weblogic:informix4:myDB@myHost:8659",props);
```

JDBC によるデータの操作

この節では、プログラムで以下の処理を実装するための基本手順について説明します。

- 簡単な SQL クエリの作り方
- レコードの挿入、更新、および削除

- ストアド プロシージャとストアド関数の作り方と使い方
- 接続の切断とオブジェクトのクローズ
- 挿入後の SERIAL カラムの取り出し
- Informix INTERVAL データ型の使い方
- ResultSetMetaData メソッドの使い方
- 自動コミット モードの使い方

これらは JDBC の基本的な手順であり、JDBC を使ってデータを操作するための基本を説明するためのものです。詳細については、Informix のドキュメントと JDBC に関する Java 指向のドキュメントを参照してください。また、JavaSoft の「JDBC tutorial」も参照してください。

簡単な SQL クエリの作り方

データベース アクセスにおける最も基本的な作業は、データを検索することです。WebLogic jDriver for Informix では、次の 3 段階の手順に従ってデータを取り出せます。

1. SQL クエリを DBMS に送る文を作成します。
2. 作成した Statement を実行します。
3. 実行結果を ResultSet に保存します。このサンプルでは、従業員テーブル（エイリアス名 emp）に対して簡単なクエリを実行し、3 つのカラムのデータを表示します。また、データの検索先のテーブルに関するメタデータにアクセスして表示します。最後に文を閉じます。

```
Statement stmt = conn.createStatement();
stmt.execute("select * from emp");
ResultSet rs = stmt.getResultSet();

while (rs.next()) {
  System.out.println(rs.getString("empid") + " - " +
                     rs.getString("name")  + " - " +
                     rs.getString("dept"));
  }

ResultSetMetaData md = rs.getMetaData();

System.out.println("Number of columns: " +
```

```
      md.getColumnCount());
for (int i = 1; i <= md.getColumnCount(); i++) {
    System.out.println("Column Name: "     +
      md.getColumnName(i));
    System.out.println("Nullable: "        +
      md.isNullable(i));
    System.out.println("Precision: "       +
      md.getPrecision(i));
    System.out.println("Scale: "           +
      md.getScale(i));
    System.out.println("Size: "            +
      md.getColumnDisplaySize(i));
    System.out.println("Column Type: "     +
      md.getColumnType(i));
    System.out.println("Column Type Name: "+
      md.getColumnTypeName(i));
    System.out.println("");
  }

stmt.close();
```

レコードの挿入、更新、および削除

この手順では、データベース テーブルのレコードの挿入、更新、および削除という、データベースに関する 3 つの一般的な作業を示します。これらの処理には、JDBC PreparedStatement を使います。まず、PreparedStatement を作成してから、それを実行し、閉じます。

PreparedStatement（JDBC Statement のサブクラス）を使用すると、同じ SQL を値を変えて何度でも実行できます。PreparedStatement では、JDBC の「?」構文を使用します。

```
String inssql =
   "insert into emp(empid, name, dept) values (?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(inssql);
for (int i = 0; i < 100; i++) {
  pstmt.setInt(1, i);
  pstmt.setString(2, "Person " + i);
  pstmt.setInt(3, i);
  pstmt.execute():
}
  pstmt.close();
```

PreparedStatement を使用してレコードを更新することもできます。次のサンプルでは、カウンタ「i」の値を「dept」フィールドの現在の値に追加します。

```
String updsql =
   "update emp set dept = dept + ? where empid = ?";
```

```
PreparedStatement pstmt2 = conn.prepareStatement(updsql);
 for (int i = 0; i < 100; i++) {
  pstmt2.setInt(1, i);
  pstmt2.setInt(2, i);
  pstmt2.execute();
}
pstmt2.close();
```

最後に、PreparedStatement を使用して、さきほど追加および更新されたレコードを削除します。

```
String delsql = "delete from emp where empid = ?";
PreparedStatement pstmt3 = conn.prepareStatement(delsql);
for (int i = 0; i < 100; i++) {
  pstmt3.setInt(1, i);
  pstmt3.execute();
}
pstmt3.close();
```

ストアド プロシージャとストアド関数の作り方と使い方

WebLogic jDriver for Informix を使用して、ストアド プロシージャとストアド関数の作成、使用、および削除が行えます。

次のサンプル コードでは、一連の文を実行して、ストアド プロシージャとストアド関数をデータベースから削除します。

```
  Statement stmt = conn.createStatement();
  try {stmt.execute("drop procedure proc_squareInt");}
  catch (SQLException e) {;}
  try {stmt.execute("drop procedure func_squareInt");}
  catch (SQLException e) {;}
  try {stmt.execute("drop procedure proc_getresults");}
  catch (SQLException e) {;}
  stmt.close();
```

JDBC Statement を使用してストアド プロシージャまたはストアド関数を作成してから、JDBC の ? 構文で JDBC CallableStatement（Statement のサブクラス）を使用して、`IN` および `OUT` パラメータを設定します。

ストアド プロシージャの入力パラメータは、JDBC の `IN` パラメータにマップされており、`setInt()` などの `CallableStatement.setXXX()` メソッドと JDBC PreparedStatement ? 構文で使われます。ストアド プロシージャの出力パラメー

タは、JDBC の OUT パラメータにマップされており、CallableStatement.registerOutParameter() メソッドと JDBC PreparedStatement ? 構文で使われます。パラメータを IN と OUT の両方に設定することもできます。その場合、setXXX() と registerOutParameter() の呼び出しが両方とも同じパラメータ番号に対して行われる必要があります。

次のサンプルでは、JDBC Statement を使用して ストアド プロシージャを 1 つ作成してから、そのプロシージャを CallableStatement. を使用して実行しています。registerOutParameter() メソッドを使用して、2 乗された値を入れるための出力パラメータを設定しています。

```
  Statement stmt1 = conn.createStatement();
  stmt1.execute
    ("CREATE OR REPLACE PROCEDURE proc_squareInt " +
    "(field1 IN OUT INTEGER, field2 OUT INTEGER) IS " +
    "BEGIN field2 := field1 * field1; field1 := " +
    "field1 * field1; END proc_squareInt;");
  stmt1.close();


  String sql = "{call proc_squareInt(?, ?)}";
  CallableStatement cstmt1 = conn.prepareCall(sql);

  // 出力パラメータを登録する
  cstmt1.registerOutParameter(2, java.sql.Types.INTEGER);
  for (int i = 0; i < 5; i++) {
  cstmt1.setInt(1, i);
  cstmt1.execute();
  System.out.println(i + " " + cstmt1.getInt(1) + " "
   + cstmt1.getInt(2));
} cstmt1.close();
```

次のサンプルでは、同様のコードを使用して、整数を 2 乗するストアド関数を作成して実行します。

```
  Statement stmt2 = conn.createStatement();
  stmt2.execute("CREATE OR REPLACE FUNCTION func_squareInt " +
                "(field1 IN INTEGER) RETURN INTEGER IS " +
                "BEGIN return field1 * field1; " +
                "END func_squareInt;");
  stmt2.close();


  sql = "{ ? = call func_squareInt(?)}";
  CallableStatement cstmt2 = conn.prepareCall(sql);

  cstmt2.registerOutParameter(1, Types.INTEGER);
  for (int i = 0; i < 5; i++) {
  cstmt2.setInt(2, i);
  cstmt2.execute();
  System.out.println(i + " " + cstmt2.getInt(1) +
                     " " + cstmt2.getInt(2));
```

```
}
cstmt2.close();
```

次に、sp_getmessages というストアド プロシージャを使用します（このストアド プロシージャのコードはこのサンプルには含まれていません）。sp_getmessages は、入力パラメータとしてメッセージ番号を取り、メッセージ テキストを出力パラメータ ResultSet に格納して返します。ストアド プロシージャから返された ResultSets に対して Statement.execute() および Statement.getResult() メソッドを実行してからでないと、OUT パラメータと戻りステータスは使用可能になりません。

```
String sql = "{ ? = call sp_getmessage(?, ?)}";
  CallableStatement stmt = conn.prepareCall(sql);

  stmt.registerOutParameter(1, java.sql.Types.INTEGER);
  stmt.setInt(2, 18000);           // メッセージ番号 18000
  stmt.registerOutParameter(3, java.sql.Types.VARCHAR);
```

まず、CallableStatement に対する 3 つのパラメータを設定します。

- パラメータ 1（出力のみ）はストアド プロシージャの戻り値
- パラメータ 2（入力のみ）は sp_getmessage への msgno 引数
- パラメータ 3（出力のみ）はメッセージ番号に対応して返されたメッセージ テキスト

次に、ストアド プロシージャを実行し、戻り値をチェックして、ResultSet が空かどうかを調べます。空でない場合は、ループを使用して、その内容を取り出して表示するという処理を繰り返します。

```
boolean hasResultSet = stmt.execute();
while (true)
{
ResultSet rs = stmt.getResultSet();
  int updateCount = stmt.getUpdateCount();
  if (rs == null && updateCount == -1) // 他に結果がない場合
    break;
  if (rs != null) {
    // 空になるまで ResultSet オブジェクトを処理する
    while (rs.next()) {
      System.out.println
      ("Get first col by id:" + rs.getString(1));
    }
  } else {
    // 更新件数がある
    System.out.println("Update count = " +
     stmt.getUpdateCount());
```

```
    }
    stmt.getMoreResults();
  }
```

ResultSet の処理が終わったら、次のサンプルに示すように、OUT パラメータと戻りステータスが使用可能になります。

```
int retstat = stmt.getInt(1);
String msg = stmt.getString(3);

System.out.println("sp_getmessage: status = " +
                   retstat + " msg = " + msg);
stmt.close();
```

接続の切断とオブジェクトのクローズ

接続を閉じる前に、データベースに対する変更をコミットする場合もあります。この場合は、commit() メソッドを呼び出します。

自動コミットが true（デフォルトの JDBC トランザクション モード）に設定されている場合、各 SQL 文がそれぞれトランザクションになります。しかし、このサンプルでは、Connection を作成した後に、自動コミットを false に設定しました。このモードでは、Connection は関連する暗黙的なトランザクションを常に持っており、rollback() または commit() メソッドを呼び出すと、現在のトランザクションが終了し、新しいトランザクションが開始されます。close() の前に commit() を呼び出すと、Connection を閉じる前にすべてのトランザクションが必ず完了します。

Statement、PreparedStatement、および CallableStatement を使う作業が終了したときにこれらのオブジェクトを閉じるように、アプリケーションの最後のクリーンアップ手順として、Connection オブジェクトの close() メソッドを try {} ブロック内で必ず呼び出すようにします。例外を捕捉して適切な処理を行います。このサンプルの最後の 2 行では、commit を呼び出してから接続を close します。

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

挿入後の SERIAL カラムの取り出し

挿入の後にシリアル値を取得するには、Statement.getSerialNumber() メソッドを使用します。これは、WebLogic jDriver for Informix の JDBC の WebLogic 拡張機能です。これを使用すると、テーブルに行を追加するたびに行のインデックス順をトラッキングできます。ただし、SERIAL カラムを持つテーブルを作成しなければなりません。

この拡張機能を使用するには、Statement オブジェクトを weblogic.jdbc.informix4.Statement に明示的にキャストする必要があります。

次の簡単なサンプル コードは、getSerialNumber() メソッドの使用方法を示します。

```
  weblogic.jdbc.informix4.Statement stmt =
    (weblogic.jdbc.informix4.Statement)conn.createStatement();
  String sql = "CREATE TABLE test ( s SERIAL, count INT )";
  stmt.executeUpdate(sql);

  for (int i = 100; i < 110 ; i++ ) {
    sql = "INSERT INTO test VALUES (0, " + i + ")";
    stmt.executeUpdate(sql);
    int ser = stmt.getSerialNumber();
    System.out.println("serial number is: " + ser);
}
  sql = "SELECT * from test";
  ResultSet rs = stmt.executeQuery(sql);
  while (rs.next()) {
    System.out.println("row: " + rs.getString(2) +
              " serial: " + rs.getString(1));
```

Informix INTERVAL データ型の使い方

Informix INTERVAL データ型を使用するには、weblogic.jdbc.common.InformixInterval をインポートして、ユーザのオブジェクトを weblogic.jdbc.common.InformixInterval にキャストします。

INTERVAL 値を SQL 文に入力するには、Informix INTERVAL フォーマットのリテラル文字列を使用します。prepared statement に INTERVAL 値パラメータを設定するには、preparedStatement.setString() を使用します。

INTERVAL データ型を Informix サーバから取り出すために、WebLogic jDriver for Informix は、ResultSet について次の 3 つの標準的な API をサポートしています。

- `ResultSet.getString()` は、interval の文字列表記を標準 Informix フォーマットで返します。interval が null の場合は null を返します。
- `ResultSet.getBytes()` は interval を表すためにサーバから返される実バイトを返します。
- `ResultSet.getObject()` は、`weblogic.jdbc.common.InformixInterval` 型のオブジェクトを返します。interval が null の場合は null を返します。

`InformixInterval` インタフェースは、以下のパブリック メソッドを提供します。

`String getString() throws SQLException`
: `ResultSet.getString()` と同じです。

`int getYear() throws SQLException`
: INTERVAL の符号付き年を返します。YEAR が定義されていない場合は 0 を返します。

`int getMonth() throws SQLException`
: INTERVAL の符号付き月を返します。MONTH が定義されていない場合は 0 を返します。

`int getDay() throws SQLException`
: INTERVAL の符号付き日を返します。DAY が定義されていない場合は 0 を返します。

`int getHour() throws SQLException`
: INTERVAL の符号付き時間を返します。HOUR が定義されていない場合は 0 を返します。

`int getMinute() throws SQLException`
: INTERVAL の符号付き分を返します。MINUTE が定義されていない場合は 0 を返します。

`int getSecond() throws SQLException`
: INTERVAL の符号付き秒を返します。SECOND が定義されていない場合は 0 を返します。

```
int getFraction() throws SQLException
```

10**5 の FRACTION 倍の実際の値を返します。

ResultSetMetaData メソッドの使い方

ResultSetMetaData メソッドを使用すると、Informix サーバが返したメタデータにアクセスできます。ただし、Informix サーバは以下の情報を返しません。

```
getSchemaName(int)
getTableName(int)
getCatalogName(int)
```

自動コミット モードの使い方

他のデータベース システムの属性と違い、Informix データベースの自動コミット モードは動的に設定できません。データベースが作成されたときに定義します。定義は、`Connection.setAutoCommit` への呼び出しで変更できません。非 ANSI、非ログ データベースだけが自動コミット モードを動的に変更する機能をサポートします。

JDBC 仕様では、自動コミット モードはデフォルトで true に設定されます。しかし、Informix に関しては、自動コミットのデフォルト設定を `true` に変更することはできません。Informix では、自動コミット モードを識別することしかできません。このモードを変更するには、まずデータベースを再構築する必要があります（詳細については、Informix のマニュアルの「CREATE DATABASE」を参照してください）。

データベースを再構築してからでないと自動コミットの状態を変更できないということは、トランザクションとロックの動作に影響します。さまざまな JDBC プログラムは、Informix データベースが各プログラム内でどのように作成されているかによって動作が異なります。

自動コミットに依存する前に、使用するデータベースの自動コミットがどのような設定になっているかを把握する必要があります。データベースの自動コミット モードをチェックするには、`Connection.getAutoCommit()` メソッドを使用します。このメソッドは、自動コミットが使用可能であれば true を返します。

Informix の場合、このメソッドは ANSI データベースについては false をデフォルトで返し、非 ANSI データベースについては、データベースがどのように作成されたかによって true または false を返します。

次の設定は、`Connection.setAutoCommit()` メソッドを呼び出した場合に、WebLogic jDriver for Informix がサポートしているものです。

- ANSI データベースについては、autocommit=false だけをサポートします。
- 非 ANSI データベースについては、autocommit を true にも false にも設定できます。
- ロギングなしの非 ANSI データベースについては、autocommit=true だけをサポートします。

したがって、プログラムは、使用する Informix データベースの状態に合わせて機能するはずです。

非 ANSI データベースを使用し、自動コミットを false に設定した場合、トランザクションを構成する SQL はすべて、`Connection.commit()` または `Connection.rollback()` メソッドを使用して実行されなければなりません。WebLogic jDriver for Informix は、autocommit=false ステータスをシミュレートするためにトランザクション コマンドを内部的に使用するので、明示的なトランザクション制御 BEGIN WORK、COMMIT WORK、または ROLLBACK WORK を Statement で絶対に実行しないでください。トランザクションは、必ず、Connection クラスの *`commit()`* および *`rollback()`* メソッドを使って制御してください。

ロギングなしの非 ANSI データベースでは、トランザクションをサポートしていないので、autocommit=false はサポートされません。したがって、そのようなデータベースを使用する場合は、autocommit=true だけがサポートされます。

Informix 固有の機能のサポート

WebLogic jDriver for Informix は、JDBC 仕様には含まれていない Informix 固有の機能もサポートします。このサポートによって、Informix データベース用のクライアント アプリケーションをより柔軟に作成できます。その内容は次のとおりです。

- VARCHAR/CHAR データをバイト列として取り出す

- コードセットのサポート
- Prepared Statement での Unicode ストリームの使い方

それぞれの機能について以下の節で説明します。

VARCHAR/CHAR データをバイト列として取り出す

WebLogic jDriver for Informix が提供する Informix 用の JDBC 拡張機能では、ユーザが `ResultSet.getBytes(String columnName)` メソッドと `ResultSet.getBytes(int columnIndex)` メソッドを使って VARCHAR カラムと CHAR カラムを取り出すことができます。この作業は JDBC 仕様には含まれていませんが、顧客の要望に応えて実装されました。この機能を利用するために、ResultSet をキャストする必要はありません。

コードセットのサポート

Java アプリケーションとして、WebLogic jDriver for Informix は文字列を Unicode 文字列として扱います。異なるコードセットを使って動作するデータベースと文字列をやりとりするには、`weblogic.codeset` 接続プロパティを適切な JDK コードセットに変更する必要があります。ユーザのデータベースのコードセットと JDK が提供した文字セットが直接対応しない場合は、`weblogic.codeset` 接続プロパティを最も適切な Java 文字セットに設定することができます。

たとえば、次のサンプル コードのように、cp932 コードセットを使用するには、`Driver.connect()` を呼び出す前に、Properties オブジェクトを作成し、`weblogic.codeset` プロパティを設定します。

```
java.util.Properties props = new java.util.Properties();
props.put("weblogic.codeset", "cp932");
props.put("user", "scott");
props.put("password", "tiger");

String connectUrl = "jdbc:weblogic:informix4:myDB@myHost:1493";

Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
  Connection conn =
     myDriver.connect(connectUrl, props);
```

Prepared Statement での Unicode ストリームの使い方

PreparedStatement.setUnicodeStream メソッドを使用中に、コンストラクタの String 値を使用して、独自の InputStream オブジェクトや、**weblogic.jdbc.informix4.UnicodeInputStream** オブジェクトを作成できます。次のサンプル コードは、Unicode ストリームを Informix TEXT カラムに入力する方法（上記の connectUrl オブジェクトと props オブジェクトを使用）を示します。

```
Driver myDriver = (Driver)
  Class.forName("weblogic.jdbc.informix4.Driver").newInstance();
  Connection c =
     myDriver.connect(connectUrl, props);

  PreparedStatement ps =
    c.prepareStatement("insert into dbTEST values (99,?)");

  String s = new String("\u93e1\u68b0\u897f");
  weblogic.jdbc.informix4.UnicodeInputStream uis =
    new weblogic.jdbc.informix4.UnicodeInputStream(s);

    try {
    ps.setUnicodeStream(1,uis,uis.available());
  }
  catch (java.io.IOException ioe) {
    System.out.println("--  IO Exception in setUnicodeStream");
  }
  ps.executeUpdate();
```

UnicodeInputStream からデータを取り出すために java.io.InputStream. を使用します。次に例を示します。

```
  InputStream uisout = rs.getUnicodeStream(2);
  int i=0;
  while (true) {
    try {
   i = uisout.read();  // 1 度に 1 バイトずつ UnicodeStream から読み込む
  }
  catch (IOException e) {
    System.out.println("-- IOException reading UnicodeStream");
  }
```

詳細については、samples\examples\jdbc\informix4 ディレクトリに含まれている WebLogic Server に付属の完全なサンプルを参照してください。

WebLogic jDriver for Informix の JDBC への適合状況

WebLogic jDriver for Informix は、JDBC 仕様に準拠した完全な実装です。ただし、Informix でサポートされていない機能や、使用できない機能を除きます。DatabaseMetaData インタフェースの実装に関して混乱を招く可能性があるので、すべてのメソッドをこの節に列挙します。ほとんどのメソッドは現在サポートされていますが、将来のリリースでサポートする予定のものと、（Informix の制限または実装によって）WebLogic jDriver for Informix ではサポートされないものがあります。

以下の DatabaseMetaData メソッドがサポートされています。

```
allProceduresAreCallable()
allTablesAreSelectable()
dataDefinitionCausesTransactionCommit()
dataDefinitionIgnoredInTransactions()
doesMaxRowSizeIncludeBlobs()
getCatalogSeparator()
getCatalogTerm()
getColumns()
getDatabaseProductName()
getDatabaseProductVersion()
getDefaultTransactionIsolation()
getDriverMajorVersion()
getDriverMinorVersion()
getDriverName()
getDriverVersion()
getExportedKeys()
getExtraNameCharacters()
getIdentifierQuoteString()
getImportedKeys()
getMaxBinaryLiteralLength()
getMaxCatalogNameLength()
getMaxCharLiteralLength()
getMaxColumnNameLength()
getMaxColumnsInGroupBy()
getMaxColumnsInIndex()
getMaxColumnsInOrderBy()
getMaxColumnsInSelect()
getMaxColumnsInTable()
getMaxConnections()
```

```
getMaxCursorNameLength()
getMaxIndexLength()
getMaxProcedureNameLength()
getMaxRowSize()
getMaxSchemaNameLength()
getMaxStatementLength()
getMaxStatements()
getMaxTableNameLength()
getMaxTablesInSelect()
getMaxUserNameLength()
getNumericFunctions()
getPrimaryKeys()
getProcedures()
getProcedureTerm()
getSchemas()
getSchemaTerm()
getSearchStringEscape()
getSQLKeywords()
getStringFunctions()
getSystemFunctions()
getTables()
getTableTypes()
getTimeDateFunctions()
getTypeInfo()
getURL()
getUserName()
isCatalogAtStart()
isReadOnly()
nullPlusNonNullIsNull()
nullsAreSortedAtEnd()
nullsAreSortedAtStart()
nullsAreSortedHigh()
nullsAreSortedLow()
storesLowerCaseIdentifiers()
storesLowerCaseQuotedIdentifiers()
storesMixedCaseIdentifiers()
storesMixedCaseQuotedIdentifiers()
storesUpperCaseIdentifiers()
storesUpperCaseQuotedIdentifiers()
supportsAlterTableWithAddColumn()
supportsAlterTableWithDropColumn()
supportsANSI92EntryLevelSQL()
supportsANSI92FullSQL()
supportsANSI92IntermediateSQL()
supportsCatalogsInDataManipulation()
supportsCatalogsInIndexDefinitions()
```

```
supportsCatalogsInPrivilegeDefinitions()
supportsCatalogsInProcedureCalls()
supportsCatalogsInTableDefinitions()
supportsColumnAliasing()
supportsConvert()
supportsCoreSQLGrammar()
supportsCorrelatedSubqueries()
supportsDataDefinitionAndDataManipulationTransactions()
supportsDataManipulationTransactionsOnly()
supportsDifferentTableCorrelationNames()
supportsExpressionsInOrderBy()
supportsExtendedSQLGrammar()
supportsFullOuterJoins()
supportsGroupBy()
supportsGroupByBeyondSelect()
supportsGroupByUnrelated()
supportsIntegrityEnhancementFacility()
supportsLikeEscapeClause()
supportsLimitedOuterJoins()
supportsMinimumSQLGrammar()
supportsMixedCaseIdentifiers()
supportsMixedCaseQuotedIdentifiers()
supportsMultipleResultSets()
supportsMultipleTransactions()
supportsNonNullableColumns()
supportsOpenCursorsAcrossCommit()
supportsOpenCursorsAcrossRollback()
supportsOpenStatementsAcrossCommit()
supportsOpenStatementsAcrossRollback()
supportsOrderByUnrelated()
supportsOuterJoins()
supportsPositionedDelete()
supportsPositionedUpdate()
supportsSchemasInDataManipulation()
supportsSchemasInIndexDefinitions()
supportsSchemasInPrivilegeDefinitions()
supportsSchemasInProcedureCalls()
supportsSchemasInTableDefinitions()
supportsSelectForUpdate()
supportsStoredProcedures()
supportsSubqueriesInComparisons()
supportsSubqueriesInExists()
supportsSubqueriesInIns()
supportsSubqueriesInQuantifieds()
supportsTableCorrelationNames()
```

```
supportsTransactionIsolationLevel()
supportsTransactions()
supportsUnion()
supportsUnionAll()
usesLocalFilePerTable()
usesLocalFiles()
```

以下のメソッドは実装済みで、現在検証中です。

```
getBestRowIdentifier()
getColumnPrivileges()
getTablePrivileges()
```

以下のメソッドのサポートが予定されています。

```
getIndexInfo()
supportsConvert()
```

以下のメソッドのサポートは予定されていません。

```
getCatalogs()
getCrossReference()
getProcedureColumns()
getVersionColumns()
```

参考資料

この節では、WebLogic jDriver for Informix を使用する場合に参考となるドキュメントおよびサンプル コードを示します。

ドキュメント

- 『WebLogic JDBC プログラミング ガイド』の「JDBC の概要」

  その他の WebLogic JDBC ドライバ、補足ドキュメント、サポート リソースなどに関する情報が入っています。

- 『WebLogic HTTP サーブレット プログラマーズ ガイド』の「プログラミング タスク」にあるサーバサイド Java での接続プールの使い方
- 『管理者ガイド』の「JDBC 接続の管理」

接続プール、データソース、およびマルチプールの作成など、JDBC 接続のコンフィグレーションに関する管理作業について説明しています。

- JavaSoft の「JDBC tutorial」

コード例

WebLogic jDriver for Informix では、サンプル コードを用意しています。サンプル コードは、WebLogic jDriver for Informix 配布キットの `samples\examples\jdbc\informix4` ディレクトリに入っています。