Pervasive PSQL v11

JDBC Driver Guide

Developing Applications Using the Pervasive JDBC Driver

PERVASIVE®

免責事項

Pervasive Software Inc. は、本ソフトウェアおよびドキュメントの使用を、利用者またはその会社に対して「現状のまま」で、かつ同梱の使用許諾契約書に記載の契約条件によってのみ許諾するものです。Pervasive Software Inc. は、いかなる場合にも本ソフトウェアおよび本マニュアルに記載された内容に関するその他の一切の保証を、明示的にも黙示的にも行いません。Pervasive Software Inc. は、市場性、権利、特定の目的に対する適合性、あるいは一連の取引業務や職業的な使用に関する問題などに対し、一切の保証を行わないことを明示するとともに、利用者およびその会社がこれに同意したものとします。

商標

JDBC Driver Guide
2013 年 1 月

目次

表

このマニュアルについて

このマニュアルは、SQL ステートメント（JDBC）実行用の Java API を使用する Pervasive PSQL アプリケーションを開発するための手引書です。

このマニュアルの読者

このマニュアルは、SQL ステートメント（JDBC）実行用の Java API を使用する Pervasive PSQL 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。

このマニュアルの構成

この『JDBC Driver Guide』では、まず新機能の概要を述べ、次にその詳細が記載される章へのリンクを示します。章の構成は以下のとおりです。

- 第 1 章 「Pervasive JDBC ドライバーの概要」

  この章では、Pervasive リレーショナル データベース エンジンを使用した Java プログラミングのための JDBC API について概説します。

- 第 2 章 「Pervasive JDBC 2 ドライバーを使用したプログラミング」

  この章では、Pervasive リレーショナル データベース エンジンを使用した Java プログラミングのための JDBC API について概説します。

- 第 3 章 「JDBC API のリファレンス」

  この章では、JDBC API のリファレンスへのリンクについて説明します。

このマニュアルの巻末には索引が用意されています。

表記上の規則

特段の記述がない限り、コマンド構文、コード、およびコード例では、以下の表記が使用されます。

大文字小文字の区別	通常、コマンドと予約語は、大文字で表記されます。本書で別途記述がない限り、これらの項目は大文字、小文字、あるいはその両方を使って入力できます。たとえば、MYPROG、myprog、または MYprog と入力することができます。
太字	太字で表示される単語には次のようなものがあります。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、ステートメントなど。
固定幅フォント	固定幅フォントは、コマンド構文など、ユーザーが入力するテキストに使われます。
[ ]	省略可能な情報には、［*log_name*］のように、角かっこが使用されます。角かっこで囲まれていない情報は必ず指定する必要があります。
|	縦棒は、[*file name* | *@file name*] のように、入力する情報の選択肢を表します。
< >	＜＞ は、/D=<5|6|7> のように、必須項目に対する選択肢を表します。
変数	*file name* のように斜体で表されている語は、適切な値に置き換える必要のある変数です。
…	［*parameter…*］のように、情報の後に省略記号が続く場合は、その情報を繰り返し使用できます。
::=	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。

Pervasive JDBC ドライバーの概要

1

Pervasive の JDBC 開発サポートの概要

この章では、Pervasive の JDBC インターフェイスについて説明します。以下の項目が含まれます。

- 「Pervasive PSQL JDBC サポート」
- 「Pervasive JDBC 2 ドライバー」
- 「Pervasive JDBC ドライバーの制限」

Pervasive PSQL 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 アプリケーションとインターネット アプリケーションの開発に使用する標準 API です。JDBC は、バージョン 1.1 以上の Sun Microsystems の Java Developer Kit に付属しています。JDBC は、開発者が Java プログラミング言語を用いた SQL ベースのデータベース アプリケーション開発に使用できるインターフェイスを中心に構成されているパッケージです。

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 Web サイト java.sun.com を参照してください。

JDBC の条件

Pervasive JDBC ドライバーは Pervasive PSQL と共に動作します。サーバーまたはワークグループ エンジンを使用することができます。

JDBC の機能

以下は、Pervasive JDBC ドライバーの機能についての概略です。

- 100% Java 対応
- JDBC 2 対応、タイプ 4 ドライバー
- 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 Pervasive PSQL 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 データ型（2 GB まで）で long char 型データをサポートします。
- パラメーターでストアド 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。
- 追加されたセキュリティのデコードに不可欠なネイティブ　バイナリ形式でサーバーとクライアントの間のデータ フローを伝送します。

Pervasive JDBC 2 ドライバー

JDBC 2 API が Pervasive PSQL でサポートされています。JDBC バージョン 2 には、JDK バージョン 1.2 以上が付属しています。Pervasive JDBC ドライバーは、この JDBC 2 標準をオプションのパッケージも含めてサポートします。

表 1 Pervasive JDBC ドライバーの新しい機能の概要

機能要素	注記
接続文字列	接続で、どのコード ページを使用するかを指定することができるパラメーターが追加されました。
カーソルのサポートの向上	このドライバーは、CONCUR_UPDATABLE、TYPE_SCROLL_INSENSITIVE および TYPE_SCROLL_SENSITIVE をサポートします。
DataSource インターフェイスのサポート	JNDI で Pervasive PSQL を登録し、Pervasive 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。

仕様

Pervasive JDBC ドライバーは、純粋な Java タイプ 4 のネット プロトコル ベース ドライバーです。これは中核の JDBC 2.x の条件に適合しています。このドライバーは機能が ODBC クライアント ドライバーに似ており、バルク処理についてはサーバー側の Pervasive PSQL エンジンの ODBC インターフェイスにより異なります。

Pervasive JDBC 1 ドライバーからのアップグレード

以前の Pervasive JDBC 1 ドライバーをお使いの場合は、以下のトピックで新しい JDBC 2 ドライバーでの変更点について説明します。

JDBC API の改善点

- スクロール可能 ― 相対位置および絶対位置の両方を設定する機能
- 変更可能 ― さらに SQL を実行することなく Insert、Update、および Delete を行える機能
- 動的および静的なサーバー側のカーソル ― 自分自身またはほかのユーザーが行った変更を見たいかどうかを選択する機能
- バッチ アップデート ― たくさんのオペレーションを待ち行列に入れ、それらを一度に実行する機能
- 文字エンコードのサポート ― 接続ごとにエンコードを変更できる機能

JDBC Optional Package のサポート

- DataSource インターフェイス ― データ ソース オブジェクトを JNDI に登録可能
- 接続プーリングのサポート ― 接続プーリング インターフェイスの完全な実装

以前のバージョンとの互換性

Pervasive JDBC バージョン 2 は以前のバージョンと互換性があります。以前のリリースの Pervasive JDBC ドライバーでコンパイルされたアプリケーションは、再コンパイルすることなく、新しいドライバーで動作します。アプレットは、HTML ファイル内の jar ファイル名を pervasiveJDBC.jar から pvjdbc2.jar に変更する必要があります。

新しいドライバーは異なるパッケージ名を介してアクセスされます。

- com.pervasive.jdbc.v2.Driver は JDBC 2 ドライバーを読み込みます。
- pervasive.jdbc.PervasiveDriver は JDBC 1 ドライバーを読み込みます。

クラス名

Pervasive JDBC バージョン 2 ドライバーでは、クラス名は Sun が推奨する基準に適合するように変更されました。Pervasive のすべてのクラスは com.pervasive.jdbc.v2　で始まります。古いバージョンでは、クラスは pervasive.jdbc で始まっていました。

Pervasive JDBC ドライバーの制限

サポートされない API

Pervasive の JDBC ドライバーは次の JDBC インターフェイスをサポートしません。

- Array
- Blob
- Clob
- Ref
- Struct
- SQLData
- SQLInput
- SQLOutput

これらがサポートされないのは、Pervasive PSQL 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。

Pervasive JDBC 2ドライバーを使用したプログラミング

2

Pervasive PSQL の JDBC 2 機能の概要

この章では、以下の項目について説明します。

環境設定の方法

このセクションでは、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 の設定」
- 「システム PATH の設定」
- 「Pervasive 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」

CLASSPATH の設定

Java アプリケーションおよびアプレットが Pervasive PSQL 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 では、このファイルはデフォルトで /usr/local/psql/bin にインストールされます。

Windows の場合：

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

Linux 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
```

システム PATH の設定

共有メモリまたは IPX 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、通常は、DLL は必要とされません。

Pervasive 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 Pervasive 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 Pervasive PSQL 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ことがあります。

1. 1 台のマシンが Pervasive PSQL 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。「システム PATH 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、Pervasive PSQL 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。Pervasive JDBC ドライバーの URL の構文は次のとおりです。

```
jdbc:pervasive://< マシン名 >:< ポート番号 >/< データ　ソース >
```

＜マシン名＞	Pervasive データベース 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。
＜ポート番号＞	Pervasive データベース サーバー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。
＜データソース＞	アプリケーションが使用する予定の Pervasive データベース サーバー上の ODBC DSN の名前。

たとえば、Pervasive PSQL 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
```

したがって、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 Pervasive PSQL 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、<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>
```

メモ:アプレットのホストとなる Web サーバーで Pervasive PSQL エンジンが実行されている必要があります。

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。Pervasive 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、Pervasive PSQL 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、Pervasive PSQL 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 Pervasive PSQL 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。

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、Pervasive PSQL v11 SP3 エンジンは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 Pervasive PSQL データベースに接続する方法を次に示します。

ドライバー クラスパス

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

ドライバー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 2 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 インターフェイスのデフォルトのポートは 1583 です。ポートが指定されない場合、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
encrypt	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。『Advanced Operations Guide』の「ワイヤ暗号化」を参照してください。 値：always（常時）、never（しない） このオプションを指定しなかった場合、ドライバー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。 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。 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 jpscs.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。
encryption	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。 値：low（低）、medium（中）、high（高） デフォルト：medium（中） 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。 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。 `jdbc:pervasive://host/demodata?encoding=UTF-8&encrypt=always&encryption=low`

JDBC 接続文字列の例

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

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

// Pervasive JDBC 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。文字データはデータベース エンジンとやり取りするためにコード ページに変換する必要があります。文字データは、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。

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://MYSERVR: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

1 つのデータベースに 2 つの異なるエンコードを使用したテーブルがある場合、2 つの別個の接続を確立する必要があります。データベース エンジンは 1 接続につき 1 つのエンコードしか識別しません。このため接続ごとに別個のエンコードを指定する必要があります。

Pervasive JDBC ドライバーは、コード ページのための 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 ページのリストは Sun の Web サイト java.sun.com 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、Pervasive 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 pervasive jdbc パッケージを設定する必要があります。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 Pervasive PSQL エンジンは、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

Pervasive 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には、JSP を使用することができます。

次に示すのは、Pervasive PSQL に含まれるサンプル データベースの 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>Pervasive PSQL JSP Sample</title>
 </head>
 <body>

<h1>Pervasive PSQL JSP Sample</h1>
<h2>Course table in DEMODATA database</h2>
<p>
この例は、Pervasive PSQL データベースにある 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 の詳細に関しては、Sun の Web サイト java.sun.com 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、Sun 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ー情報を登録します。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。

Pervasive JDBC ドライバーは JDBC 2.0 Standard Extension API をサポートしました。現在、Pervasive 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% ピュアな 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 Pervasive は RowSet インターフェイスの実装を提供していませんが、Pervasive JDBC ドライバーは Sun の RowSet インターフェイス の実装で検証済みです。

DataSource

Sun はアプリケーション開発者がドライバー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ー固有の要素をなくすことができます。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、Sun 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 Pervasive 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る)。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リクエスト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、読み込み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()` メソッドを呼び出して、スクロール可能な結果セットのすべて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込み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。Pervasive PSQL の場合、インセンシティブな結果セットはこれに加えられた変更を一切反映しません（データの静的なスナップショットです)。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更新可能な 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。

この章では、以下の項目について説明します。

- 「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、Sun Microsystem の Web サイトに記載されている標準 API です。java.sun.com で 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、「Pervasive JDBC ドライバーの制限」に記述されている Pervasive 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。

Pervasive の 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。

- 「Pervasive JDBC ドライバーの概要」
- 「Pervasive JDBC 2 ドライバーを使用したプログラミング」

JDBC サンプル

Pervasive PSQL SDK では Web ダウンロードにより、JDBC のサンプルが入手可能です。デフォルトの場所にインストールする場合、この場所は *file_path*¥PSQL¥SDK¥JDBC¥SAMPLES です（*file_path* はデフォルトで¥Program Files¥Pervasive Software です）。

Pervasive PSQL ファイルのデフォルトの保存場所については、『Getting Started with Pervasive PSQL』の 「Pervasive PSQL 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。

索引