Zen v14

Java Class Library Guide

Developing Applications Using the Zen Java Class Library

Java Class Library Guide
2020 年 4 月

目次

このドキュメントについて

このドキュメントでは、Java クラス ライブラリを使用して Zen アプリケーション開発を行う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、Zen に精通し、Java クラス ライブラリを使用してアプリケーションを開発したいユーザー向けにデザインされています。

本ソフトウェアの全般的な使用手順やほかのデータベース アクセス方法については説明していません。

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。

大文字と小文字の区別	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。

Zen Java インターフェイスの概要

1

Java アプリケーションのインターフェイス

Zen Java インターフェイスは、標準の Btrieve API に追加されるオブジェクト指向のインターフェイスです。

以下のトピックでは、このインターフェイスと基本的な使用法をご紹介します。

- 「Zen Java インターフェイス」
- 「環境設定の方法」
- 「PVideo Java サンプル アプリケーションの実行」
- 「PVideo アプリケーションのソースの表示」
- 「開発者用リソース」

Zen における Java インターフェイス用開発の詳細については、「Java クラス ライブラリを使ったプログラミング」を参照してください。

Zen Java インターフェイス

Zen Java クラスは MicroKernel エンジンへのインターフェイスです。

Java クラス ライブラリと MicroKernel エンジン

Zen リレーショナル エンジンでは、列レベルでのデータベースへのアクセスが可能です。一方、MicroKernel エンジンでは、ファイル、レコードおよびインデックスによるアクセスのみが可能で、アプリケーション プログラム自体では Btrieve API 呼び出しにより返されたデータ バッファー内でフィールド レベルのアクセスを行う必要があります。

SQL データベースでは、データ辞書ファイルに格納されているデータ辞書で、各列レベルの情報にアクセスできます。

メモ Btrieve は、「ファイル、レコード、フィールド」の用語を使用しています。ただし、このセクションでは、Zen 辞書を持つデータベースと併用できるクラスについて説明する場合は、代わりに「テーブル、行、列」という用語を使用します。

この API の目的は、単に Java や C++ のようなオブジェクト指向言語へのインターフェイスを提供するということだけでなく、オブジェクト指向アプリケーションに適する論理構造を提供することにあります。

オブジェクト指向 API の設計では、次の目標に取り組んでいます。

- Btrieve アプリケーション開発者に一連の抽象化を提供する。
- 使いやすさを保証する。
- プラットフォームに依存するバイトオーダーを隠す。
- 開発者が Btrieve システムのすべての機能を使用できるようにする。
- MicroKernel エンジンのパフォーマンスを損なわないようにする。

Java の登場後まもなく、データベース インターフェイスとして Java Database Connectivity（JDBC）が導入されました。JDBC は、データベース開発者の間にますます普及しつつあります。新しい Zen Java API の設計では、モデルとして JDBC API を使用し、JDBC で使用されているアイデアと手法の多くを利用しています。ただし、JDBC は、SQL と一般的にリレーショナル モデルをサポートするよう設計されているため、いくつかの新しい概念を導入する必要がありました。その一方で、この API セットはトランザクショナル Btrieve もサポートします。

既に述べたように、Btrieve アプリケーションが SQL データベースに属するデータ ファイルにアクセスする場合、新しい API 呼び出しはデータベースの辞書に格納されている列の記述を使用できます。SQL データベースの一部ではない Btrieve データ ファイルの場合、新しい API はこれらのファイルにアクセスするための別の手段を提供します。

データベース コンセプト

SQL データベースに属さない一連の Btrieve データ ファイルは、これまでどおり、アプリケーション プログラムが論理的な意味でそれらのファイルを 1 つに結合する場合にデータベースを構成します。そのようなデータベースを「疎結合型」データベースと呼びます。疎結合型データベースには、データベース辞書がありません。それに対して、SQL データベースを「密結合型」データベースと呼びます。密結合型データベースには、永続的なデータベース辞書があります。

Zen Java インターフェイスは、使用するクラスに応じて、高レベルでも低レベルでも動作可能です。

密結合型データベース

Java インターフェイスの高レベルの部分は、Btrieve のプログラマが以前扱っていたポジション ブロックおよびデータ バッファーなど、実装の詳細部分を隠します。

疎結合型データベース

疎結合型データベースを現在持っており、新しい API の列レベルのサポートを活用したいと考えているユーザーは、以下のオプションのうちの 1 つを選択する必要があります。

1 Zen Control Center を使用して、疎結合型データベースのデータベース辞書を作成します。実際には、新しい API を使用する前にデータベースが密結合型データベースに切り替えられます。

2 新しい永続的な辞書を作成し、新しい API を使用してアプリケーション プログラム内で Btrieve ファイルごとにテーブル、列などを定義します。この場合、データベースはプログラムによって密結合型データベースに切り替えられます。

環境設定の方法

このセクションでは、Zen Java 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。

- 「JDK のサポート」
- 「CLASSPATH 環境変数」

JDK のサポート

Zen Java インターフェイスは、JDK 1.4 以降のバージョンをサポートしています。

CLASSPATH 環境変数

Zen SDK では、Zen クラスを指すように CLASSPATH 変数を設定する必要があります。

サンプル CLASSPATH は、Zen が *file_path*¥Zen にインストールされており、Java Class Library SDK が *file_path*¥Zen¥SDK¥JCL にインストールされていることを前提とします。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。

CLASSPATH 環境変数を以下のように変更する必要があります。

```
SET CLASSPATH=.; file_path¥Zen¥bin¥psql.jar;
file_path¥Zen¥SDK¥JCL¥Samples¥PVideo¥pvideoj.jar;
file_path¥Zen¥SDK¥JCL¥Samples¥PVideo;
```

file_path は Zen のインストール場所です。

Windows CLASSPATH

ある Zen クラス ファイルが見つからないというエラーが発生した場合、コントロール パネルで設定されているユーザー CLASSPATH 変数が、Zen パスが設定されているシステム CLASSPATH 変数を上書きしている可能性があります。

以下の手順で Windows の設定を確認し、必要に応じて修正してください。

1 ［スタート］メニューから［コントロール パネル］にアクセスします。

2 ［システム］をダブルクリックします。

3 ［詳細］タブまたは［詳細設定］タブを選択し、［環境変数］をクリックします。

4 Zen パスは、システム変数に含まれています。ユーザー環境変数で、CLASSPATH が存在するかどうかを確認します。存在しない場合、何もする必要はありません。存在する場合、ユーザー変数の先頭に %CLASSPATH%; を付けます。

PVideo Java サンプル アプリケーションの実行

PVideo サンプル アプリケーションは、Zen Java インターフェイスの機能を利用して作成されています。

▶▶ JCL SDK サンプル アプリケーションを実行するには

1 Zen SDK Java Pvideo フォルダーを開きます。デフォルトの場所に保存されていると仮定した場合、この場所は *file_path*¥SDK¥jcl¥samples¥Pvideo です。

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。

2 PVideoJ.bat ファイルをダブルクリックします。

次のアプリケーション ウィンドウが表示されます。

図 1 Zen PVideo サンプル アプリケーションのメイン ウィンドウ - Java

PVideo アプリケーションのソースの表示

Zen ビデオ店サンプル アプリケーションの Java ソース ファイルは、.jar ファイルに格納されています。これらのファイルを抽出して、Btrieve テーブルに接続する Zen Java インターフェイスの実例を得ることができます。

以下の手順は、JDK がインストールされており、その ¥bin ディレクトリがパスに設定されていることを前提とします。

▶▶ Java ソース ファイルを解凍するには

1 コマンド プロンプトで、以下のように入力します。

```
cd ¥zen¥sdk¥jcl¥samples¥pvideo
```

2 次に、JAR コマンドを入力します。

```
jar -xvf source.zip
```

ソース ファイルが、現在のディレクトリと zen¥sdk¥jcl¥samples¥pvideo のサブディレクトリに展開されます。

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。

開発者用リソース

Java インターフェイスを使用してアプリケーションを開発する際の情報は、以下の Web サイトで得ることができます。

オンライン リソース

リソース	オンラインの場所
Actian Web サイトの開発者用コンテンツ	www.actian.com
Oracle Java Web サイト	http://www.oracle.com/technetwork/java/
Oracle Java チュートリアル	http://docs.oracle.com/javase/tutorial/

Javadocs 形式のリファレンス

Java クラス ライブラリに付属の Javadoc 形式のドキュメントでは、JCL ベースのアプリケーションで使用されるクラスとメソッドのリファレンスを提供します。

Java クラス ライブラリを使ったプログラミング

2

以下のトピックでは、Zen における Java プログラミングの情報や手順を提供します。

- 「Zen Java インターフェイスのクラス階層」
- 「主要なクラスとメソッドの概要」
- 「Java アプリケーションにおける一連の手順」
- 「Java クラス ライブラリ使用時の最初の手順」
- 「Java データ型の問題点」
- 「サンプル データベース ファイル」
- 「追加の Java サンプル」

Zen Java インターフェイスのクラス階層

ここでは、Zen Java インターフェイスのクラスについて説明します。

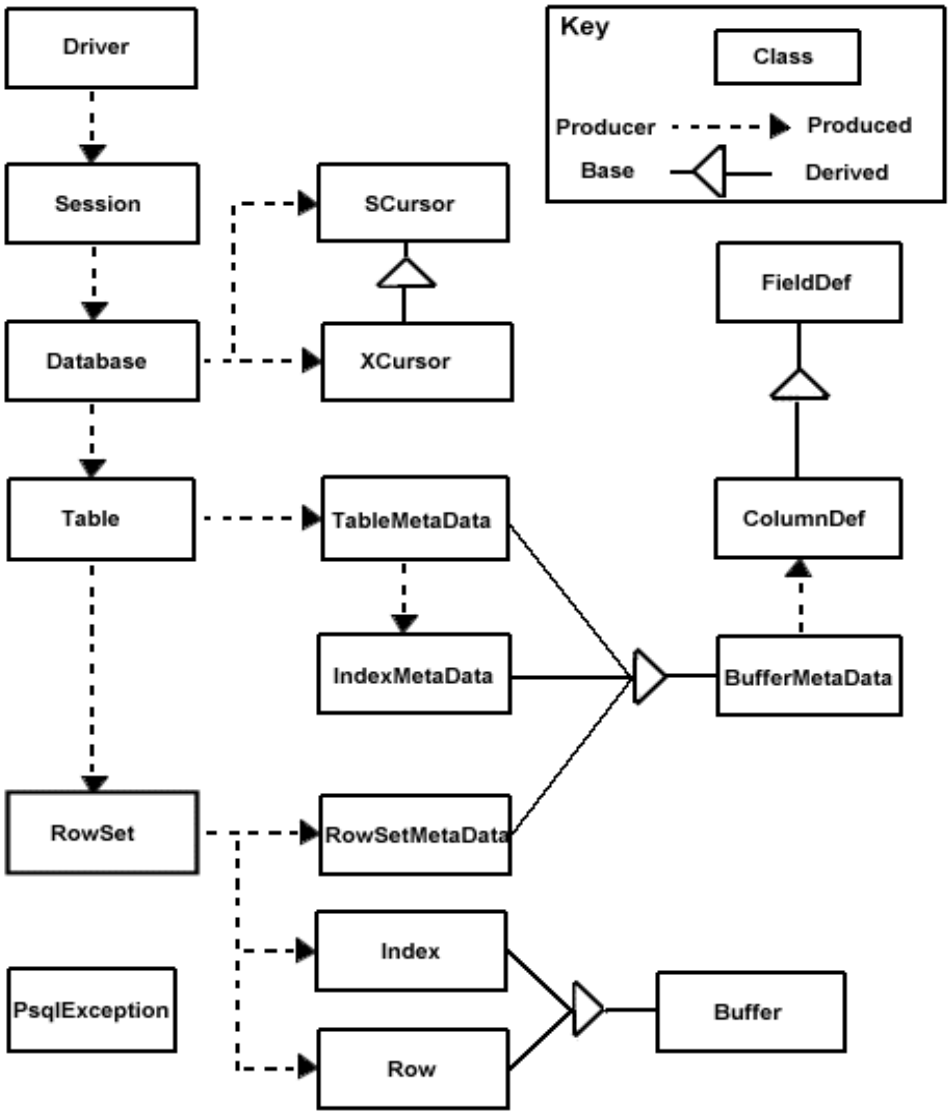
「ファクトリ」パターンを適用することにより、Driver、Timestamp、FieldDef および PsqlException を除く、ほとんどすべてのクラスは（単一または複数の）ほかのクラスのメソッドを通してのみ作成できます。クラスの説明で、パブリック コンストラクターが使用できない場合、ラベル Constructors の下にコンストラクターのリストは表示されません。

Zen Java インターフェイスには、API のメソッドを呼び出すときに発生するエラー状態を処理する特別なクラス PsqlException があります。PsqlException は、ava.lang.Exception を拡張します。Zen Java クラスの個々のメソッドは PsqlException を返しますが、簡潔にするために以下の節はメソッド定義から省略されています。

```
throws PsqlException
```

図 2 に、すべてのクラスとクラス間の関係を示します。

図 2 Java API のクラス階層



Zen Java インターフェイスのクラスは、以下の 3 つのグループに分類できます。

- 「一般クラス」
- 「Rowset ファミリ クラス」
- 「Cursor ファミリ クラス」

一般クラス

以下のクラスは、一般クラスのグループに属します。

- Driver
- Session
- Database
- Timestamp
- PsqlException

これらのクラスは、密結合型データベースと疎結合型データベースの両方に使用できます。

Rowset ファミリ クラス

以下のクラスは、Rowset ファミリ クラス グループに属します。

- Table
- Rowset
- Buffer
- Row
- Index
- DBMetaData
- TableMetaData
- RowSetMetaData
- ColumnDef
- IndexMetaData

これらのクラスは Zen データベースの辞書に見られるメタ データを使用するので、密結合型データベースにのみ適用できます。

Cursor ファミリ クラス

これは、Zen Java インターフェイスの低レベル部分です。このインターフェイス部分を使用する場合、ポジション ブロック、データ バッファーやその他 Btrieve API 特有のプログラミング要素を処理する必要があります。

メモ Btrieve ファイルに DDF がない場合は、Java インターフェイスのこれらのクラスを使用しなければなりません。データ用の DDF ファイルは、Zen Control Center ユーティリティを使用して作成できます。詳細については、『*Zen User's Guide*』を参照してください。

以下のクラスは、Cursor ファミリ クラス グループに属します。

- SCursor
- XCursor
- FieldDef

これらのクラスはメタ データを使用せず、従来の Btrieve オペレーション（BTRV メッセージなど）を直接サポートします。したがって、これらのクラスは一般に疎結合型データベースに適用されます。ただし、これらのクラ

スを使用して、DDF とデータとの密結合型データベースのテーブルを格納するデータ ファイルにアクセスすることもできます。（しかし、後者では「Rowset ファミリ」を使用する方が便利です。）

主要なクラスとメソッドの概要

以下の表に、基本的な Java アプリケーションにおける主なクラスとメソッドの一覧を示します。

主なクラスとメソッド	使用目的
Driver establishSession killAllSessions	• 1 つまたは複数のセッションの確立。 • すべてのセッションの停止。
Session connectToDatabase startTransaction commitTransaction abortTransaction	• データベースへの接続。 • トランザクションの処理。
Database getTable getTableNames createxCursor	• テーブル名とテーブルの取得。 • xCursor を作成し、必要に応じて低レベルのオペレーションを実行。
Table getTableMetaData createRowSet	• テーブル データへのビューである RowSet の作成。 • テーブルに関するメタ データ情報の取得。
TableMetaData getIndexNames	• テーブルからのインデックスの取得。
RowSet createRow insertRow deleteRow updateRow getNext getByIndex reset	• テーブルの一部の取得。 • 行情報の更新。
RowSetMetaData AddColumns DropColumns AddFirstTerm SetAccessPath SetDirection	• 列の追加または削除。 • WHERE 句の機能の実行。 • ナビゲーションの方向の変更。
Row getString setString getFloat setFloat	• RowSet により取得したデータを実際に更新。

ほかのクラスやメソッドも Zen Java Class Library javadoc で読み取れますが、アプリケーションで最も頻繁に使用するクラスは上記のものです。

Java インターフェイス クラスの構造はフラットで、継承関係はありません。

DRIVER

Driver はセッションを確立します。Driver を使用して、必要な個数の Session オブジェクトのインスタンスを作成できます。KillallSessions メソッドはソケット接続、データベース接続、開いているファイル、システム リソースをクリーンアップするので、このメソッドを使用してアプリケーションを停止することもできます。

SESSION

Sessions はドライバーとの接続を使用して、URI によるデータベースへの接続およびトランザクションの開始と終了を行うことができます。セッションごとに一意の Btrieve クライアント ID を取得します。アプリケーションが、特に複数の Btrieve クライアント ID を必要としない限り、通常はアプリケーション全体に対して 1 つの Session を使用することができます。セッションおよび Btrieve クライアントは、Session.close() を呼び出すことによってリセットできます。クライアント ID の詳細については、『*Zen Programmer's Guide*』の「Btrieve クライアント」および『*Zen User's Guide*』の「ライセンス モデル」を参照してください。

DATABASE

このクラスでインスタンスが作成されるオブジェクトは、データベース自体です。このクラスのメソッドを使用すれば、開発者は密結合型データベースの場合にテーブル名、テーブルなどを取得したりすることができます。疎結合型データベースの場合、開発者は Extended オペレーションなどの Btrieve に似た使い慣れた API でアクセスできます。

TABLE

このクラスのインスタンスは、テーブルをオブジェクトとして表します。Table オブジェクトを使用して、行セットを作成し、テーブルに関するメタデータ情報を取得できます。

ROWSET

RowSet は、非常に重要なクラスです。このクラスは、関連するテーブルの行にアクセスする場合に使用します。また、このクラスのメソッドから行を挿入、更新および削除できます。

このクラスのインスタンスは、ベース テーブルから得られる一連の行を表します。ベース テーブルを見るもう 1 つの方法である RowSet は、物理的に格納されたテーブルから作成された仮想テーブル（ビュー）です。

メモ　このオブジェクトを作成できるのは、Table の createRowSet メソッドを使用した場合のみです。

行セットには、ベース テーブルのすべての行または行のサブセットを含めることができます。addFirstTerm、addAndTerm などの RowSetMetaData 内の多数のメソッドは、ベース テーブルからのどの行を行セットに組み込むかを決定する制限基準の条件を指定する場合に使用します。

行セットには、ベース行のすべての列を含めることも、ベース行から任意の個数の列を選択して含めることもできます。RowSet オブジェクトが Table.createRowSet で作成された場合、このオブジェクトは呼び出しで明示的または暗黙に指定された noColumns の値に従って初期化されます。noColumns に *True* を指定すると、行セットに列が含まれません。この場合、アプリケーションでは、RowSetMetaData.addColumns を呼び出すことによって、選択した列を行セットに含めることができます。noColumns に False を指定したり、引数として含めない場合、作成された RowSet にはテーブルのすべての列が含まれます。

通常、アプリケーションは行セットを開いたり閉じたりする必要はありません。関連するデータ ファイルは RowSet オブジェクトの作成時に開かれ、また、アプリケーションが RowSet.close を明示的に呼び出すか、あるいは、さらに高レベルの関連オブジェクト、たとえば、Table、Database などを閉じた結果としてデータ ファイルが閉じられます。

行セットの要素は、Row オブジェクトを通じてアクセスできます。行を取得する複数のオプションがあります。

行セットに関する追加情報については、Zen Java Class Library javadoc を参照してください。

ROW

このクラスのインスタンスは、RowSet インスタンス内の行の行バッファーを表します。Row クラスによって、行、および行内の列の値にアクセスできます。RowSet を使用して、行の取得、更新または削除を行ったり、挿入のための行を作成することができます。

このクラスには独自のメソッドがありませんが、Buffer から継承されたメソッドを使用して行バッファー内で列の値を取得または設定することができます。

Java アプリケーションにおける一連の手順

以下の一連の手順は、アプリケーションまたはアプレットで通常使用されます。最初の 2 つの手順は、密結合型データベースでも、疎結合型データベースでも同じです。

1 実行環境として使用される Session のインスタンスを設定します。Driver は静的クラスなので、インスタンス化する必要はありません。また、アプリケーションまたはアプレットで複数のセッションを作成することもできます。

2 URI を指定する Session.connectToDatabase を呼び出して、データベースの Database オブジェクトを取得します。データベース URI の詳細については、『*Zen Programmer's Guide*』を参照してください。また、アクセスする個々のデータベースごとに、1 つのセッション内で複数の Database オブジェクトを取得することもできます。

手順 3 は、「ケース 1 : アプリケーションが密結合型データベースのテーブルにアクセスする」（Btrieve データベースは DDF を使用）または「ケース 2 : 疎結合型データベースのデータ ファイル、または密結合型データベースのテーブルを標準的な方法で格納するデータ ファイルにアプリケーションがアクセスする」を選択してください。

ケース 1 : アプリケーションが密結合型データベースのテーブルにアクセスする

3 オプションとして、アプリケーションがデータベースのテーブル名を認識していない場合は、`Database.getTableNames` を呼び出してテーブル名を取得します。

4 `Database.getTable` を呼び出して、作業するテーブルごとに Table オブジェクトを取得します。

5 オプションとして、アプリケーションが辞書から列の名前またはインデックスの名前を取得する必要がある場合は、テーブルごとに TableMetaData オブジェクトを取得します。

6 アプリケーションで処理するテーブルごとに、少なくとも 1 つの RowSet オブジェクトを作成します。RowSet オブジェクトは、データの取得と変更に使用します。テーブルのすべての列を含むか、まったく含まない行セットを作成できます。手順 7 の詳細を参照してください。

テーブル内をナビゲートする際に、複数のカレンシー（位置）を保持しながら同時にテーブルを処理したい場合、アプリケーションでは、1 つのテーブルに対して複数の RowSet を作成します。所定のテーブルの RowSet オブジェクトは、アプリケーションの実行中いつでも使用できます。つまり、異なるインデックス、異なる条件などによる取得に使用できます。

7 所定の行セットに対して

a. 行セットの RowSetMetaData オブジェクトを取得します。

b. 行セットが列を含めずに作成された場合は、必要な列を定義に追加します。

c. 必要であれば、行のデフォルトの特性、たとえば、方向、アクセス パスなどを変更します。

d. ベース テーブル内の選択した行のみを 1 つの行セットに含める場合は、選択条件、たとえば、addFirstTerm、addAndTerm などを定義します。

e. Row オブジェクトを取得するための RowSet.getNext メソッドを呼び出します。オプションとして、RowSet のほかのメソッド、たとえば、getByIndex、getbyPercent、insertRow、updateRow などを呼び出します。

f. 行で列の値を取得または設定するには、Row のメソッド、たとえば、getString、setString、getInt などを呼び出します。

メモ 手順 a ～ d はオプションであり、また、アプリケーションにもよりますが、行セットによっては必要でない場合があります。

ケース 2 : 疎結合型データベースのデータ ファイル、または密結合型データベースのテーブルを標準的な方法で格納するデータ ファイルにアプリケーションがアクセスする

3. アプリケーションが動作する個々のデータ ファイルに対して、SCursor オブジェクトまたは XCursor オブジェクトを作成します。XCursor は、このファイルに対して Extended get オペレーションまたは Extended step オペレーションが実行される場合のみ必要です。オプションとして、複数の位置を同時に保持しながら、並行してファイルを処理したい場合、アプリケーションで同一ファイルに対して複数の Cursor オブジェクトを取得することができます。

4 SCursor.BTRV を使用して、従来の Btrieve オペレーション、たとえば、open、get、step、insert などを実行します。

メモ XCursor は SCursor の拡張なので、XCursor 上で BTRV メソッドを使用できます。

5 オプションとして、データ フィールドにいくつかの FieldDef オブジェクトを定義したり、これらのフィールド オブジェクトを使用して選択基準、たとえば、XCursor.addFirstTerm などを設定して抽象リストを定義したり、getNextExtended のようなメソッドを使用して オペレーションを実行します。

6 SCursor（XCursor）オブジェクトのプライベート データおよびキー バッファーでフィールドの値を取得または設定するには、getDString、setKString、getDInt4 などのメソッドを呼び出します。

以降のトピックでは、これらの概要手順についてさらに詳しく説明します。

Java クラス ライブラリ使用時の最初の手順

Java Btrieve アプリケーションを作成するには、まず以下の手順に従います。

1 「環境が正しいかどうかの確認」

2 「データベースとテーブルの作成」

3 「データベースへの接続」

4 「テーブルの取得と行セットの作成」

5 「行セット内での行間の移動」

6 「データの制限またはフィルター処理」

7 「行の挿入、更新または削除」

8 「複数のオペレーションによるトランザクションへの結合」

9 「バイナリ ラージ オブジェクト（BLOB）の格納と取得」

環境が正しいかどうかの確認

「環境設定の方法」で説明しているように、環境が正しくセットアップされていることを確認してください。

また、psql.jar が CLASSPATH 環境変数内にあるかどうかを確認します。この手順は、インストール手順で既に実行しているはずです。Zen Java インターフェイス クラス ライブラリ内のクラスにアクセスするには、Java ソース ファイルにパッケージをインポートする必要があります。

```
import PSQL.database.*;
```

データベースとテーブルの作成

現在、Zen Java インターフェイスのクラス ライブラリは、データベースとデータベース テーブルの作成をサポートしていません。これらの作業を実行するには Zen Control Center を使用します。データベースのデータ辞書ファイル（DDF）およびデータ ファイルを作成したら、Java API クラスを使用してテーブルを追加してアクセスすることができます。

Zen Control Center とテーブルの作成の詳細については、『*Zen User's Guide*』を参照してください。

データベースへの接続

Zen データベースに接続するには、エンジンに直接接続します。

URI を使用してデータベースに接続するには

1 Driver から Session オブジェクトを取得します。

```
Session session = Driver.establishSession();
```

2 Session オブジェクトを使用してデータベースに接続します。URI を指定してデータベースに接続します。少なくとも、この URI には `btrv:///dbname` を含めてください。

```
Database db = session.connectToDatabase("btrv:///demodata");
```

または

```
Database db = session.connectToDatabase("btrv://user@host/
demodata?pwd=password");
```

データベース セキュリティおよび URI の詳細については、『*Advanced Operations Guide*』を参照してください。

テーブルの取得と行セットの作成

⏩ データベースからのテーブルの取得

```
Table table = db.getTable("MyTable");
```

ここで、「MyTable」はデータベース テーブルの名前です。テーブル名では大文字と小文字を区別します。

また、データベースのテーブル名のリストを取得することもできます。

```
String [] names = db.getTableNames();
```

⏩ Table オブジェクトのプロパティにアクセスするには

Table オブジェクトがあれば、TableMetaData オブジェクトからの列とインデックスに関する情報が得られます。

```
TableMetaData tmd = table.getTableMetaData();
```

いくつかの例を以下に示します。

```
// 列数を取得する
int num_columns = tmd.getColumnCount();
// 列 0 のデータ型を取得する
int data_type = tmd.getColumnDef(0).getType());
// 列 0 の列の長さを文字数で取得する
int col_length = tmd.getColumnDef(0).getOffset();
```

⏩ Table オブジェクトの行にアクセスするには

テーブルの行にアクセスするには、Table オブジェクトでしか作成できない RowSet オブジェクトが必要です。

```
RowSet rowset = table.createRowSet();
```

同じテーブルから複数の RowSet オブジェクトを作成できます。RowSet オブジェクトは、テーブルの行を取得したり、新しい行を挿入したり、既存の行を削除および更新する場合に使用します。すべてのテーブルの行を繰り返し取得するには、RowSet オブジェクトを作成し、PsqlEOFException が発生するまで getNext を呼び出します。

```
try {
  while(true)
    Row row = rowset.getNext();
}
catch(PsqlEOFException ex) {
// もう行は残っていない
}
```

⏩ Rowset オブジェクトのプロパティにアクセスするには

RowSet オブジェクトがあれば、RowSetMetaData から行セットに関する情報を得ることができます。

```
RowSetMetaData rsmd = rowset.getRowSetMetaData();
```

詳細については、Zen Java Class Library javadoc の RowSetMetaData クラスのメソッドを参照してください。

行セット内での行間の移動

前に示したように、getNext メソッドを繰り返し呼び出し、行セットのすべての行を取得することができます。行セットの RowSetMetaData オブジェクトの direction プロパティを変更すれば、後方へ繰り返すことができます。

```
rsmd.setDirection(Const.BTR_BACKWARDS);
```

このプロパティを設定した後、getNext メソッドの動作は getPrevious になります。

メモ　以下の手順に示すように、行セットのカレンシーは、いつでもリセットして、最初の行の前または最後の行の後になるようにすることができます。これは、行セット全体を繰り返さなくても、最初の行または最後の行を取得できる効率的な方法です。

▶▶ 行セット内の最初の行を取得するには

1　方向を前方に設定します。これはデフォルトです。

```
rsmd.setDirection(Const.BTR_FORWARDS);
```

2　最初の行の前に行セットのカレンシーを設定します。

```
rowset.reset();
```

3　最初の行を検索します。

```
Row first = rowset.getNext();
```

▶▶ 行セット内の最後の行を取得するには

1　方向を後方に設定します。

```
rsmd.setDirection(Const.BTR_BACKWARDS);
```

2　最後の行の後に行セットのカレンシーを設定します。

```
rowset.reset();
```

3　最後の行を検索します。

```
Row last = rowset.getNext();
```

▶▶ Row オブジェクトの列データにアクセスするには

Row オブジェクトは、Buffer クラスから多数の accessor メソッドまたは mutator メソッドを継承します。これらのメソッドで、Row オブジェクトのバッファーに対して列データを設定または取得できます。

1　たとえば、String として列 0 でデータを取得するには、以下のうちの 1 つを実行します。

```
String str = row.getString(0);
```

または

```
String str = row.getString("ColumnName");
```

ここで ColumnName は列 0 に対してデータ辞書で定義されている名前です。列名は大文字と小文字を区別します。

2　次に、列 0 に対してデータを設定するには、setString メソッドのうちの 1 つを使用します。

```
row.setString(0, "MyColumnData");
```

または

```
row.setString("ColumnName", "MyColumnData");
```

3　詳細については、Zen Java Class Library javadoc の Buffer クラスのメソッドを参照してください。

データの制限またはフィルター処理

▶▶ 行セット内の行を制限またはフィルター処理するには

行セット内の行を制限またはフィルター処理するには、RowSetMetaData の addFirstTerm、addOrTerm および addAndTerm を使用する必要があります。たとえば、テーブルの最初の列に整数データが入っていて、その最初の列の値が 23 より大きい行をすべて取得する場合は、以下の手順に従います。

1　行セットの RowSetMetaData を取得します。

```
RowSetMetaData rsmd = rowset.getRowSetMetaData();
```

2 最初の列、つまり列番号 0 に対して ColumnDef を取得します。

```
ColumnDef cmd = rsmd.getColumnDef(0);
```

3 行セットのカレンシーを先頭位置にリセットします。

```
rowset.reset();
```

4 最初の条件を追加します。

```
rowset.addFirstTerm(cmd, Const.BTR_GR, "23");
```

5 getNext を呼び出して、列 0 が 23 よりも大きい最初の行を取得します。

```
Row row = rowset.getNext();
```

RowSetMetaData の addOrTerm および addAndTerm を使用して項を追加することができます。これらのメソッドを使用して、SQL の WHERE 句に似た、さらに複雑なフィルター処理条件を設定できます。

▶▶ 行セットから列を選択するには

RowSetMetaData のメソッド addColumns と dropColumns を使用すれば、SQL の select ステートメントのような取得する列のサブセットを指定できます。デフォルトでは、noColumns パラメーターに True を設定して行セットが作成されない限り、行セットはすべての列データを取得します。

```
RowSet rowset = table.createRowSet(true);
```

この場合、列は取得されません。その代わり、行セットを作成後、行の一部または全部を削除できます。

```
RowSetMetaData rsmd =rowset.getRowSetMetaData();
rsmd.dropAllColumns();
```

次に、対象の列のセットを行セットに追加できます。

```
rsmd.addColumns("LastName", "FirstName");
```

行セットに列を追加するには、列名または列番号を使用できます。詳細については、Zen Java Class Library javadoc の RowSetMetaData クラスの addColumns メソッドと dropColumns メソッドの項を参照してください。

列番号で行内の列データにアクセスする場合、列番号は addColumns メソッドと dropColumns メソッドによって生成される新しい順序の影響を受けるので、注意してください。たとえば、FirstName は最初は行セットの列番号 3 であった場合、すべての列を削除した後に LastName と FirstName を追加すると、FirstName の列番号は 1 になります。

▶▶ インデックスで行を取得するには

定義されたインデックスを比較演算子と共に使用すれば、行を取得できます。たとえば、インデックス Last_Name が仮想テーブルの LastName 列で作成されている場合、以下のことを実行して Smith という LastName を持つ行を検索することができます。

1 行セットの RowSetMetaData オブジェクトを使用して、インデックス Last_Name を使用するようにアクセスパスを設定します。

```
rsmd.setAccessPath("Last_Name");
```

2 インデックス名を使用してインデックス オブジェクトを作成します。名前の代わりにインデックス番号を使用できます。

```
Index index = rowset.createIndex("Last_Name");
```

3 インデックスに対して LastName 列のデータを設定します。

```
index.setString("LastName", "Smith");
```

4 LastName = = Smith の最初の行を取得します。

```
try {
  Row row = rowset.getByIndex(Const.BTR_EQ, index);
}
```

5　発生する例外を把握します。

```
catch(PsqlOperationException ex) {

// エラー コード == 4 である場合、
// LastName== Smith を持つ行は存在しません。
// これは正常なオペレーションと見なすことができます。
// ステータス コードが 4 でない場合は、「見つからない」
// 以外の何らかの理由でオペレーションが失敗しています。

  if (ex.getErrorCode() != 4)
    throw(ex);
}
```

インデックスが重複キー値を許可する場合は、比較演算子に適合する最初の行が返されます。返された行は現在の行になるので、インデックスに基づいて、行セットの getNext メソッドで次の論理行を取得できます。

テーブルの TableMetaData オブジェクトからインデックス名のリストを取得できます。

```
String [] index_names = table.getTableMetaData().getIndexNames();
```

インデックスに名前を付けなくてもよいことに注意してください。その場合、代わりにインデックス番号を使用してください。インデックス番号はゼロから始まります。

行の挿入、更新または削除

▶▶ 新しい行を挿入するには

1　まず、新しい Row オブジェクトを作成します。

```
Row row = rowset.createRow();
```

2　行の列データを設定します。

```
// 列 0 のデータを設定する
row.setString(0, "Column0String");
// 列 1 のデータを設定する
row.setInt(1, 45);
// 列 2 のデータを更新する
row.setDouble(2, 99.99);
```

3　次に行を挿入します。

```
rowset.insertRow(row);
```

この新たに挿入された行は、行セットの現在の行になります。行セットのカレンシーを挿入により変更したくない場合は、オーバーロード版の insertRow を使用できます。これは、カレンシー変更なし（NCC）を示すブール引数をとります。

```
rowset.insertRow(row, true);
```

行セットの現在の行は変更されません。

▶▶ 行またはレコードを更新するには

1　まず、行セットの行取得メソッド、getNext、getByIndex などの 1 つを呼び出して、行を取得します。

```
Row row = rowset.getNext();
```

2　行の列データを変更します。

```
// 列 1 を更新する
row.setInt(1, 45);
```

3 次に行を更新します。

```
rowset.updateRow(row);
```

次に示すように NCC（No-currency-change : カレンシー変更なし）が示されていない場合は、insertRow のように、新しく更新された行が行セットの現在の行になります。

```
rowset.updateRow(row, true);
```

⏩ 行またはレコードを削除するには

1 まず、行セットの行取得メソッド、getNext、getByIndex などの 1 つを呼び出して、行を取得します。

```
Row row = rowset.getNext();
```

2 次に行を削除します。

```
rowset.deleteRow(row);
```

削除する行は、現在の行、つまり、最終取得オペレーションから返された行である必要はありません。削除する行が現在の行でない場合、deleteRow メソッドはその行を現在の行にした後削除します。削除後、getNext を呼び出すと、その削除された行の後の行が返されます。

複数のオペレーションによるトランザクションへの結合

トランザクションを使用すると、一連のオペレーションを 1 つのオペレーションに結合し、これをコミットまたは中止することができます。

⏩ 一連のオペレーションをトランザクションにするには

Session クラスのトランザクション メソッドを使用して、一連のオペレーションをトランザクションにすることができます。

次に、トランザクションの例を示します。

1 トランザクションを開始します。

```
try {
session.startTransaction(BTR_CONCURRENT_TRANS);
```

2 障害が発生するとロールバックする、いくつかのオペレーションを実行します。

```
// ここに 1 つまたは複数のオペレーションを追加する
```

3 トランザクションのコミットを試行します。

```
session.commitTransaction(); }
catch(PsqlException ex)
```

4 エラーが検出されたら、トランザクションを中止します。

```
{     // エラーが発生
session.abortTransaction(); }
```

トランザクションは「排他的」にすることも「並行」にすることもできます。トランザクションの詳細については、『*Zen Programmer's Guide*』を参照してください。

バイナリ ラージ オブジェクト（BLOB）の格納と取得

setObject および getObject メソッドは、java io Serializable インターフェイスを実装する Java オブジェクトの格納および取得に使用できます。setBinaryStream および getBinaryStream メソッドは、Java InputStreams を使用したバイナリ データの格納および取得に使用できます。次のインターフェイスを持つ「単純な」Employee オブジェクトがあるとします。

```
public class  Employee implements java.io.Serializable
public int    getID();              // Employee の ID を取得する
public void   setID(int ID);        // Employee の ID を設定する
public String getName();            // Employee の名前を取得する
public void   setName(String name); // Employee の名前を設定する
public String getManagerName();     // Manager の名前を取得する
public void   setManagerName();     // Manager の名前を設定する
```

また、C:¥Employees¥Java Duke¥report.txt というファイルが存在し、読み込めるとします。この場合、Employee オブジェクトのインスタンスを作成し、さまざまなメソッドを使用して設定を行い、データベースの Employee_Data 列に格納することができます。さらに、そのファイルを Manager_Report 列に格納することもできます。

```
// 通常のセットアップ（Driver.establishSession など）および
//employeeObject という名前の Employee オブジェクトの
// インスタンスの作成は完了済み
employeeObject.setName("Java Duke");
employeeObject.setID(123456789);
employeeObject.setManagerName("Big Boss");

FileInputStream managerReport = null;
try
{
   managerReport = new FileInputStream(C:¥Employees¥Java Duke¥report.txt);
}
catch(IOException ioe)
{
   // 例外処理
}

// その行の列値を設定する
// RowSet オブジェクト（行セット）が既に
// 作成されているものとする

Row employeeRow = rowset.createRow();

// データベースの ID 列を設定する
employeeRow.setInt("ID", employeeObject.getID());

// Employee オブジェクトを行に設定する
try
{
   employeeRow.setObject("Employee_Data", employeeObject);
}
catch(PsqlIOException pioe)
{
   // 例外処理
}

// マネージャーのレポートを行に設定する
employeeRow.setBinaryStream("Manager_Report", managerReport);

// 行を挿入する
rowset.insertRow(employeeRow);

// これで、データベースからこの行を取得することができる

RowSetMetaData rsmd = rowset.getRowSetMetaData();
ColumnDef cdef = rsmd.getColumnDef("ID");
```

```
rsmd.addFirstTerm(cdef, Const.BTR_EQ, "123456789");
Row rowRetrieved = rowset.getNext();

// 行の取得後、その行に getObject および getBinaryStream
// メソッドを実行して、目的のデータを取得することができる

try
{
   Employee employeeRetrieved =
      (Employee)rowRetrieved.getObject("Employee_Data");
}
catch(PsqlException pe) // このメソッドは PsqlIOException および
{                       // PsqlClassNotFoundException の両方をスローする
   // 例外処理
}

InputStream reportRetrieved =
   rowRetrieved.getBinaryStream("Manager_Report");

// これらのオブジェクトはこれで再構成された
// 通常の方法で、オブジェクト自体またはその親に対して
// 定義されたメソッドを呼び出すことができる

String managerName = employeeRetrieved.getManagerName();

// 通常、ファイル全体を 1 つのチャンクで処理することは
// あまりないが、リソースがあれば可能
byte file[] = new byte[reportRetrieved.available()];
reportRetrieved.read(file);
```

詳細については、「バイナリ ラージ オブジェクトのサポート」を参照してください。

Java データ型の問題点

このトピックでは、Zen Java プログラマに役立つと思われる情報を示します。

バイナリ ラージ オブジェクトのサポート

Java クラス ライブラリにバイナリ ラージ オブジェクト（BLOB）処理のサポートが追加されました。BLOB は大きな（最大 4 GB）バイナリ データを表し、Zen エンジンでは LONGVARBINARY データ型として示されます。Java クラス ライブラリは、次のメソッドでこれらのデータ型をサポートします。これらは PSQL.database.Row クラスにあります。

```
public void setObject(int columnNumber, Serializable object),
public void setObject(String columnName, Serializable object),
public Serializable getObject(int columnNumber),
public Serializable getObject(String columnName),


public InputStream getBinaryStream(int columnNumber),
public InputStream getBinaryStream(String columnName),
public void setBinaryStream(int columnNumber, InputStream inStream),
public void setBinaryStream(String columnName, InputStream inStream)
```

ここで、columnNumber は、行バッファー内の列のシーケンス番号です。シーケンス番号はゼロから始まります。columnName は行バッファー内の列の名前です。object はデータベースに格納されるシリアル化可能な Java オブジェクトです。inStream は、データベースにバイトをストリームとして書き込むのに使用します。

上記のメソッドは 2 つのカテゴリに分割されます。1 つは、Serialized Java オブジェクトで動作するもので、もう 1 つは JavaInputStream オブジェクトで動作するものです。これらのカテゴリは、それぞれ次のセクションで詳しく説明します。次のセクションの例では、以下のスキーマで Employee という名前のテーブルが作成されていることを前提としてます。

メモ Employee_Data および Manager_Report には「not null」の指定があります。これは、PSQL 2000 以降のバージョンがサポートする「真のヌル」を Zen Java クラス ライブラリがサポートしていないためです。

```
table Employees (SS_Num ubigint primary key,
   Employee_Data longvarbinary not null,
   Manager_Report longvarbinary not null)
```

SQL インターフェイスで「1 バイト整数列」に挿入されたデータを Java インターフェイスで取得することはできない

Zen データベース エンジンでは、1 バイト整数は 0 ～ 255 の値を持つものと解釈されます。一方 Java では、このバイト型は符号付きで、-128 ～ 127 の値を持つものと解釈されます。

このような解釈の違いがあるので、データが誤って解釈されないように、コードから以下のプロシージャを実行する必要があります。

▶▶ Java の解釈と Zen データベース エンジンの解釈との間で、1 バイトの整数を変換するには

```
int theOneByteInt = 0;
PSQL.database.Row row  = _rowset.getNext();
theOneByteInt = row.getByte("OneByte") & 0x00FF;
listCourses.add(theOneByteInt + " ");
```

メモ すべての Java データ型は符号付きです。

Java データ型の詳細およびその他の言語に関する情報は、docs.oracle.com/javase/tutorial を参照してください。

サンプル データベース ファイル

本 SDK コンポーネントに付属するサンプル データベース ファイルを使用して、Java アプリケーションを作成することができます。プログラミングの欠陥によりデータベース ファイルを壊してしまった場合は、SDK でインストールされたバックアップ ディレクトリからデータベース ファイルを復元できます。

▶▶ サンプル データベース ファイルを復元するには

1 復元する前に、サンプル アプリケーションや開発環境など、データベース ファイルを開いているプログラムを閉じます。

2 *file_path*¥Zen¥SDK（デフォルトの場所）にインストールしたという前提で、以下のフォルダーを開きます。

file_path¥zen¥sdk¥jcl¥samples¥pvideo¥pvideodb¥dbbackup

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。

3 このディレクトリにあるファイルを 1 つ上のフォルダーにコピーしますが、手順 1 の場合は、以下のフォルダーになります。

file_path¥zen¥sdk¥jcl¥samples¥pvideo¥pvideodb¥

追加の Java サンプル

Java API に付属の追加サンプルがあります。まず、本 SDK コンポーネントで提供する 2 つのテーブルの結合を実証するサンプル、次に Demodata データベースへ接続するサンプル、最後に、オブジェクトのシリアル化を実証するサンプルです。

デフォルトの場所にインストールしたという前提で、これらのサンプルは以下のディレクトリに置かれます。

file_path¥zen¥sdk¥jcl¥samples¥join

file_path¥zen¥sdk¥jcl¥samples¥helloworld¥

file_path¥zen¥sdk¥jcl¥samples¥serialization

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。