# BEA WebLogic Server

## BEA WebLogic Express™

### WebLogic jDriver for Microsoft SQL Server 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



## 限定的権利条項



## 商標または登録商標



**WebLogic jDriver for Microsoft SQL Server 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 Microsoft SQL Server のインストール

## 2. WebLogic jDriver for Microsoft SQL Server 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 Microsoft SQL Server データベース管理システム用 Type 2 Java Database Connectivity（JDBC）ドライバである WebLogic jDriver for Microsoft SQL Server 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 Microsoft SQL Server のインストール」
- 第 2 章「WebLogic jDriver for Microsoft SQL Server の使い方」

# 対象読者

このマニュアルは、データベース アクセスが必要なアプリケーションの構築に関心があ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

このマニュアル（『WebLogic jDriver for Microsoft SQL Server のインストールと使い方』）の最新版は、BEA の「e-docs」という Web サイトで、HTML 形式で入手することもできます。BEA の Web サイトにあるオンラインの HTML 版にアクセスするには、以下のリンクを使用します。

『WebLogic jDriver for Microsoft SQL Server のインストールと使い方』

このマニュアルの PDF 版は Adobe Acrobat バージョン 3.0 以上を使用して表示または印刷できます。

# 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。 |

| 表記法 | 適用 |
| --- | --- |
| 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。<br>例 :<br>`import java.util.Enumeration;`<br>`chmod u+w *`<br>`config/examples/applications`<br>`.java`<br>`config.xml`<br>`float` |
| *斜体の等幅テキスト* | コード内の変数を示す。<br>例 :<br>`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。<br>例 :<br>LPT1<br>BEA_HOME<br>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。<br>例 :<br>`java utils.MulticastTest -n *name* -a *address*`<br>`[-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。<br>例 :<br>`java weblogic.deploy [list\|deploy\|undeploy\|update]`<br>`password {application} {source}` |

| 表記法 | 適用 |
| --- | --- |
| . . . | コマンドラインで以下のいずれかを示す。<br>♦ 引数を複数回繰り返すことができる<br>♦ 任意指定の引数が省略されている<br>♦ パラメータや値などの情報を追加入力できる |
| .<br>.<br>. | 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 Microsoft SQL Server のインストール

この章では、BEA の Microsoft SQL Server 用 pure-Java Type 4 JDBC ドライバである、WebLogic jDriver for Microsoft SQL Server のインストール方法と以下の内容について説明します。



## 概要

WebLogic jDriver for Microsoft SQL Server 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 100% pure Java 実装です。この実装を使用すると、Java クライアントから Microsoft SQL Server に直接アクセスできます。このドライバは、SQL Server バージョン 6.5 および 7.0 用と SQL Server 7.0 専用の 2 つのバージョンがあります。両方のバージョンは、「始める前に」で説明する例外を除いて、同じ機能です。

その他の Type 4 JDBC ドライバと同様、WebLogic jDriver for Microsoft SQL Server は pure Java であり、ベンダがサポートするクライアント ライブラリは不要です。WebLogic jDriver for Microsoft SQL Server は、TCP/IP ネットワークを通じて SQL Server Tabular Data Stream プロトコルを使用し、SQL Server と直接通信するので、DB-Library をクライアント コンピュータにインストールする必要がありません。

# 始める前に

この節では、2 つのバージョンの WebLogic jDriver for Microsoft SQL Server の違いについて説明します。

# 使用するバージョン

BEA では、2 つのバージョンの WebLogic jDriver for Microsoft SQL Server ドライバを提供しています。Microsoft SQL Server バージョン 6.5 および 7.0 をサポートするものと、Microsoft SQL Server バージョン 7.0 および 2000 をサポートするものです。

## WebLogic jDriver for Microsoft SQL Server バージョン 6.5 および 7.0

- SQL Server バージョン 6.5 をサポートします。
- 以下の制約付きで SQL バージョン 7.0 をサポートします。
  - SQL Server は、SQL Server バージョン 6.5 クライアントから接続した場合と同じように応答し、SQL Server バージョン 6.5 のセマンティクスを実装します。たとえば CREATE TABLE SQL 文を実行した場合、DBMS は、デフォルトでは null 値を受け入れないカラムを作成します（SQL Server バージョン 6.5 では正規の動作）。
  - SQL Server 7.0 で追加されたデータ型はサポートされません。

## WebLogic jDriver for Microsoft SQL Server バージョン 7.0 および 2000

- SQL Server 7.0 をサポートします。SQL Server は、このドライバからのリクエストに対して、SQL Server バージョン 7.0 およびバージョン 2000 クライアントからのリクエストと同じように応答し、SQL Server 7.0 および 2000 のセマンティクスを実装します。たとえば（SQL Server バージョン 6.5 のセマンティクスと対照的に）DBMS は、デフォルトで null 値を受け入れるカラムを作成します。
- SQL Server バージョン 7.0 および 2000 で新しく追加されたデータ型をサポートします。

# 評価ライセンス

WebLogic jDriver for Microsoft SQL Server のライセンス機能は、WebLogic Server をインストールした BEA ホーム ディレクトリ内のライセンス 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

WebLogic Server 配布キットには、WebLogic jDriver for Microsoft SQL Server が付属しています。バージョン 7.0 および 2000 用として、`weblogic.jar` ファイルには、必要な Micrososft SQL Server jDriver クラスが入っています。ただし、バージョン 6.5 を使用している場合は、次のように、mssqlserver4v65.jar ファイルをクラスパスに追加する必要があります。

```
$ set
CLASSPATH=%CLASSPATH%;%WL_HOME%/lib/mssqlserver4v65.JAR;%WL_HOME%
/lib/weblogic.jar.
```

# 接続プールの使い方

WebLogic Server または WebLogic Express で WebLogic jDriver for Microsoft SQL Server を使用している場合、WebLogic Server の起動時に SQL Server 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（または 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 Microsoft SQL Server 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 SQL Server 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」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。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、以下を参照してください。

- 『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」

# SQL Server インストールの確認

**注意：** 使用している Microsoft SQL Server のバージョンが 6.5 または 7.0 であることを確認します。それよりも古いバージョンの SQL Server では、JDBC メタデータ関数が適切にサポートされません。また、サポートされるデータ型にも制限があります。

SQL Server に接続するには、以下の情報が必要となります。

- 有効な SQL Server アカウントのユーザ名およびパスワード
- SQL Server が稼働するマシンのホスト名または IP 番号
- DBMS が接続リクエストをリスンする TCP/IP ポートのアドレス

  Microsoft SQL Server のデフォルトのポート番号は 1433 です。ただし、サーバがリスンするポート番号は任意に設定できます。コンフィグレーション ファイルでポート番号を確認してください。ポート番号の設定の詳細については、「SQL Server に接続するためのポートの設定」を参照してください。

# SQL Server に接続するためのポートの設定

SQL Server に接続するためのホスト名とポートの設定は、SQL Server コンフィグレーション ファイルにエントリを作成することで行います。コンフィグレーション ファイルでは、論理上のサーバ名が、サーバ マシン名とポート番号に関連付けられています。WebLogic jDriver for Microsoft SQL Server では、論理上のサーバ名は使わず、ホスト名とポート番号だけを使います。

SQL Server の設定を変更するには、管理者特権が必要です。ポートを設定するには、次の操作を行います。

1. [MS SQL Server Setup] を実行します。
2. [Change Network Support] を選択します。
3. [TCP/IP] を選択します。
4. 1433 など、使用するポートを選択します。

ポートを設定したら、`telnet` を使用して、サーバがそのポートでリスンしているかどうかを確認できます。次のコマンドを入力します。

```
$ telnet hostname_or_IP_address port
```

たとえば、SQL Server が `myhost` というコンピュータのポート 1433 でリスンしているかどうかをチェックするには、次のように入力します。

```
$ telnet myhost 1433
```

サーバがこのポートでリスンしていなければ、`telnet` によりエラー メッセージが表示されます。サーバがこのポートでリスンしていれば、`telnet` による表示はなく、ホストにより接続は終了します。

ログイン情報をテストするには、次のコマンドを入力します。

```
$ isql -Uusername -Ppassword -Sserver
```

# dbping による JDBC ドライバの確認

dbping という WebLogic の Java アプリケーションを使用すると、WebLogic jDriver for Microsoft SQL Server が SQL Server に接続できるかどうかを確認できます。dbping を使用するには、WebLogic jDriver for Microsoft SQL Server のクラスが CLASSPATH（インストール手順 で説明した CLASSPATH）に指定されていることを確認します。確認した後に、次のコマンドを入力します。

```
$ java utils.dbping MSSQLSERVER4 username password
   [database@]host[:port]
```

このコマンド ラインの引数の定義は次のとおり。

- *username* はデータベース ユーザの名前
- *password* は、そのユーザのパスワード
- *database*（省略可能）は使用する SQL Server データベース
- *host* は、SQL Server が動作しているコンピュータの名前または IP 番号
- *port*（省略可能）は SQL Server がリスンしている TCP/IP ポート

たとえば、次のコマンドは、デフォルトの TCP/IP ポート、sa というログイン名、null パスワードを使用して、myhost というコンピュータの pubs という SQL Server データベースに ping を実行します。

```
$ java utils.dbping MSSQLSERVER4 sa "" pubs@myhost
```

このコマンドによる出力には、Java プログラムでデータベースの接続に使用できるコードが含まれます。

# 関連情報

この節では、参考となるドキュメントおよびサンプル サンプルを示します。

## ドキュメント

WebLogic Server で JDBC および jDriver を使用する方法の詳細については、『WebLogic JDBC プログラミング ガイド』を参照してください。

## コード例

WebLogic Server では、初期段階で参考となるサンプル コードを用意しています。サンプル コードの場所は、WebLogic Server の `samples\examples\jdbc\mssqlserver4` ディレクトリです。

# 2 WebLogic jDriver for Microsoft SQL Server の使い方

この章では、WebLogic jDriver for Microsoft SQL Server の設定方法と使い方について説明します。内容は以下のとおりです。



## WebLogic jDriver for Microsoft SQL Server とは

WebLogic jDriver for Microsoft SQL Server 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。

Type 4 ドライバはそれ以外の多くの点で Type 2 ドライバと似ています。Type 2 も Type 4 も 2 層ドライバです。つまり、クライアントがデータベースと接続するためには、それぞれにドライバをメモリにコピーする必要があります。JDBC ドライバのタイプの詳細については、『WebLogic JDBC プログラミング ガイド』の「WebLogic JDBC の概要」を参照してください。

WebLogic 環境内では、WebLogic Server とデータベースの接続に Type 2 または Type 4 の 2 層ドライバを使用します。そして、WebLogic の多層ドライバ、RMI、JTS、または Pool のいずれかを使用します。クライアントと WebLogic Server との接続には、pure-Java Type 3 多層 JDBC ドライバを使用します。

このドライバが準拠している JDBC の API リファレンスは、JavaSoft からオンラインで入手できます。

**注意：** WebLogic jDriver for Microsoft SQL Server は、以前は Connect Software の FastForward、次いで jdbcKona/MSSQLServer として知られていた製品の新しい名称です。

# SQL Server DBMS への接続

ここでは、WebLogic jDriver for Microsoft SQL Server を使用して Microsoft SQL Server に接続する方法について説明します。

# 英語以外の言語を使用して接続する方法

Microsoft SQL Server データベースで英語以外の言語を使用する場合は、`language` プロパティを指定して使用する言語を反映させる必要があります。たとえば、フランス語を指定する場合は、接続時に次のプロパティを追加します。

```
props.put ("language", "francais")
```

このプロパティを設定しないと、Primary Key Constraint Violation などの例外が発生する場合があります。

# 接続手順

次の手順に従って、WebLogic jDriver for Microsoft SQL Server を使用して SQL Server に接続するよう、アプリケーションを設定します。

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、ドット区切りで指定します。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:mssqlserver4` 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。

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 Microsoft SQL Server JDBC ドライバ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 = (java.sql.Driver)Class.forName
    ("weblogic.jdbc.mssqlserver4.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:mssqlserver4", props);
   ```

# 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.mssqlserver4.Driver").newInstance();
Connection conn =
  myDriver.connect("jdbc:weblogic:mssqlserver4", 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");
//  props.put("appname",     "MyApplication");
//  props.put("hostname",    "MyHostName");
```

最後の 2 つのプロパティ、appname および hostname は省略可能で、Microsoft SQL server へ渡され、program_name および hostname というカラム名で、sysprocesses テーブル内に読み込まれます。hostname の値は、WebLogic によって前に付加されます。

```
Driver myDriver = (java.sql.Driver)
  Class.forName
    ("weblogic.jdbc.mssqlserver4.Driver").newInstance();
  Connection conn  =
     myDriver.connect("jdbc:weblogic:mssqlserver4", 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。

# 接続オプションの追加

接続 URL の終わりに接続オプションを追加することもできます。次のサンプルのように、接続オプションと URL を分けるために疑問符（?）を用い、オプションどうしを分けるためにアンパサンド（&）を用います。

```
String myUrl =
"jdbc:weblogic:mssqlserver4:db@myhost:myport?
   user=sa&password=";
```

URL オプションの詳細については、実行時に `Driver.getPropertyInfo()` を使用して調べることができます。

# JDBC によるデータの操作

この節では、JDBC を使ったデータ操作の概要と、アプリケーションで以下の基本作業を実装するための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

詳細については、Microsoft SQL Server のマニュアルと JDBC に関する Java 指向のドキュメントを参照してください。

**注意：** WebLogic jDriver for Microsoft SQL Server は、テーブル名やカラム名の中にある疑問符 (“?”) を処理できません。エラーを回避するため、使用するデータベースでは、テーブル名やカラム名に疑問符を使用しないでください。

# 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 Microsoft SQL Server 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. 実行した結果を ResultSet に入れます。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());
stmt.close();
```

# レコードの挿入、更新、および削除

この節では、データベースのテーブルへのレコードの挿入、更新、テーブルからの削除という、データベースに関する 3 つの一般的な作業の方法を示します。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。

次のサンプル コードでは、PreparedStatement を作成します。? 構文を使用しています。

```
  String inssql =
     "insert into emp(empid, name, dept) values (?, ?, ?)";
  PreparedStatement pstmt = conn.prepareStatement(inssql);
  pstmt.setString(1, "12345");
  pstmt.setString(2, "gumby");
  pstmt.setString(3, "Cartoons");
```

次に、PreparedStatement を使用してレコードを更新します。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);

  pstmt2.setString(1, "Cartoons");
  pstmt2.setString(2, "12345);
```

最後に、PreparedStatement を使用して、先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);
  pstmt3.setString(1, "12345");
```

# 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 Microsoft SQL Server 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);
```

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);
```

次のサンプルでは、sp_getmessages（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）というストアド プロシージャを使用します。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。Statement.execute() および Statement.getResult() メソッドを使ってストアド プロシージャから返された ResultSet を処理してからでないと、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。

1. 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
2. パラメータ 2（入力のみ）は sp_getmessage への msgno 引数
3. 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ために、commit() メソッドを呼び出すと便利な場合もあります。

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として、Connection オブジェクトの `close()` メソッドを `finally {}` 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();
```

# 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 Microsoft SQL Server 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ために、このドライバは、データベースのコードセットを検出して、JDK がサポートしている文字セットを使って Unicode 文字列に変換します。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", "sa");
props.put("password", "");

String connectUrl = "jdbc:weblogic:mssqlserver4:myhost:1433";

Driver myDriver = (Driver)Class.forName
   ("weblogic.jdbc.mssqlserver4.Driver").newInstance();

Connection conn = myDriver.connect(connectUrl, props);
```

# JDBC 拡張機能

この節では、以下の JDBC 拡張機能について説明します。

# JDBC 拡張 SQL のサポート

JavaSoft の JDBC 仕様には、SQL Escape Syntax と呼ばれる SQL 拡張 が含まれています。WebLogic jDriver for Microsoft SQL Server は、拡張 SQL をサポートしています。拡張 SQL によって、DBMS 間で移植可能な共通の SQL 拡張機能にアクセスできます。

たとえば、日付から曜日を取り出す関数は、SQL 標準では定義されていません。Oracle の SQL では次のようになります。

```
select to_char(date_column, 'DAY') from table_with_dates
```

拡張 SQL を使うと、どちらの DBMS に対しても、次のようにして曜日を取り出すことができます。

```
select {fn dayname(date_column)} from table_with_dates
```

次のサンプル コードは、拡張 SQL の機能のいくつかを示します。

```
String query =
"-- This SQL includes comments and " +
    "JDBC extended SQL syntax.\n" +
"select into date_table values( \n" +
"      {fn now()},         -- current time \n" +
"      {d '1997-05-24'},   -- a date       \n" +
"      {t '10:30:29' },    -- a time       \n" +
"      {ts '1997-05-24 10:30:29.123'},  -- a timestamp\n" +
"     '{string data with { or } will not be altered}'\n" +
"-- Also note that you can safely include" +
   " { and } in comments or\n" +
"-- string data.";
Statement stmt = conn.createStatement();
stmt.executeUpdate(query);
```

拡張 SQL は、一般の SQL と区別するために中括弧（{}）で囲ってあります。コメントはダブルハイフンで始まり、改行コード（\n）で終わっています。コメント、SQL、および拡張 SQL を含む拡張 SQL のシーケンス全体は、二重引用符で囲み、Statement オブジェクトの execute() メソッドに渡します。

次のサンプル コードは、拡張 SQL を CallableStatement の一部として使用する方法を示します。

```
CallableStatement cstmt =
 conn.prepareCall("{ ? = call func_squareInt(?)}");
```

次のサンプルは、拡張 SQL 式をネストする方法を示します。

```
select {fn dayname({fn now()})}
```

サポートされている拡張 SQL 関数の一覧は、DatabaseMetaData オブジェクトから取り出すことができます。次のサンプルは、JDBC ドライバがサポートしているすべての関数をリストする方法を示します。

```
 DatabaseMetaData md = conn.getMetaData();
System.out.println("Numeric functions:     " +
    md.getNumericFunctions());
System.out.println("\nString functions:    " +
    md.getStringFunctions());
System.out.println("\nTime/date functions: " +
    md.getTimeDateFunctions());
System.out.println("\nSystem functions:    " +
    md.getSystemFunctions());
conn.close();
```

拡張 SQL の記述については、JavaSoft の JDBC 1.2 仕様の第 11 章を参照してください。

# メタデータのクエリ

メタデータのクエリは、現在のデータベースに対してのみ行うことができます。metadata メソッドは、対応する SQL Server のストアド プロシージャを呼び出します。これらのプロシージャは、現在のデータベースでのみ動作します。たとえば、現在のデータベースが *master* の場合、その接続では *master* に関連するメタデータのみ利用することができます。

# マルチスレッド アプリケーション間での Connection オブジェクトの共有

WebLogic jDriver for Microsoft SQL Server では、複数のスレッドが 1 つの Connection オプションを共有するアプリケーションを作成できます。各スレッドには、アクティブな Statement オブジェクトを持つことができます。ただし、あるスレッドで `Statement.cancel()` を呼び出すと、SQL Server は別のスレッドの Statement をキャンセルすることがあります。キャンセルされる実際の Statement は、SQL Server のタイミングの問題によって異なります。予期しないキャンセルを回避するために、各スレッドが別々の Connection オプションを持つようにすることをお勧めします。

# ストアド プロシージャの Execute キーワード

Transact-SQL 機能を使うと、ストアド プロシージャがバッチの最初のコマンドである場合に、EXECUTE キーワードを省略できます。しかし、ストアド プロシージャにパラメータがある場合、WebLogic jDriver for Microsoft SQL Server は、プロシージャ の呼び出しの前に、（JDBC 実装に固有の）変数宣言を追加します。このため、ストアド プロシージャには EXECUTE を使用することをお勧めします。EXECUTE キーワードを含まない JDBC 拡張 SQL のストアド プロシージャ構文には、この問題は影響しません。

# JDBC の制限

この節では、以下の JDBC の制限について説明します。

- cursorName( ) メソッドの非サポート
- java.sql.TimeStamp の制限
- autoCommit モードの変更
- Statement.executeWriteText( ) メソッドの非サポート

# cursorName( ) メソッドの非サポート

cursorName() メソッドはサポートされていません。

# java.sql.TimeStamp の制限

JavaSoft JDK の java.sql.TimeStamp クラスは、1970 年より後の日付に制限されます。それ以前の日付では例外となります。ただし、getString() を使って日付を取り出す場合には、WebLogic jDriver for Microsoft SQL Server は独自の date クラスを使って、この制限を乗り越えます。

# autoCommit モードの変更

Connection.setAutoCommit() に true または false 引数を付けて呼び出すと、連鎖トランザクション モードを有効または無効にできます。autoCommit が false の場合、WebLogic jDriver for Microsoft SQL Server ドライバは、前回のトランザクションがコミットまたはロールバックされると、必ずトランザクションを開始します。トランザクションは、明示的にコミットまたはロールバックで終了させる必要があります。setAutoCommit() を呼び出したときにコミットされていないトランザクションがあった場合、ドライバはそのトランザクションをロールバックしてからモードを切り換えるので、このメソッドを呼び出す前に、変更があれば必ずコミットしてください。

# Statement.executeWriteText( ) メソッドの非サポート

WebLogic Type 2 JDBC ドライバの拡張機能として、テキストまたは画像データを、テキスト ポインタを使用せずに、SQL INSERT または UPDATE 文の一部として、1 行に書き込めるようにしています。この拡張機能 Statement.exexecuteWriteText() は、DB-Library ネイティブ ライブラリを必要とするので、WebLogic jDriver for Microsoft SQL Server ドライバではサポートされていません。

ストリームを使ってテキスト データおよび画像データを読み書きするには、以下のメソッドを使用します。

- `prepareStatement.setAsciiStream()`
- `prepareStatement.setBinaryStream()`
- `ResultSet.getAsciiStream()`
- `ResultSet.getBinaryStream()`

# 参考資料

この節では、BEA WebLogic jDriver for Microsoft SQL Server 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。

# 関連ドキュメント

- 『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

  サーバサイド Java で接続プールを使用する方法についての情報が入っています。

- 『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

初期段階の参考用として、WebLogic Server では、WebLogic jDriver for Microsoft SQL Server で使用するサンプル コードを用意しています。サンプルの場所は、WebLogic Server の `samples\examples\jdbc\msqlserver4` ディレクトリです。