

Zen v14

JDBC Driver Guide

Developing Applications Using the Zen JDBC Driver



Copyright © 2021 Actian Corporation. All Rights Reserved.

このドキュメントはエンドユーザーへの情報提供のみを目的としており、Actian Corporation (“Actian”) によりいつでも変更または撤回される場合があります。このドキュメントは Actian の専有情報であり、著作権に関するアメリカ合衆国国内法及び国際条約により保護されています。本ソフトウェアは、使用許諾契約書に基づいて提供されるものであり、当契約書の条件に従って使用またはコピーすることが許諾されます。いかなる目的であっても、Actian の明示的な書面による許可なしに、このドキュメントの内容の一部または全部を複製、送信することは、複写および記録を含む電子的または機械的のいかなる形式、手段を問わず禁止されています。Actian は、適用法の許す範囲内で、このドキュメントを現状有姿で提供し、如何なる保証も付しません。また、Actian は、明示的暗示的法的に関わらず、黙示的商品性の保証、特定目的使用への適合保証、第三者の有する権利への侵害等による如何なる保証及び条件から免責されます。Actian は、如何なる場合も、お客様や第三者に対して、たとえ Actian が当該損害に関してアドバイスを提供していたとしても、逸失利益、事業中断、のれん、データの喪失等による直接的間接的損害に関する如何なる責任も負いません。

このドキュメントは Actian Corporation により作成されています。

米国政府機関のお客様に対しては、このドキュメントは、48 C.F.R 第 12.212 条、48 C.F.R 第 52.227 条第 19(c)(1) 及び (2) 項、DFARS 第 252.227-7013 条または適用され得るこれらの後継的条項により限定された権利をもって提供されます。

Actian、Actian DataCloud、Actian DataConnect、Actian X、Avalanche、Versant、PSQL、Actian Zen、Actian Director、Actian Vector、DataFlow、Ingres、OpenROAD、および Vectorwise は、Actian Corporation およびその子会社の商標または登録商標です。本資料で記載される、その他すべての商標、名称、サービス マークおよびロゴは、所有各社に属します。

本製品には、Powerdog Industries により開発されたソフトウェアが含まれています。© Copyright 1994 Powerdog Industries. All rights reserved. 本製品には、KeyWorks Software により開発されたソフトウェアが含まれています。© Copyright 2002 KeyWorks Software. All rights reserved. 本製品には、DUNDAS SOFTWARE により開発されたソフトウェアが含まれています。© Copyright 1997-2000 DUNDAS SOFTWARE LTD., all rights reserved. 本製品には、Apache Software Foundation Foundation (www.apache.org) により開発されたソフトウェアが含まれています。

本製品ではフリー ソフトウェアの unixODBC Driver Manager を使用しています。これは Peter Harvey (pharvey@codebydesign.com) によって作成され、Nick Gorham (nick@easysoft.com) により変更および拡張されたものに Actian Corporation が一部修正を加えたものです。Actian Corporation は、unixODBC Driver Manager プロジェクトの LGPL 使用許諾契約書に従って、このプロジェクトの現在の保守管理者にそのコード変更を提供します。unixODBC Driver Manager の Web ページは www.unixodbc.org にあります。このプロジェクトに関する詳細については、現在の保守管理者である Nick Gorham (nick@easysoft.com) にお問い合わせください。

GNU Lesser General Public License (LGPL) は本製品の配布メディアに含まれています。LGPL は www.fsf.org/licenses/licenses/lgpl.html でも見ることができます。

JDBC Driver Guide

2021 年 2 月

目次

| | |
|---------------------------------|-----|
| このドキュメントについて | v |
| このドキュメントの読者 | vi |
| 表記上の規則 | vii |
| 1 Zen JDBC ドライバーの概要 | 1 |
| Zen の JDBC 開発サポートの概要 | |
| Zen JDBC サポート | 2 |
| JDBC の条件 | 2 |
| JDBC の機能 | 2 |
| Zen JDBC データ型 | 3 |
| Zen JDBC ドライバーの制限 | 4 |
| サポートされない API | 4 |
| ドライバーの制限 | 4 |
| 2 Zen JDBC 2 ドライバーを使用したプログラミング | 5 |
| Zen の JDBC 2 機能の概要 | |
| 環境設定の方法 | 6 |
| CLASSPATH システム変数の設定 | 6 |
| PATH システム変数の設定 | 6 |
| JDBC ドライバーの Java 環境への読み込み | 6 |
| データ ソースの指定 | 7 |
| JDBC アプレットの開発 | 7 |
| JDBC プログラミング作業 | 8 |
| 接続文字列の概要 | 8 |
| 接続文字列の要素 | 8 |
| JDBC 接続文字列の例 | 9 |
| 文字エンコードを使用する | 10 |
| 文字エンコードの注意点 | 10 |
| Web ベース アプリケーションの開発 | 11 |
| アプレット | 11 |
| サーブレットと Java Server Page | 11 |
| JDBC 2.0 Standard Extension API | 13 |
| DataSource | 13 |
| 接続および並行制御 | 16 |
| スクロール可能な結果セット | 17 |
| JDBC プログラミング例 | 18 |
| 3 JDBC API のリファレンス | 19 |
| JDBC API のリファレンス | 20 |
| JDBC サンプル | 21 |

このドキュメントについて

このドキュメントでは、SQL ステートメント（JDBC）実行用の Java API を使用して Zen アプリケーションを開発する方法について説明します。

このドキュメントの読者

このドキュメントは、SQL ステートメント (JDBC) 実行用の Java API を使用して Zen アプリケーションを開発するユーザーを対象としています。

表記上の規則

このマニュアルでは、次の表記規則を使用しています。

| 表記上の規則 | 説明 |
|----------------|---|
| 太字 | 太字は、通常、メニュー名、ダイアログ ボックス名、コマンド、オプション、ボタンなどのグラフィカル ユーザー インターフェイス要素を示します。太字は、標準的な印刷上の強調にも使用されます。 |
| 斜体 | 斜体は、適切な値に置き換える必要がある変数を示します。たとえば、 <i>user_name</i> は実際のユーザー名に置き換えます。斜体は、マニュアル名など、標準的な印刷上の強調にも使用されます。 |
| 大文字 | 大文字のテキストは、通常、SQL 構文やコード例などを読みやすくするために使用します。オペレーティング システムによっては、大文字と小文字の区別には意味があります。そのような場合には、その説明でリテラル テキストが大文字または小文字である必要があるかどうかを記載します。 |
| 固定スペース フォント | 固定幅のテキストは、通常、構文例およびコード例を読みやすくするために使用します。また、コードの実行により返された結果やコマンド ラインに表示されるテキストにも使用します。テキストは、前後関係によって大文字または小文字で表示されます。 |
| '、"、および“” | まっすぐな引用符は、一重引用符も二重引用符も、コードおよび構文の例に使用され、一重引用符または二重引用符が必要であることを示します。カールしている二重引用符は、標準的な印刷上の引用符に使用されます。 |
| | 縦棒は OR 区切り記号を表し、いずれか 1 つを選ぶ必要のある項目を示します。下記の山かっこの説明を参照してください。 |
| [] | 角かっこはオプション項目を示します。かっこで囲まれていないコード構文は、必須の構文です。 |
| < > | 山かっこは、かっこ内の 1 項目を選択する必要があることを示します。たとえば、<yes no> と表記される場合は、"yes" または "no" のいずれかを指定する必要があります。 |
| ... | 省略記号は、前の項目を何度でも繰り返せることを示します。たとえば、[<i>parameter...</i>] は、 <i>parameter</i> を繰り返せることを示します。省略記号に続くかっこは、かっこで囲まれた内容全部を繰り返せることを示します。 |
| ::= | 記号 ::= は、ある項目が別の項目用語で定義されていることを意味します。たとえば、a::=b は、項目 "a" が "b" で定義されていることを意味します。 |
| %string% | Windows オペレーティング システムによって定義される変数です。 <i>string</i> 部分は可変テキストを表します。パーセント記号はリテラル テキストです。 |
| \$string | Linux オペレーティング システムによって定義される環境変数です。 <i>string</i> 部分は可変テキストを表します。ドル記号はリテラル テキストです。 |

Zen JDBC ドライバーの概要

1

Zen の JDBC 開発サポートの概要

以下のトピックでは、Zen Java Database Connectivity (JDBC) インターフェイスについて説明します。

- 「[Zen JDBC サポート](#)」
 - 「[JDBC の条件](#)」
 - 「[JDBC の機能](#)」
 - 「[Zen JDBC データ型](#)」
- 「[Zen JDBC ドライバーの制限](#)」
 - 「[サポートされない API](#)」
 - 「[ドライバーの制限](#)」

この Zen 機能の詳細および手順については、「[Zen JDBC 2 ドライバーを使用したプログラミング](#)」および「[JDBC API のリファレンス](#)」を参照してください。

Zen JDBC サポート

JDBC は、Java プログラマが Java を使用してデータベース アプリケーションやインターネット アプリケーションを開発するために使用できる標準 API です。これは、Java プログラミング言語で SQL ベースのデータベース アプリケーションを開発するためのインターフェイスで構成されています。JDBC インターフェイスは、Java Developer Kit の一部として含まれています。

JDBC は Java の ODBC に相当するものであり、ODBC データベースとリレーショナル データベースの大きな影響を受けます。

JDBC API の詳細については、Oracle の Web サイトを参照してください。

このセクションの残り部分では、以下の項目について説明します。

- 「[JDBC の条件](#)」
- 「[JDBC の機能](#)」
- 「[Zen JDBC データ型](#)」

JDBC の条件

Zen JDBC ドライバーは Zen と共に動作します。Enterprise Server、Cloud Server、または Workgroup エンジンを使用することができます。

JDBC の機能

以下は、Zen JDBC ドライバーの機能をまとめたものです。

- 100% Java 対応
- JDBC 4 対応、タイプ 4 ドライバー。JDBC 2 ドライバーを使用するアプリケーションにも対応しています。
- スレッドセーフ オペレーションをサポートします。
- READ_COMMITTED、SERIALIZABLE などの Zen エンジンがサポートするトランザクション分離レベルをサポートします。
- ネットワーク アクセスを減らすために結果セットのキャッシュ登録を行います。
- longvarbinary データ型（2 GB まで）でバイナリ データをサポートします。
- longvarchar および nlongvarchar データ型（2 GB まで）で long char 型データをサポートします。
- パラメーターを持つストアド プロシージャをサポートします。
- セキュリティを提供するために接続文字列を暗号化します。
- 接続文字列パラメーターを使用して、指定されたコード ページでデータベースから読み取る際のコード ページのフィルターリングをサポートします。
- 結果セットのカーソル `CONCUR_UPDATABLE`、`TYPE_SCROLL_INSENSITIVE`、および `TYPE_SCROLL_SENSITIVE` をサポートします。
- DataSource インターフェイスをサポートします。JNDI に Zen データベースを登録し、Zen 固有のドライバー機能からアプリケーションを保護します。
- ParameterMetaData インターフェイスをサポートします。

Zen JDBC データ型

Zen JDBC データ型を指定するのに使用される数値識別子は、場合によっては、JDBC 標準の識別子と異なることがあります。次の表は、完全な一覧です。

| データ型 | 識別子 |
|---------------|-----|
| AUTOTIMESTAMP | 93 |
| BFLOAT4 | 7 |
| BFLOAT8 | 8 |
| BIGIDENTITY | -5 |
| BIGINT | -5 |
| BINARY | -2 |
| BIT | -7 |
| CHAR | 1 |
| CURRENCY | 3 |
| DATE | 91 |
| DATETIME | 93 |
| DECIMAL | 3 |
| DOUBLE | 8 |
| IDENTITY | 4 |
| INTEGER | 4 |
| LONGVARBINARY | -4 |
| LONGVARCHAR | -1 |
| MONEY | 3 |

| データ型 | 識別子 |
|---------------|-----|
| MONEY | 3 |
| NCHAR | -8 |
| NLONGVARCHAR | -10 |
| NUMERIC | 2 |
| NVARCHAR | -9 |
| REAL | 7 |
| SMALLIDENTITY | 5 |
| SMALLINT | 5 |
| TIME | 92 |
| TIMESTAMP | 93 |
| TIMESTAMP2 | 93 |
| TINYINT | -6 |
| UBIGINT | -5 |
| UINTeger | 4 |
| USMALLINT | 5 |
| UTINYINT | -6 |
| VARCHAR | 12 |
| | |

Zen JDBC ドライバーの制限

サポートされない API

Zen JDBC ドライバーは次の JDBC インターフェイスをサポートしていません。

- Array
- Blob
- Clob
- NClob
- Ref
- RowId
- SQLXML
- Struct
- SQLData
- SQLInput
- SQLOutput
- URL

これらがサポートされないのは、Zen エンジンがその基盤にある SQL 3 データ型を現在サポートしていないためです。

ドライバーの制限

- "out" パラメーターで long データ型を使用できません。
- 実際の最小フェッチサイズは 2 行です。
- 結合で更新可能な結果セットを持つことはできません。
- "group by" で更新可能な結果セットを持つことはできません。
- JDBC ドライバーは、データを UnicodeBig または UnicodeLittle 形式で保存しません。
- サポートされる保持機能は HOLD_CURSORS_OVER_COMMIT のみです。
- プールされたステートメントはサポートされていません。
- 名前付きパラメーターはサポートされていません。

Zen JDBC 2 ドライバーを使用したプログラミング

2

Zen の JDBC 2 機能の概要

以下のトピックでは、JDBC 2.0 について説明します。

- 「[環境設定の方法](#)」
- 「[JDBC プログラミング作業](#)」
- 「[Web ベース アプリケーションの開発](#)」
- 「[JDBC 2.0 Standard Extension API](#)」
- 「[接続および並行制御](#)」
- 「[スクロール可能な結果セット](#)」
- 「[JDBC プログラミング例](#)」

環境設定の方法

このトピックでは、JDBC インターフェイスを使用する場合の適切な設定について説明します。

- 「[CLASSPATH システム変数の設定](#)」
- 「[PATH システム変数の設定](#)」
- 「[JDBC ドライバーの Java 環境への読み込み](#)」
- 「[データ ソースの指定](#)」
- 「[JDBC アプレットの開発](#)」

オンライン チュートリアル「[Accessing Actian Zen with Java and JDBC](#)」(Java と JDBC を使用した Actian Zen へのアクセス) トピック (英語) も利用できます。

CLASSPATH システム変数の設定

Java アプリケーションおよびアプレットが Zen JDBC ドライバーを認識できるように、CLASSPATH 環境変数に pvjdbc2.jar、pvjdbc2x.jar、および jpscs.jar ファイルを含めるように設定してください。Windows プラットフォームでは、デフォルトでこれらのファイルは Program Files フォルダー下の <インストールディレクトリ>\bin に存在します。Linux、macOS、および Raspbian では、このファイルはデフォルトで /usr/local/actianzen/bin/lib にインストールされます。

Windows の場合

```
set CLASSPATH=%CLASSPATH%;<pvjdbc2.jar ディレクトリのパス>/pvjdbc2.jar
set CLASSPATH=%CLASSPATH%;<pvjdbc2x.jar ディレクトリのパス>/pvjdbc2x.jar
set CLASSPATH=%CLASSPATH%;<jpscs.jar ディレクトリのパス> /jpscs.jar
```

Windows のシステム設定で環境変数を編集することによって、変更を永続化することもできます。

Linux および macOS の場合

```
export CLASSPATH=$CLASSPATH:<pvjdbc2.jar ディレクトリのパス>/pvjdbc2.jar
export CLASSPATH=$CLASSPATH:<pvjdbc2x.jar ディレクトリのパス>/pvjdbc2x.jar
export CLASSPATH=$CLASSPATH:<jpscs.jar ディレクトリのパス>/jpscs.jar
```

PATH システム変数の設定

共有メモリを使用してデータベース エンジンに接続する場合、JDBC ドライバーは pvjdbc2.dll を見つける必要があります。PATH 環境変数に DLL の場所を含めてください。

```
set PATH=%PATH%;<pvjdbc2.dll ディレクトリのパス>
```

ソケットを使用してデータベースに接続する場合、DLL は必要とされません。

JDBC ドライバーの Java 環境への読み込み

CLASSPATH 変数を設定すると、Java アプリケーションから Zen JDBC ドライバーを参照することができます。これは、次の java.lang.Class クラスを使用して行います。

```
Class.forName("com.pervasive.jdbc.v2.Driver");
```

IPv6 環境

IPv6 のみの環境で Zen JDBC ドライバーを使用する場合は、Java JRE 1.7 も使用することをお勧めします。IPv6 のみの環境でアプリケーションが Java JRE 1.6 より前のバージョンを使用した場合、ライセンス数に関する問題やクライアント追跡の問題が生じる可能性があります。

また、次のような条件が組み合わさった場合にも、ライセンス数に関する問題が生じる可能性があります。

- 1 1 台のマシンが Zen JDBC ドライバーを使用して複数のアプリケーションを実行しており、それらのアプリケーションが IPv4 アドレスと IPv6 アドレスを併用してデータベース エンジンに接続している。
- 2 マシンの SYSTEM PATH に pvjdbc2.dll の場所が含まれていない。「[PATH システム変数の設定](#)」も参照してください。

データ ソースの指定

Java 環境に PervasiveDriver クラスを読み込んだ後、Zen データベースに接続するために URL 形式の文字列を java.sql.DriverManager クラスに渡す必要があります。JDBC ドライバーの URL の構文は次のとおりです。

```
jdbc:pervasive://<machinename>:<portnumber>/<datasource>
```

| | |
|---------------|---|
| <machinename> | データベース エンジンを実行するマシンのホスト名または IP アドレス。 |
| <portnumber> | データベース エンジンが受信を行うポート。このポートのデフォルト値は 1583 です。 |
| <datasource> | アプリケーションが使用する予定のデータベース サーバー上の ODBC DSN の名前。 |

たとえば、Zen エンジンが DBSERV というマシン上にあって、Demodata データベースに接続したい場合の URL は次のようになります（サーバーがデフォルトのポートを使用するように設定されているものとします）。

```
jdbc:pervasive://dbserv/demodata
```

DriverManager クラスを使用してデータベースに接続するには、次の構文を使用します。

```
Connection conn = DriverManager.getConnection("jdbc:pervasive://dbserv:1583/demodata", loginString, passwordString);
```

loginString はユーザーのログイン名を表す文字列で、*passwordString* はユーザーのパスワードを表す文字列です。



メモ JDBC アプレットおよびアプリケーションがデータにアクセスするためには、指定したホスト マシンで Zen エンジンが実行されている必要があります。

JDBC アプレットの開発

JDBC を使用して Web ベース アプリケーションを開発するには、アプレット クラスを含むコード ベース ディレクトリに JDBC jar ファイルを置いておく必要があります。

たとえば、MyFirstJDBCApplet と呼ぶアプリケーションを開発する場合は、MyFirstJDBCApplet クラスを含むディレクトリに pvjdbc2.jar ファイルを置く必要があります。たとえば、C:\inetpub\wwwroot\myjdbc\ となります。

これにより、クライアント Web ブラウザーは JDBC ドライバーをダウンロードし、データベースに接続できます。

また、<applet> タグ内に archive パラメーターを指定する必要があります。たとえば、次のようになります。

```
<applet CODE="MyFirstJDBCApplet.class"
        ARCHIVE="pvjdbc2.jar" WIDTH=641 HEIGHT=554>
```

アプレットのホストとなる Web サーバーで Zen エンジンが実行されている必要があることに留意してください。

JDBC プログラミング作業

ここでは、JDBC プログラミングの重要なコンセプトに焦点を当てます。

接続文字列の概要

JDBC ドライバーは、データベースの接続に URL を必要とします。JDBC ドライバー用の URL 構文は以下のとおりです。

```
jdbc:pervasive://machinename:port number/datasource[;encoding=;encrypt=;encryption=]
```

machinename は、Zen サーバーを実行するマシンのホスト名または IP アドレスです。

port number は、Zen サーバーが受信を行うためのポートです。このポートのデフォルト値は 1583 です。

datasource は、アプリケーションが使用する予定の Zen サーバー上の ODBC エンジン データ ソースの名前です。

encoding= は、文字エンコードです。これは指定したコード ページを介して読み込んだデータにフィルターをかけることができます。これによりデータが正しく書式設定およびソートされます。値 "auto" は、接続時にデータベースのコード ページを決定し、エンコードをその文字エンコードに設定します。また、値 "auto" により、SQL クエリ内の NCHAR リテラルが保持されます。"auto" でない場合は、SQL クエリはデータベースのコード ページに変換されます。

encrypt= は、JDBC ドライバーが暗号化ネットワーク通信（ワイヤ暗号化とも呼ばれます）を使用する必要があるかどうかを決定します。

encryption= は、JDBC ドライバーが許可する暗号化の最低レベルを指定します。



メモ JDBC アプリケーションを実行するためには、Zen エンジンに指定したホストで実行されている必要があります。

接続文字列の要素

JDBC を使用して Zen データベースに接続する方法を次に示します。

ドライバ クラスパス

```
com.pervasive.jdbc.v2
```

ドライバを読み込むステートメント

```
Class.forName("com.pervasive.jdbc.v2.Driver");
```

URL

```
jdbc:pervasive://server:port/DSN[;encoding=;encrypt=;encryption=]
```

または

```
jdbc:pervasive://server:port/DSN[?pvtranslate=&encrypt=&encryption=]
```


表 1 接続文字列の要素

| 引数 | 説明 |
|------------|--|
| server | ID または URL を使用したサーバー名。 |
| port | リレーショナル エンジンのデフォルトのポートは 1583 です。ポートが指定されていない場合、このデフォルトが使用されます。 |
| DSN | 通常の ODBC メソッドを使用してサーバーで設定する DSN の名前。 |
| encoding | 「 文字エンコードを使用する 」を参照してください。 |
| encrypt | <p>JDBC ドライバーが暗号化ネットワーク通信（ワイヤ暗号化とも呼ばれます）を使用する必要があるかどうかを決定します。（『<i>Advanced Operations Guide</i>』の「ワイヤ暗号化」を参照してください。）</p> <p>値 : always（常時）、never（しない）</p> <p>このオプションを指定しなかった場合、ドライバにはサーバーの設定が反映されます。これは、" 必要な場合 " と同等です。</p> <p>値 "always" を指定した場合、JDBC ドライバーは暗号化を使用します。ただし、サーバーがワイヤ暗号化を許可していない場合はエラーを返します。値 "never" を指定した場合、JDBC ドライバーは暗号化を使用しません。サーバーがワイヤ暗号化を要求した場合はエラーを返します。</p> <p>JDBC ドライバーでワイヤ暗号化を使用するには、別の JAR ファイルが classpath に必要となります。この JAR ファイル jpjcs.jar はデフォルトでインストールされ、Java Cryptography Extensions (JCE) を使用します。</p> |
| encryption | <p>JDBC ドライバーが許可する暗号化の最低レベルを決定します。</p> <p>値 : low（低）、medium（中）、high（高）</p> <p>デフォルト : medium（中）</p> <p>これらの値はそれぞれ 40 ビット、56 ビット、および 128 ビット暗号化に対応しています。</p> <p>次の例では、JDBC ドライバーは UTF-8 エンコードを使用し、常に暗号化を要求し、最低でも " 低 " レベルの暗号化を必要とすることを指定しています。そうでない場合はエラー コードを返します。</p> <p>jdbc:pervasive://host/demodata?encoding=UTF-8&encrypt= always&encryption=low</p> |

JDBC 接続文字列の例

JDBC ドライバーを使用して Zen データベースに接続する方法を次に示します。

```
// JDBC ドライバーを読み込みます。
Class.forName("com.pervasive.jdbc.v2.Driver")

// JDBC の URL 構文
// jdbc:pervasive://<ホスト名または IP アドレス> :
//   <ポート番号（デフォルト 1583）>/<ODBC エンジン DSN>

String myURL = "jdbc:pervasive://127.0.0.1:1583/demodata";
try
{
    // m_Connection = DriverManager.getConnection(myURL,username, password);
}
catch(SQLException e)
{
    e.printStackTrace();

    // その他の例外処理
}
```

文字エンコードを使用する

Java は文字列にワイド文字を使用します。データベースのエンコードがワイド文字でない（たとえば、UCS-2 である）場合は、データベース エンジンによって文字データを正しく変換するために、ドライバーはデータベースのコード ページを知っている必要があります。データベースの文字データ エンコードは、ドライバー マネージャーに渡す接続文字列の中で "encoding" 属性を使用して指定します。

encoding 属性

encoding 属性は、文字データの変換に使用する特定のコード ページを指定します。encoding 属性を "auto" に設定することで、これを自動化できます。これは、データベースで使用されているコード ページを自動的に使用するよう、ドライバーに指示します。特定のコード ページを指定することもできます。encoding 属性が指定されない場合は、クライアント マシンに用いられているデフォルトのオペレーティング システムのコード ページが使用されます。これはクライアントとサーバーが同じオペレーティング システムのエンコードを使用していることが前提です。

また、encoding 属性を "auto" に設定すると、SQL クエリ テキストが、データベース コード ページのエンコードではなく UTF-8 エンコードを使用して送信されるようになります。これにより、クエリ テキスト内の NCHAR 文字列リテラルが保持されます。

文字エンコードの使用例

```
public static void main(String[] args)
{
    // latin 2 エンコードを指定
    String url = "jdbc:pervasive://MYSERVER:1583/SWEDISH_DB;encoding=cp850";
    try{
        Class.forName("com.pervasive.jdbc.v2.Driver");
        Connection conn = DriverManager.getConnection(url);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("select * from SwedishTable");
        rs.close();
        stmt.close();
        conn.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

文字エンコードの注意点

Zen JDBC ドライバーは、コード ページの Java ネイティブ サポートを使用します。サポートされるコード ページの一覧は Oracle Corporation の Web サイトから入手できます。

Web ベース アプリケーションの開発

ここでは、Zen JDBC ドライバーを使用して Web ベースのアプリケーションを作成する方法を説明します。

アプレット

JDBC を使用して Web ベース アプリケーションを開発するには、アプレット クラスを含むコードベース ディレクトリに JDBC jar ファイルを置いておく必要があります。

たとえば、MyFirstJDBCApplet というアプリケーションを開発する場合は、MyFirstJDBCApplet クラスを含むディレクトリに pvjdbc2.jar ファイルまたは Zen jdbc パッケージを置く必要があります。たとえば、C:\inetpub\wwwroot\myjdbc と指定できます。これにより、クライアント Web ブラウザーはネットワークから JDBC ドライバーをダウンロードし、データベースに接続できます。

また、JAR ファイルを使用する場合、<APPLET> タグ内にアーカイブ パラメーターを設定する必要があります。たとえば、次のようになります。

```
<applet CODE="MyFirstJDBCApplet.class" ARCHIVE="pvjdbc2.jar" WIDTH=641 HEIGHT=554>
```



メモ Zen エンジン、アプレットのホストとなる Web サーバー上で運用しなければなりません。

サーブレットと Java Server Page

JSP を使用して、Zen JDBC ドライバーを用いる Web ベースのアプリケーションを作成することができます。

次に示すのは、Zen に含まれるサンプル データベース Demodata のテーブルの 1 つを表示する Java Server Page の例です。

```
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>

<%
Class.forName("com.pervasive.jdbc.v2.Driver");
Connection con = DriverManager.getConnection("jdbc:pervasive://localhost:1583/
    demodata");
PreparedStatement stmt = con.prepareStatement("SELECT * FROM Course ORDER BY Name");
ResultSet rs = stmt.executeQuery();
%>

<html>
<head>
<title>JSP Sample</title>
</head>
<body>

<h1>JSP Sample</h1>
<h2>Course table in Demodata database</h2>
<p>
この例は、Demodata データベースの Course テーブルを開き、
そのテーブルの内容を表示します
</p>

<table border=1 cellpadding=5>
<tr>
<th>Name</th>
<th>Description</th>
<th>Credit Hours</th>
```

```
<th>Department Name</th>
</tr>

<% while(rs.next()) { %>
    <tr>
        <td><%= rs.getString("Name") %></td>
        <td><%= rs.getString("Description") %></td>
        <td><%= rs.getString("Credit_Hours") %></td>
        <td><%= rs.getString("Dept_Name") %></td>
    </tr>
<% } %>

</table>

</body>
</html>
```

サーブレットと JSP に関する情報

サーブレットと JSP の詳細については、Oracle の Web サイトを参照してください。

JDBC 2.0 Standard Extension API

接続文字列はベンダー固有であるため、Java は DataSource インターフェイス仕様を作成しました。これは、Java レジストリとして機能する JNDI を利用します。DataSource インターフェイスにより、JDBC 開発者は名前付きデータベースを作成することができます。開発者は、JNDI にデータベース名とベンダー固有のドライバ情報を登録します。そうすると、JDBC アプリケーションはデータベースをまったく知る必要がなく、「ピュア」な JDBC となります。

Zen JDBC ドライバーは JDBC 2.0 Standard Extension API をサポートしています。現在、Zen JDBC ドライバーは次のインターフェイスをサポートしています。

- javax.sql.ConnectionEvent
- javax.sql.ConnectionEventListener
- javax.sql.ConnectionPoolDataSource
- javax.sql.DataSource
- javax.sql.PooledConnection



メモ これらのインターフェイスは、コア JDBC API を 100% Pure Java として維持するため、pvjdbc2x.jar に別にパッケージされています。

現時点では、Zen は RowSet インターフェイスの実装を提供していませんが、Zen JDBC ドライバーは Oracle の RowSet インターフェイスの実装で検証済みです。

DataSource

Java はアプリケーション開発者がドライバに依存しないアプリケーションを作成する方法を提供しています。DataSource インターフェイスと JNDI を使用することにより、アプリケーションは標準の方法でデータにアクセスでき、接続文字列のようなドライバ固有の要素をなくすることができます。DataSource インターフェイスを使用するには、データベースを JNDI サービス プロバイダーに登録する必要があります。そうすると、アプリケーションはデータベースに名前でアクセスすることができます。

次に DataSource インターフェイスの使用例を挙げます。

```
// このコードは、DataSource を
// 登録するために、管理者が実行する
// 必要があります。
// このサンプルは、Oracle の参照 JNDI
// 実装を使用します。

public void registerDataSources()
{
    // この例では JNDI ファイルシステム
    // オブジェクトをレジストリとして使用します。

    Context ctx;
    jndiDir = "c:¥¥jndi";

    try
    {
        Hashtable env = new Hashtable (5);
        env.put (Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");

        env.put(Context.PROVIDER_URL, jndiDir);
        ctx = new InitialContext(env);
    }
}
```

```

    }
    catch (Exception e)
    {
        System.out.println(e.toString());
    }

    // demodata を通常のデータ ソースとして登録
    com.pervasive.jdbc.v2.DataSource ds = new com.pervasive.jdbc.v2.DataSource();
    String dsName = "";

    try
    {
        // ユーザー名、パスワード、ドライバーの種類、およびネットワーク プロトコルを設定
        ds.setUser("administrator");
        ds.setPassword("admin");
        ds.setPortNumber("1583");
        ds.setDatabaseName("DEMODATA");
        ds.setServerName("127.0.0.1");
        ds.setDataSourceName("DEMODATA_DATA_SOURCE");
        ds.setEncoding("cp850");
        dsName = "jdbc/demodata";

        // バインド
        try
        {
            ctx.bind(dsName,ds);
            System.out.println(" バウンド データ ソース [" + dsName + "]);
        }
        catch (NameAlreadyBoundException ne)
        {
            System.out.println(" データ ソース [" + dsName + "] は既にバインドされています ");
        }
        catch (Throwable e)
        {
            System.out.println("JNDI バインド エラー :");
            throw new Exception(e.toString());
        }
    }
}

// この DataSource をアプリケーションで使用するには、次のコードを実行することが必要

public DataSource lookupDataSource(String ln) throws SQLException
{
    Object ods = null;
    Context ctx;

    try
    {
        Hashtable env = new Hashtable (5);
        env.put (Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");

        // JNDI ディレクトリを作成し、
        // その名前を返す
        // ただしそのディレクトリがまだ存在していない場合のみ

        String jndiDir = "c:¥¥jndi";

```

```

        env.put(Context.PROVIDER_URL, jndiDir);
        ctx = new InitialContext(env);
    }
    catch (Exception e)
    {
        System.out.println(e.toString());
    }
    try
    {
        ods = ctx.lookup(ln);
        if (ods != null)
            System.out.println(" データ ソース [" + ln + "]"+" が見つかりました ");
        else
            System.out.println(" データ ソース [" + ln + "]"+" が見つかりません ");
    }
    catch (Exception e)
    {
        throw new SQLException(e.toString());
    }

    return (DataSource)ods;
}

// ConnectionPoolDataSource も
// 同様に扱われることに注意

```

接続および並行制御

単一の Zen JDBC 接続は、簡単に複数スレッドをサービスすることができます。ただし、接続がスレッドセーフのとき、その接続によって作成されたオブジェクトはスレッドセーフにはなりません。たとえば、ユーザーは4つのスレッドを作成できます。これらのスレッドは、それぞれの Statement オブジェクトを与えられます（すべて同じ Connection オブジェクトによって作成されます）。4つのスレッドはすべて同一接続を使用し、同時にデータを送ったり要求したりすることができます。これは、4つの Statement オブジェクトが同一 Connection オブジェクトを参照し、読み取りと書き込みがこのオブジェクト上で同期することにより、動作します。ただし、このアクセスが同期していなければ、1番目のスレッドは2番目のスレッドの Statement オブジェクトにアクセスすることはできません。このことは、JDBC API 内のほかのすべてのオブジェクトにも当てはまります。

スクロール可能な結果セット

スクロール可能な結果セットにより、結果セット内を前方または後方へ移動することができます。このタイプの移動は、それぞれ相対または絶対に分類されます。`first()`、`last()`、`beforeFirst()`、`afterLast()`、および `absolute()` メソッドを呼び出すと、スクロール可能な結果セットに絶対的に位置付けることができます。相対的な位置付けは `next()`、`previous()`、および `relative()` メソッドを使用します。

また、スクロール可能な結果セットは更新可能または読み取り専用にすることができます。これは、その基盤にあるデータベースに変更を加えることができるかどうかに関係します。そのほかの用語として、センシティビティは、これらの変更が現在の結果セットに影響するかどうかに関連します。

センシティブな結果セットは、これに行われた Insert、Update、Delete の結果をすべて反映します。Zen の場合、インセンシティブな結果セットはデータの静的なスナップショットであるため、これに加えられた変更を一切反映しません。言い換えると、自身またはほかの人が行った変更を知ることができません。

センシティブおよびインセンシティブな結果セットは、それぞれ ODBC の動的および静的に対応します。センシティブな結果セットは、トランザクション分離レベルに `READ_COMMITTED` が設定されている場合、自身で行った変更およびほかの人が行った変更を反映します。トランザクション分離レベルは、Connection オブジェクトを使用して設定します。結果セットのタイプはステートメント作成で設定されます。

結果セットがインセンシティブの場合、現在の行番号を判断するために `getRow()` メソッド呼び出しを行うことができます。また、インセンシティブな結果セットでは、`isLast()`、`isFirst()`、`isBeforeFirst()`、および `isAfterLast()` 呼び出しを行うことができます。センシティブな結果セットでは、`isBeforeFirst()` および `isAfterLast()` のみを呼び出すことができます。また、インセンシティブな結果セットでは、ドライバーはユーザーが指示したフェッチ方向を受け入れます。センシティブな結果セットでは、ドライバーは指示されたフェッチ方向を無視します。

JDBC プログラミング例

次の例では、"MYSERVER" サーバー上の "DB" という名前のデータベースへの接続を作成します。それから、その接続上にセンシティブで変更可能な Statement オブジェクトを作成します。その Statement オブジェクトを使用して "SELECT" クエリを実行します。結果セットオブジェクトが取得されると、"absolute" 呼び出しを行い、5 番目の行に移動します。5 番目の行の 2 番目の列が整数値 101 に変更されると、"updateRow" 呼び出しで実際にその更新を行います。

```
Class.forName("com.pervasive.jdbc.v2.Driver");
Connection conn=
DriverManager.getConnection("jdbc:pervasive://MYSERVER:1583/DB");

Statement stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

ResultSet rs =
m_stmt.executeQuery("SELECT * FROM mytable");

rs.absolute(5);
rs.updateInt(2, 101);
rs.updateRow();

rs.close();
stmt.close();
conn.close();
```

JDBC API のリファレンス

3

JDBC API は、Java プログラミング言語を使用した、データベースとの標準インターフェイスです。このインターフェイスについて以下のトピックで説明します。

- [「JDBC API のリファレンス」](#)
- [「JDBC サンプル」](#)

JDBC API のリファレンス

JDBC は、Oracle の Web サイトで文書化されている標準 API です。JDBC および JDBC に関するドキュメントの内容を参照してください。ただし、「[Zen JDBC ドライバーの制限](#)」に記述されているドライバーの API の制約に注意してください。

その他、JDBC プログラミングについての有用なサイトとして jakarta.apache.org の Tomcat 情報や www.apache.org の Apache 情報があります。

JDBC ドライバーを用いたプログラミングに関する概念情報については、以下のトピックを参照してください。

- 「[Zen JDBC ドライバーの概要](#)」
- 「[Zen JDBC 2 ドライバーを使用したプログラミング](#)」

JDBC サンプル

Zen SDK の JDBC サンプルは、Web ダウンロードにより入手可能です。デフォルトの場所にインストールした場合、保存場所は `file_path¥Zen¥sdk¥jdbc¥samples` になります。

Zen ファイルのデフォルトの保存場所については、『*Getting Started with Zen*』の「[ファイルはどこにインストールされますか？](#)」を参照してください。

