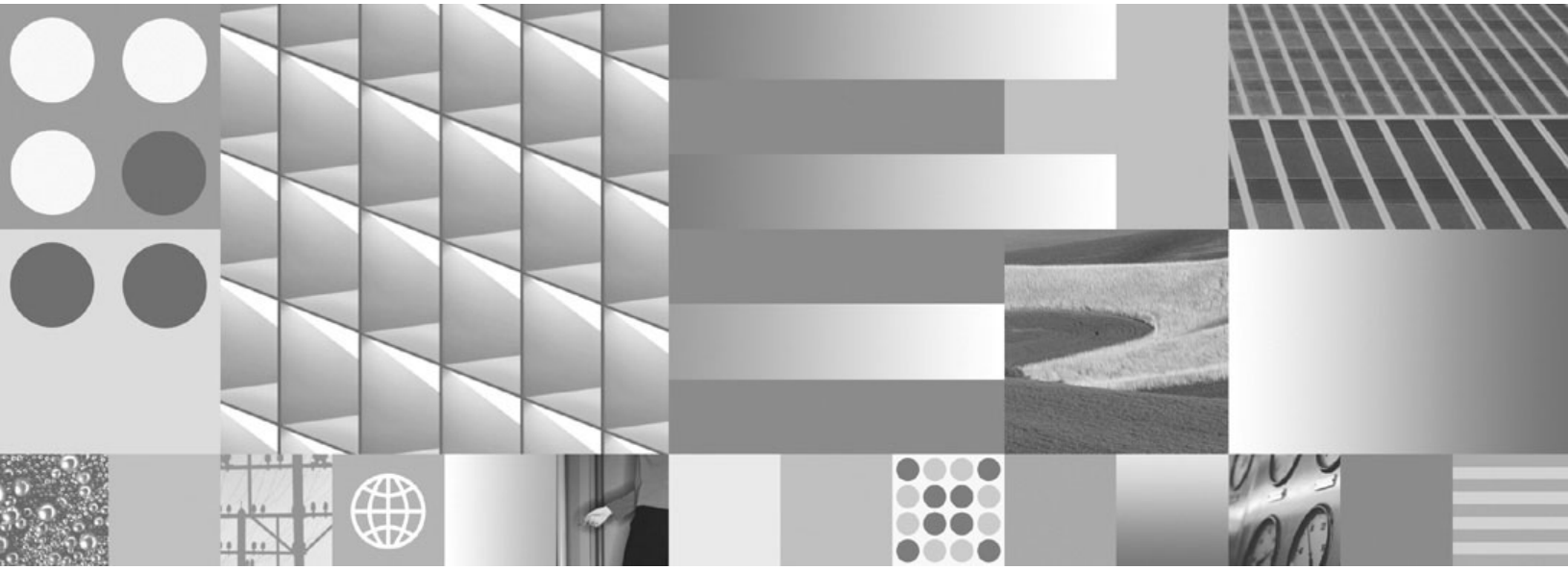
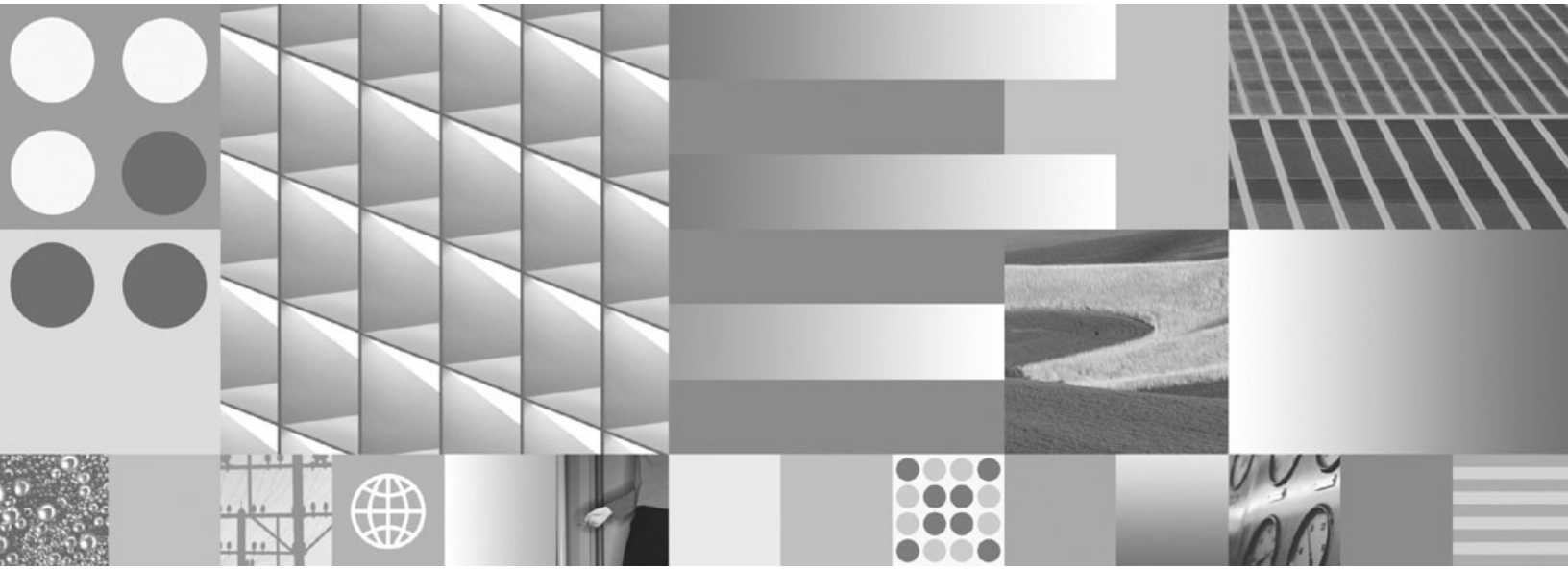
IBM DB2 9.7
for Linux, UNIX, and Windows



Java アプリケーションの開発

SC88-5878-00
(英文原典：SC27-2446-00)

IBM DB2 9.7
for Linux, UNIX, and Windows



Java アプリケーションの開発

SC88-5878-00
(英文原典：SC27-2446-00)

ご注意

本書および本書で紹介する製品をご使用になる前に、565 ページの『付録 B. 特記事項』に記載されている情報をお読みください。

原典： SC27-2446-00
IBM DB2 9.7
for Linux, UNIX, and Windows
Developing Java Applications

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2009.7

目次

本書について

本書では、DB2® for Linux®, UNIX®, and Windows® の Java™ サポートについて説明します。Java アプリケーション・プログラムからリレーショナル・データベースへのアクセスは、このサポートによって可能になります。

本書の対象読者

本書は、次のユーザーを対象にしています。

- 構造化照会言語 (SQL) を十分理解し、Java プログラミング言語を知っている DB2 for Linux, UNIX, and Windows アプリケーション開発者。
- JDBC および SQLJ サポートをインストールする DB2 for Linux, UNIX, and Windows システム・プログラマー。

第 1 章 IBM データ・サーバー用の Java アプリケーション開発

DB2 および IBM® Informix® Dynamic Server (IDS) データベース・システムでは、Java で作成されたクライアント・アプリケーションおよびアプレット用のドライバー・サポートが提供されます。

DB2 および IDS データベース・システムのデータにアクセスするには、JDBC、SQL、または pureQuery を使用できます。

JDBC

JDBC とは、Java アプリケーションによるリレーショナル・データベースへのアクセスに使用されるアプリケーション・プログラミング・インターフェース (API) のことです。 IBM データ・サーバーでの JDBC サポートを使用すると、ローカルの DB2 または IDS データ、あるいは DRDA® がサポートされるサーバー上のリモート・リレーショナル・データにアクセスする Java アプリケーションを作成することができます。

SQLJ

SQLJ では、Java アプリケーションでの組み込み静的 SQL のサポートが提供されます。 SQLJ は、最初は、動的 SQL JDBC モデルを静的 SQL モデルで補完するために、IBM、Oracle、および Tandem によって開発されました。

DB2 との接続のために、Java アプリケーションでは通常、動的 SQL に JDBC を、静的 SQL に SQLJ を使用します。

IDS との接続のために、JDBC または SQLJ アプリケーションの SQL ステートメントが動的に実行されます。

SQLJ は JDBC との相互運用が可能であるため、アプリケーション・プログラムでは同じ作業単位内で JDBC と SQLJ を使用できます。

pureQuery

pureQuery とは、データ・アクセスの開発、最適化、保護、および管理をさらに容易にする、高性能のデータ・アクセス・プラットフォームです。これは以下のもので構成されています。

- 使いやすさと、ベスト・プラクティスを容易に利用できることを目標として構築されたアプリケーション・プログラミング・インターフェース。
- Java および SQL 開発用に IBM Optim Development Studio で提供される開発ツール
- データベース・アクセスを最適化および保護し、管理タスクを簡易化するために、IBM Optim pureQuery Runtime で提供されるランタイム。

pureQuery を使用すれば、データがデータベースにあるとしても JDBC DataSource オブジェクトであるとしても、リレーショナル・データをオブジェクトとして扱う Java アプリケーションを作成できます。また、アプリケーションは、メモリー内の

Java コレクションに格納されたオブジェクトを、それらがリレーショナル・データであるかのように扱うこともできます。リレーショナル・データまたは Java オブジェクトを照会または更新するには、SQL を使用します。

pureQuery について詳しくは、Integrated Data Management インフォメーション・センターを参照してください。

JDBC および SQLJ のサポートされるドライバー

DB2 製品には、2 つのタイプの JDBC ドライバー・アーキテクチャーのサポートが含まれています。

JDBC 仕様に従って、JDBC ドライバー・アーキテクチャーには以下の 4 つのタイプがあります。

タイプ 1
別のデータ・アクセス API (Open Database Connectivity (ODBC) など) へのマッピングとして JDBC API をインプリメントするドライバー。このタイプのドライバーは通常、ネイティブ・ライブラリーに依存しており、移植性が制限されています。 DB2 データベース・システムはタイプ 1 ドライバーを提供していません。

タイプ 2
一部は Java プログラミング言語、一部はネイティブ・コードで作成されているドライバー。ドライバーは、接続先のデータ・ソースに固有のネイティブ・クライアント・ライブラリーを使用します。ネイティブ・コードのために、移植性は制限されています。

タイプ 3
pure Java クライアントを使用し、データベースに依存しないプロトコルを使用してデータベースと通信するドライバー。それから、データベースはクライアントの要求をデータ・ソースに送達します。 DB2 データベース・システムはタイプ 3 ドライバーを提供していません。

タイプ 4
pure Java であり、固有のデータ・ソース用のネットワーク・プロトコルをインプリメントするドライバー。クライアントはデータ・ソースに直接接続します。

DB2 Database for Linux, UNIX, and Windows は以下のドライバーをサポートしています。

ドライバー名	パッケージ	ドライバーのタイプ
Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー	db2java.zip	タイプ 2
IBM Data Server Driver for JDBC and SQLJ	• JDBC 3.0 サポート用の db2jcc.jar および sqlj.zip • 一部の JDBC 4.0 機能のサポート用の db2jcc4.jar および sqlj4.zip	タイプ 2 およびタイプ 4

IBM Data Server Driver for JDBC and SQLJ (タイプ 2 およびタイプ 4)

IBM Data Server Driver for JDBC and SQLJ は、JDBC タイプ 2 および JDBC タイプ 4 の動作を組み込んだ単一ドライバーです。アプリケーションが IBM Data Server Driver for JDBC and SQLJ をロードすると、タイプ 2 およびタイプ 4 のインプリメンテーション用に単一のドライバー・インスタンスがロードされます。アプリケーションは、この単一のドライバー・インスタンスを使用して、タイプ 2 およびタイプ 4 接続を行うことができます。タイプ 2 およびタイプ 4 接続は同時に行うことができます。 IBM Data Server Driver for JDBC and SQLJ タイプ 2 ドライバーの動作を *IBM Data Server Driver for JDBC and SQLJ Type 2 接続*と言います。 IBM Data Server Driver for JDBC and SQLJ タイプ 4 ドライバーの動作を *IBM Data Server Driver for JDBC and SQLJ Type 4 接続* と言います。

2 つのバージョンの IBM Data Server Driver for JDBC and SQLJ が使用可能です。 IBM Data Server Driver for JDBC and SQLJバージョン 3.5*x* は JDBC 3.0 準拠です。 IBM Data Server Driver for JDBC and SQLJ バージョン 4.*x* は JDBC 4.0 準拠です。

IBM Data Server Driver for JDBC and SQLJ は、以下の JDBC および SQLJ 機能をサポートしています。

- バージョン 3.5*x* は、JDBC 3.0 仕様で記述されているすべてのメソッドをサポートしています。
- バージョン 4.*x* は、JDBC 4.0 仕様で記述されているすべてのメソッドをサポートしています。
- Java アプリケーションからのデータ・アクセスを単純化する、SQLJ 標準によって定義された SQLJ アプリケーション・プログラミング・インターフェース。
- 接続プール用に使用可能にされている接続。WebSphere® Application Server または別のアプリケーション・サーバーによって接続プールが行われます。
- Java ユーザー定義関数およびストアード・プロシージャー (IBM Data Server Driver for JDBC and SQLJ Type 2 接続 のみ) 内でのデータベースへの接続。呼び出し側アプリケーションは、タイプ 2 接続またはタイプ 4 接続を使用できます。

  IBM Data Server Driver for JDBC and SQLJ は、Java ルーチンのデフォルト・ドライバーです。
- 分散トランザクション管理のサポート。このサポートは、Java 2 Platform、Enterprise Edition (J2EE) Java Transaction Service (JTS)、および Java Transaction API (JTA) 仕様をインプリメントします。これらは分散トランザクションの X/Open 規格に準拠しています (*Distributed Transaction Processing: The XA Specification*。http://www.opengroup.org で入手可能)。

Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) (非推奨)

DB2 JDBC Type 2 ドライバーは、Java アプリケーションが JDBC を介して DB2 への呼び出しを行うようにします。 DB2 JDBC Type 2 ドライバーへの呼び出しは、Java ネイティブ・メソッドで実装されています。 DB2 JDBC Type 2 ドライバーは DB2 CLI インターフェースを使用して DB2 データベースと通信します。このドライバーを使用する Java アプリケーションは DB2 クライアント上で実行する

必要があります。JDBC 要求はこのクライアントを介して DB2 データベースに送信されます。DB2 JDBC アプリケーション・ドライバーを使用して DB2 for i データ・ソースまたは DB2 for z/OS® 環境内のデータ・ソースにアクセスするには、その前に DB2® Connect™ をインストールする必要があります。

DB2 JDBC Type 2 ドライバーは、以下の JDBC および SQLJ 機能をサポートしています。

- JDBC 1.2 仕様で記述されているほとんどのメソッド、および JDBC 2.0 仕様で記述されている一部のメソッド。
- すべての JDBC メソッドに相当する操作を実行する SQLJ ステートメント
- 接続プール
- 分散トランザクション
- Java ユーザー定義関数およびストアード・プロシージャー

Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーは、今後のリリースではサポートされません。そのため、 IBM Data Server Driver for JDBC and SQLJ に移行することを考慮してください。

JDBC ドライバーとデータベースのバージョンの互換性

IBM Data Server Driver for JDBC and SQLJ の特定のバージョンとデータベースのバージョンの互換性は、使用しているドライバー接続のタイプと、接続先のデータ・ソースのタイプによって異なります。

IBM Data Server Driver for JDBC and SQLJ Type 4 接続の互換性

IBM Data Server Driver for JDBC and SQLJ は、常に以前のリリース・レベルの DB2 データベースと下位互換性を持ちます。例えば、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続 では、 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 に付属する IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 から DB2 Database for Linux, UNIX, and Windowsバージョン 8 データベースまでがサポートされます。

ドライバーが稼働するアプリケーションが新規フィーチャーを全く使用しない場合、 IBM Data Server Driver for JDBC and SQLJ は次のバージョンの DB2 データベースと上位互換性を持ちます。例えば、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続 では、 DB2 for z/OS バージョン 8 に付属する IBM Data Server Driver for JDBC and SQLJバージョン 2.*x* から DB2 for z/OS バージョン 9.1 データベース (ドライバーが稼働するアプリケーションに DB2 for z/OSバージョン 9.1 のフィーチャーが含まれていない場合) までがサポートされます。

IBM Informix Dynamic Server への IBM Data Server Driver for JDBC and SQLJ Type 4 接続は、IDS バージョン 11 以降でのみサポートされます。

IBM Data Server Driver for JDBC and SQLJ Type 2 接続の互換性

一般的に、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続 はローカル・データベース・システムへの接続を対象としており、そのデータベース・バージョンに付属するドライバー・バージョンが使用されます。例えば、 IBM Data

Server Driver for JDBC and SQLJ のバージョン 3.5*x* は、 DB2 Database for Linux, UNIX, and Windows バージョン 9.5、および DB2 for z/OS バージョン 8 および バージョン 9.1 に付属しています。

ただし、ローカル DB2 Database for Linux, UNIX, and Windows データベースへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合は、データベースのバージョンは、ドライバーが付属していた DB2 Database for Linux, UNIX, and Windows バージョンよりも 1 つ前か 1 つ後のバージョンが可能です。ローカル DB2 for z/OS サブシステムへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合、サブシステムのバージョンは、ドライバーが付属していた DB2 for z/OS バージョンよりも 1 つ後のバージョンが可能です。

アプリケーションが接続しているデータベース・バージョンが、ドライバーが付属していたデータベース・バージョンより後のものである場合、アプリケーションは後のデータベース・バージョンの機能を使用できません。

第 2 章 IBM Data Server Driver for JDBC and SQLJ のインストール

IBM Data Server Driver for JDBC and SQLJ のインストール後に、JDBC または SQLJ アプリケーションを準備して実行することができます。

IBM Data Server Driver for JDBC and SQLJ をインストールする前に、以下のソフトウェアが必要です。

- SDK for Java 1.4.2 以降。

  IBM Data Server Runtime Client および IBM Data Server Driver Package 以外のすべての DB2 製品。 DB2 Database for Linux, UNIX, and Windows のインストール・プロセスにより、SDK for Java バージョン 5 は自動的にインストールされます。

  JDBC 4.0 機能を使用する場合は、SDK for Java 6 以降をインストールする必要があります。

  JDBC または SQLJ アプリケーションをシステム上で実行する予定があるが、準備する予定はない場合、必要なのは Java ランタイム環境だけです。

  重要: Java ルーチンに対する SDK for Java 1.4.2 のサポートは使用すべきではありません。今後のリリースで廃止される可能性があります。
- JVM ネイティブ・スレッド・サポート

  DB2 データベースにアクセスする Java アプリケーションを実行する JVM には、ネイティブ・スレッド・サポートが組み込まれていなければなりません。THREADS_FLAG 環境変数を「native」に設定することで、ネイティブ・スレッドを一部の JVM に対するデフォルトのスレッド・サポートとして指定できます。ネイティブ・スレッドをシステム上のデフォルトにする手順については、Java 環境の資料を参照してください。
- System i® サーバーの Unicode サポート

  いずれかの SQLJ または JDBC プログラムが IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用して DB2 for i サーバーに接続する場合、System i オペレーティング・システムは Unicode UTF-8 コード化スキームをサポートする必要があります。以下の表は、Unicode UTF-8 サポートに必要な System PTF をリストしています。

表 1. Unicode UTF-8 サポート用の System i PTF

System i バージョン	PTF 番号
V5R3 以降	なし (サポートは組み込まれている)

- HP-UX クライアントおよびサーバーの Java サポート

  HP-UX サーバー: IBM Data Server Driver for JDBC and SQLJ は、HP-UX のデフォルトの文字セットである Roman8 のデータベースはサポートしません。した

がって、 IBM Data Server Driver for JDBC and SQLJ を使用してアクセスする予定の HP-UX サーバー上にデータベースを作成するには、別の文字セットでデータベースを作成する必要があります。

HP-UX クライアントおよびサーバー: HP-UX システム上の Java 環境では、IBM Data Server Driver for JDBC and SQLJ でストアード・プロシージャーを実行するために特別なセットアップが必要です。

制約事項: IBM Data Server Driver for JDBC and SQLJ を Windows 64 ビット・オペレーティング・システムにインストールする場合、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用して、32 ビット Java アプリケーションから DB2 Database for Linux, UNIX, and Windows インスタンスへ接続することはできません。

これらのステップに従って IBM Data Server Driver for JDBC and SQLJ をインストールします。

1. DB2 Database for Linux, UNIX, and Windows のインストール・プロセス中に、UNIX または Linux 上では `Java support`、Windows 上では `JDBC support` を選択します。 これらの選択はデフォルトです。 DB2 Database for Linux, UNIX, and Windows が JDBC サポートなしでインストール済みの場合、インストール・プロセスを `Custom` モードで実行して JDBC サポートを追加することができます。

   `Java support` または `JDBC support` を選択すると、インストール・プロセスにより以下のアクションが実行されます。

   - IBM Data Server Driver for JDBC and SQLJ のクラス・ファイルをインストールします。

     これらのファイルは、Windows システムの場合は sqllib¥java ディレクトリー、UNIX または Linux システムの場合は sqllib/java ディレクトリーに置かれます。

     これらのファイル名は以下のとおりです。

     db2jcc.jar または db2jcc4.jar

     JDBC 3.0 以前の機能しか組み込まれていないバージョンの IBM Data Server Driver for JDBC and SQLJ を使用する予定であれば、db2jcc.jar を CLASSPATH に組み込みます。

     JDBC 4.0 以前の機能を組み込んでいるバージョンの IBM Data Server Driver for JDBC and SQLJ を使用する予定であれば、db2jcc4.jar を CLASSPATH に組み込みます。

     sqlj.zip または sqlj4.zip

     JDBC 3.0 以前の機能だけを含む SQLJ アプリケーションを準備する予定であれば、sqlj.zip を CLASSPATH に含めてください。

     JDBC 4.0 以前の機能を含む SQLJ アプリケーションを準備する予定であれば、sqlj4.zip を CLASSPATH に含めてください。

   - CLASSPATH を変更して、 IBM Data Server Driver for JDBC and SQLJ のクラス・ファイルを組み込みます。

重要: この手順は、db2jcc.jar および sqlj.zip ファイルに対してのみ、自動的に実行されます。 db2jcc4.jar ファイルまたは sqlj4.zip ファイルを使用する場合には、CLASSPATH を手動で変更する必要があります。 CLASSPATH 内の db2jcc.jar を db2jcc4.jar に、または sqlj.zip を sqlj4.zip に変更してください。

重要: CLASSPATH に db2jcc.jar または db2jcc4.jar を組み込みます。両方のファイルを組み込むことはしないでください。

重要: CLASSPATH に sqlj.zip または sqlj4.zip を組み込みます。両方のファイルを組み込むことはしないでください。db2jcc.jar と sqlj4.zip、または db2jcc4.jar と sqlj.zip を含めることはしないでください。

- IBM Data Server Driver for JDBC and SQLJ のライセンス・ファイルをインストールし、CLASSPATH を変更してそれらを組み込むようにします。

  これらのファイルは、Windows システムの場合は sqllib¥java ディレクトリー、UNIX または Linux システムの場合は sqllib/java ディレクトリーに置かれます。これらのファイル名は以下のとおりです。

表 2. IBM Data Server Driver for JDBC and SQLJ のライセンス・ファイル

ライセンス・ファイル	ライセンス・ファイルが接続先として許可するサーバー	ライセンス・ファイルを含む製品
db2jcc_license_cisuz.jar	DB2 for z/OS DB2 for i	すべての DB2 Connect 製品

  ライセンス・ファイルは、 IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 以降からの、 DB2 Database for Linux, UNIX, and Windows、Cloudscape®、または IBM Informix Dynamic Server (IDS) データベースへの接続には不要です。

- IBM Data Server Driver for JDBC and SQLJ Type 2 接続のサポート用の IBM Data Server Driver for JDBC and SQLJ ネイティブ・ライブラリーをインストールします。

  これらのファイルは、Windows システムの場合は sqllib¥bin ディレクトリー、UNIX または Linux システムの場合は sqllib/lib ディレクトリーに置かれます。

  これらのファイル名は以下のとおりです。

  libdb2jcct2.so
  AIX®、IPF 版 HP-UX、Linux、および Solaris の場合

  libdb2jcct2.sl
  PA-RISC 版 HP-UX の場合

  db2jcct2.dll
  Windows の場合

2. いずれかのデフォルトが不適切である場合は、ドライバー全体にわたる構成プロパティーをカスタマイズします。
3. TCP/IP を構成します。

サーバーは、以下の場合の TCP/IP 通信用に構成する必要があります。

- IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用する JDBC または SQLJ アプリケーション。
- IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用し、接続 URL に *server* および *port* を指定する JDBC または SQLJ アプリケーション。

TCP/IP リスナーが実行されていることを確認します。 TCP/IP リスナーを活動化するには、以下のようにします。

a. 以下のようにして環境変数 DB2COMM を TCPIP に設定します。

```
db2set DB2COMM=TCPIP
```

b. サービス・ファイルで指定されるように、データベース・マネージャーの構成ファイルを TCP/IP サービス名で更新します。

```
db2 update dbm cfg using SVCENAME TCP/IP-service-name
```

アプレットおよび SQLJ プログラムに使用するポート番号は、データベース・マネージャー構成ファイルで使用される TCP/IP SVCENAME 番号と同じものにする必要があります。

c. サービス名の設定を有効にするには、db2stop および db2start コマンドを実行します。

4. Java ストアード・プロシージャーまたはユーザー定義関数の実行を予定している DB2 Database for Linux, UNIX, and Windows サーバー上では、DB2_USE_DB2JCCT2_JROUTINE 環境変数が設定されていないか、またはそのデータベース・サーバー上でデフォルト値の YES、yes、ON、on、TRUE、true、または 1 に設定されていることを確認してください。この設定は、Java ストアード・プロシージャーが IBM Data Server Driver for JDBC and SQLJ の下で実行することを示しています。

   ストアード・プロシージャーを Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーの下で実行する必要がある場合、DB2_USE_DB2JCCT2_JROUTINE 環境変数を OFF に設定します。

5. Java ストアード・プロシージャーまたはユーザー定義関数の実行を予定している DB2 Database for Linux, UNIX, and Windows サーバー上では、データベース・マネージャー構成を更新して、SDK for Java が置かれているパスを組み込みます。

   これは、サーバーのコマンド行での場合と類似のコマンドを入力することで実行できます。

   - *UNIX または Linux 上のデータベース・システムの場合:*

     ```
     db2 update dbm cfg using JDK_PATH /home/db2inst/jdk15
     ```

     /home/db2inst/jdk15 は、SDK for Java のインストール先のパスです。

   - *Windows 上のデータベース・システムの場合:*

     ```
     db2 update dbm cfg using JDK_PATH c:¥Program Files¥jdk15
     ```

     c:¥Program Files¥jdk15 は、SDK for Java のインストール先のパスです。

DB2 データベース・マネージャー構成の JDK_PATH フィールド用の正しい値を確認するには、以下のコマンドをデータベース・サーバー上に入力します。

```
db2 get dbm cfg
```

出力をファイルにリダイレクトすれば、一層容易に表示できます。 JDK_PATH フィールドは、出力の先頭近くに表示されます。

6. DB2 Database for Linux, UNIX, and Windows サーバー上にある SQL プロシージャーを Java プログラムから呼び出す予定であり、データベース・サーバーのテリトリー・コードと関連付けられた日付と時刻の形式が USA 形式では**ない**場合、以下のアクションを実行します。

   a. 以下のようにして DB2_SQLROUTINE_PREPOPTS レジストリー変数をデータベース・サーバー上で設定し、デフォルトの日時形式が ISO であることを示します。

      ```
      db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
      ```

   b. Java プログラムから呼び出す予定の既存の SQL プロシージャーを再定義します。

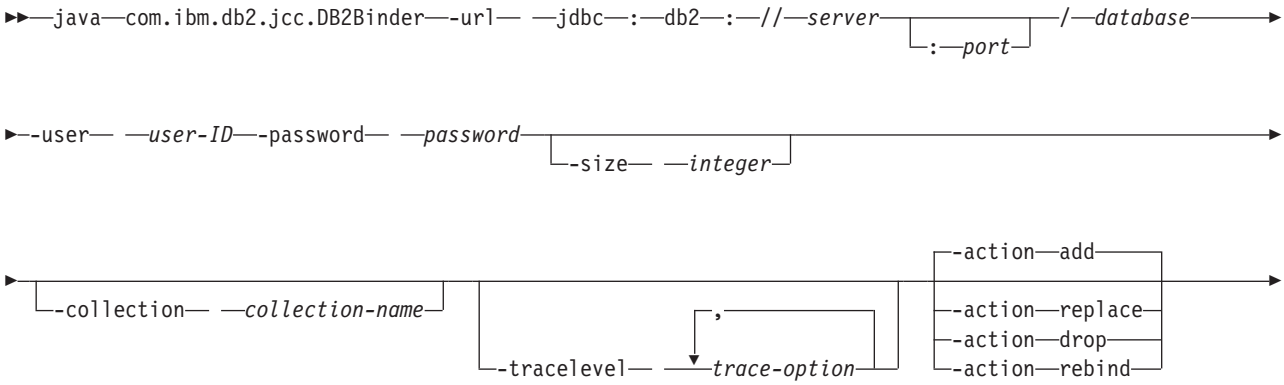
   これらのステップは、呼び出し側のアプリケーションが日付と時刻の値を正しく受け取るために必要です。

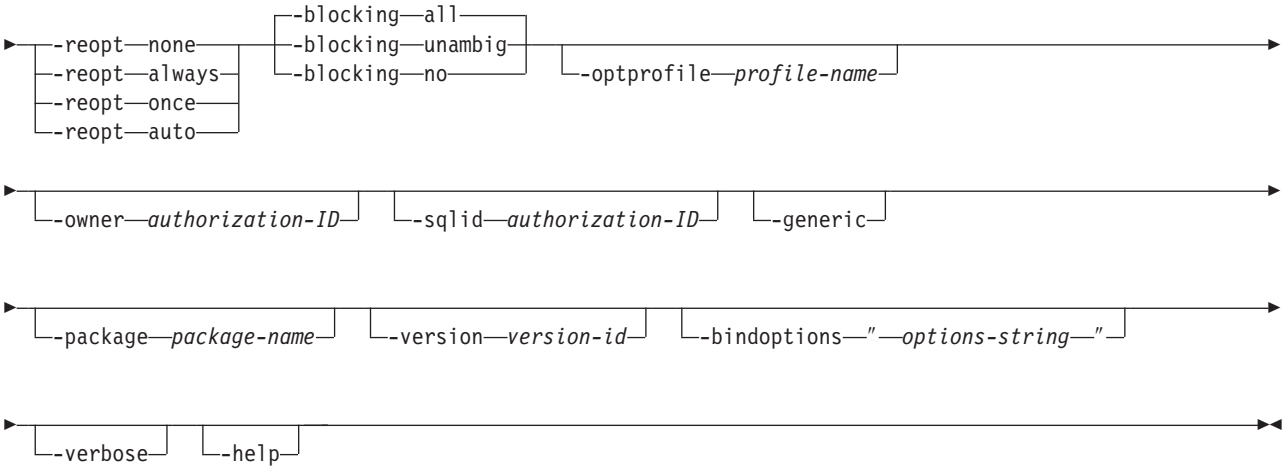
7. DB2 for z/OS データベース・サーバーに Java アプリケーションでアクセスする予定の場合は、Java プログラムから DB2 for z/OS サーバーにアクセスするための特殊なセットアップに関する説明に従ってください。

DB2Binder ユーティリティー

DB2Binder ユーティリティーは、 IBM Data Server Driver for JDBC and SQLJ によってデータベース・サーバーで使用される DB2 パッケージをバインドし、パッケージに対する EXECUTE 権限を PUBLIC に与えます。オプションで、DB2Binder ユーティリティーは IBM Data Server Driver for JDBC and SQLJ の一部ではない DB2 パッケージを再バインドすることができます。

DB2Binder の構文





DB2Binder オプションの説明

-url

IBM Data Server Driver for JDBC and SQLJ パッケージをバインドするデータ・ソースを指定します。 -url 値の可変部は、以下のとおりです。

server
データベース・サーバーが存在するオペレーティング・システムのドメイン名または IP アドレス。

port
データベース・サーバーに割り当てられている TCP/IP サーバーのポート番号。デフォルトは 446 です。

database
SYSIBM.LOCATIONS カタログ表で定義されている、データベース・サーバーのロケーション名。

-user

パッケージをバインドするユーザー ID を指定します。このユーザーは、パッケージに対する BIND 権限を持っている必要があります。

-action

パッケージに実行するアクションを指定します。

add パッケージを作成できるのは、そのパッケージがまだ存在しない場合のみであることを指示します。 add はデフォルトです。

replace
同名のパッケージがすでに存在する場合でも、パッケージを作成できることを指示します。新規のパッケージが、古いパッケージに取って代わります。

rebind 既存のパッケージを再バインドすることを示します。このオプションは IBM Data Server Driver for JDBC and SQLJ パッケージには適用されません。 -action rebind を指定する場合、-generic も指定する必要があります。

drop パッケージをドロップすることを示します。

- IBM Data Server Driver for JDBC and SQLJ パッケージの場合、-action ドロップは一部またはすべての IBM Data Server Driver for JDBC and SQLJ パッケージをドロップすることを示します。パッケージの数は、-size パラメーターに応じて異なります。
- ユーザー・パッケージの場合、-action ドロップは、指定されたパッケージをドロップすることを示します。

-action drop は、ターゲット・データベース・サーバーが DB2 for z/OS の場合にのみ適用されます。

-size

同時にオープンできる Statement、PreparedStatement、または CallableStatement オブジェクトの数、あるいはドロップされる IBM Data Server Driver for JDBC and SQLJ パッケージの数を制御します。

-size パラメーターの意味は、-action パラメーターに以下のように依存します。

- -action の値が add または replace であれば、-size の値は IBM Data Server Driver for JDBC and SQLJ がバインドする DB2 パッケージの数の計算に使用される整数です。 -size の値が *integer* であれば、パッケージの合計数は次のようになります。

```
number-of-isolation-levels*
number-of-holdability-values*
integer+
number-of-packages-for-static-SQL
= 4*2*integer+1
```

  -action add または -action replace のデフォルトの -size 値は 3 です。

  たいていの場合、デフォルトの 3 が適切です。アプリケーションが SQLException を -805 SQLCODE でスローする場合、アプリケーションが未使用のすべてのリソースをクローズしていることを確認してください。クローズしている場合は、-size 値を増やします。

  -action の値が replace で、-size の値が既存のものより少ない数のパッケージという結果になる場合は、ドロップされるパッケージはありません。
- -action の値が drop の場合、-size の値はドロップされるパッケージの数です。 -size が指定されない場合、すべての IBM Data Server Driver for JDBC and SQLJ パッケージがドロップされます。
- -action の値が rebind の場合、-size は無視されます。

-collection

IBM Data Server Driver for JDBC and SQLJ またはユーザー・パッケージのコレクション ID を指定します。デフォルトは NULLID です。 DB2Binder はこの値を大文字に変換します。

`com.ibm.db2.jcc.DB2Binder` を複数回実行して、-collection に毎回異なる値を指定すれば、単一のデータベース・サーバー上で IBM Data Server Driver for JDBC and SQLJ パッケージのインスタンスを複数作成することができます。実行時に、`currentPackageSet` プロパティーを -collection 値に一致する値に設定することによって、 IBM Data Server Driver for JDBC and SQLJ のコピーを選択します。

-tracelevel
DB2Binder の実行時に何をトレースするかを指定します。

-reopt
DB2 for z/OS データベース・サーバーが実行時にアクセス・パスを判別するかどうかを指定します。このオプションは、 DB2 for z/OS データベース・サーバーへの接続の場合にのみ有効です。指定しない場合、このオプションはデータベース・サーバーには送られません。その場合、データベース・サーバーは再最適化の動作を決定します。

なし 実行時にアクセス・パスを判別しないよう指定します。

always ステートメントを実行するたびにアクセス・パスを判別することを指定します。

once 実行時に DB2 が一度だけ動的ステートメントのアクセス・パスを判別してキャッシュに入れることを指定します。 DB2 は、準備されたステートメントが無効にされるか、動的ステートメントのキャッシュから除去されて準備が再度必要になるまで、このアクセス・パスを使用します。

auto データベース・サーバーによってアクセス・パスが自動的に決定されることを指定します。

-blocking
カーソルの行ブロッキングのタイプを指定します。

ALL FOR READ ONLY 節で指定したカーソル、または FOR UPDATE として指定していないカーソルに対して、ブロッキングが行われます。

UNAMBIG
FOR READ ONLY 節で指定したカーソルに対して、ブロッキングが行われます。

FOR READ ONLY または FOR UPDATE 節で宣言されておらず、*未確定* ではなく*読み取り専用* のカーソルはブロックされます。*未確定* カーソルはブロックされません。

NO どのカーソルにもブロッキングは行われません。

読み取り専用カーソルおよび未確定カーソルの定義については、「DECLARE CURSOR」を参照してください。

-optprofile
パッケージのデータ変更ステートメントの最適化に使用される最適化プロファイルを指定します。このプロファイルは、ターゲット・サーバー上に存在しなければならない XML ファイルです。 -optprofile が指定されておらず、CURRENT OPTIMIZATION PROFILE 特殊レジスターが設定されている場合、CURRENT OPTIMIZATION PROFILE の値が使用されます。 -optprofile が指定されておらず、CURRENT OPTIMIZATION PROFILE が設定されていない場合、最適化プロファイルは使用されません。

-optprofile は、 DB2 Database for Linux, UNIX, and Windows データベース・サーバーへの接続の場合にのみ有効です。

-owner

パッケージの所有者の許可 ID を指定します。デフォルト値はデータベース・サーバーによって設定されます。

-owner は、 IBM Data Server Driver for JDBC and SQLJ パッケージだけに適用されます。

-sqlid

IBM Data Server Driver for JDBC and SQLJ パッケージに関する GRANT 操作を DB2Binder が実行する前に CURRENT SQLID 特殊レジスターに設定しておく値を指定します。 1 次許可 ID にはパッケージに関する特権を付与するだけの十分なレベルの権限がなく、かつそのような特権を持つ 2 次許可 ID がこの 1 次許可 ID に関連付けられている場合は、-sqlid を 2 次許可 ID に設定してください。

-sqlid は、 DB2 for z/OS データベース・サーバーへの接続の場合にのみ有効です。

-generic

DB2Binder が IBM Data Server Driver for JDBC and SQLJ パッケージの代わりにユーザー・パッケージを再バインドすることを指定します。 -generic を指定する場合、-action rebind および -package も指定しなければなりません。

-package

再バインドされるパッケージの名前を指定します。このオプションはユーザー・パッケージだけに適用されます。 -package を指定する場合、-action rebind および -generic も指定しなければなりません。

-version

再バインドされるパッケージのバージョン ID を指定します。 -version を指定する場合、-action rebind、-package、および -generic も指定しなければなりません。

-bindoptions

引用符で囲んだストリングを指定します。このストリングの内容は、ユーザー・パッケージの再バインド用のオプションを表す、1 つ以上のパラメーターと値の組です。ストリング内のすべての項目は、スペースで区切られます。

```
"parm1 value1 parm2 value2 ... parmn valuen"
```

-bindoptions は、 IBM Data Server Driver for JDBC and SQLJ パッケージには適用されません。

指定可能なパラメーターと値は以下のとおりです。

bindObjectExistenceRequired

すべてのオブジェクトまたは必要とされる特権が再バインド時に存在しない場合、データベース・サーバーはエラーを発行して、パッケージを再バインドしないかどうかを指定します。可能な値は以下のとおりです。

true このオプションは SQLERROR(NOPACKAGE) バインド・オプションに対応します。

false このオプションは SQLERROR(CONTINUE) バインド・オプションに対応します。

degreeIOParallelism

パフォーマンスを最大化するために、並列処理を使用して静的照会の実行を試行するかどうかを指定します。可能な値は以下のとおりです。

1 並列処理はありません。

このオプションは DEGREE(1) バインド・オプションに対応します。

-1 並列処理を許可します。

このオプションは DEGREE(ANY) バインド・オプションに対応します。

packageAuthorizationRules

以下の動的 SQL 属性に実行時に適用される値を決定します。

- 許可の検査に使用される許可 ID
- 非修飾オブジェクトに使用される修飾子
- データベース・サーバーが動的 SQL ステートメントを構文解析して意味的に検証するために使用する、アプリケーション・プログラミング・オプションのソース
- 動的 SQL ステートメントが GRANT、REVOKE、ALTER、CREATE、DROP、および RENAME ステートメントを組み込めるかどうか

可能な値は以下のとおりです。

0 実行の動作を使用します。これはデフォルトです。

このオプションは DYNAMICRULES(RUN) バインド・オプションに対応します。

1 バインドの動作を使用します。

このオプションは DYNAMICRULES(BIND) バインド・オプションに対応します。

2 パッケージがストアード・プロシージャーまたはユーザー定義関数パッケージとして、またはその下で実行する場合、データベース・サーバーは呼び出しの動作を使用して動的 SQL ステートメントを処理します。そうでない場合、データベース・サーバーは実行の動作を使用して動的 SQL ステートメントを処理します。

このオプションは DYNAMICRULES(INVOKERUN) バインド・オプションに対応します。

3 パッケージがストアード・プロシージャーまたはユーザー定義関数パッケージとして、またはその下で実行する場合、データベース・サーバーは呼び出しの動作を使用して動的 SQL ステートメントを処理します。そうでない場合、データベース・サーバーはバインドの動作を使用して動的 SQL ステートメントを処理します。

このオプションは DYNAMICRULES(INVOKEBIND) バインド・オプションに対応します。

4 パッケージがストアード・プロシージャーまたはユーザー定義関数パッケージとして、またはその下で実行する場合、データベース・サーバーは定義の動作を使用して動的 SQL ステートメントを処理します。そうでない場合、データベース・サーバーは実行の動作を使用して動的 SQL ステートメントを処理します。

このオプションは DYNAMICRULES(DEFINERUN) バインド・オプションに対応します。

5 パッケージがストアード・プロシージャーまたはユーザー定義関数パッケージとして、またはその下で実行する場合、データベース・サーバーは定義の動作を使用して動的 SQL ステートメントを処理します。そうでない場合、データベース・サーバーはバインドの動作を使用して動的 SQL ステートメントを処理します。

このオプションは DYNAMICRULES(DEFINEBIND) バインド・オプションに対応します。

packageOwnerIdentifier
パッケージの所有者の許可 ID を指定します。

isolationLevel
他の実行中のアプリケーションの影響からアプリケーションをどの程度隔離するかを指定します。可能な値は以下のとおりです。

1 非コミット読み取り

このオプションは ISOLATION(UR) バインド・オプションに対応します。

2 カーソル固定

このオプションは ISOLATION(CS) バインド・オプションに対応します。

3 読み取り固定

このオプションは ISOLATION(RS) バインド・オプションに対応します。

4 反復可能読み取り

このオプションは ISOLATION(RR) バインド・オプションに対応します。

releasePackageResourcesAtCommit
各コミット・ポイントでプログラムが使用するリソースをいつ解放するかを指定します。可能な値は以下のとおりです。

true このオプションは RELEASE(COMMIT) バインド・オプションに対応します。

false このオプションは RELEASE(DEALLOCATE) バインド・オプションに対応します。

-bindoptions を指定する場合、-generic も指定する必要があります。

-verbose

DB2Binder ユーティリティーがバインド処理に関する詳細情報を表示するように指定します。

-help

DB2Binder ユーティリティーで、サポートされる各オプションの説明を表示するように指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

ターゲット・オペレーティング・システムが Windows ではない場合の DB2Binder 戻りコード

DB2Binder のターゲット・データ・ソースが Windows オペレーティング・システム上にない場合、DB2Binder は以下の戻りコードのいずれかを戻します。

表 3. ターゲット・オペレーティング・システムが Windows ではない場合の DB2Binder 戻りコード

戻りコード	意味
0	成功した実行。
1	DB2Binder の実行中にエラーが発生しました。

ターゲット・オペレーティング・システムが Windows の場合の DB2Binder 戻りコード

DB2Binder のターゲット・データ・ソースが Windows オペレーティング・システム上にある場合、DB2Binder は以下の戻りコードのいずれかを戻します。

表 4. ターゲット・オペレーティング・システムが Windows の場合の DB2Binder 戻りコード

戻りコード	意味
0	成功した実行。
-100	バインド・オプションが指定されていません。
-101	-url 値が指定されていません。
-102	-user 値が指定されていません。
-103	-password 値が指定されていません。
-200	有効なバインド・オプションが指定されていません。
-114	-package オプションは指定されませんでしたが、-generic オプションは指定されました。
-201	-url 値が無効です。
-204	-action 値が無効です。
-205	-blocking 値が無効です。
-206	-collection 値が無効です。
-207	-dbprotocol 値が無効です。
-208	-keepdynamic 値が無効です。
-210	-reopt 値が無効です。
-211	-size 値が無効です。

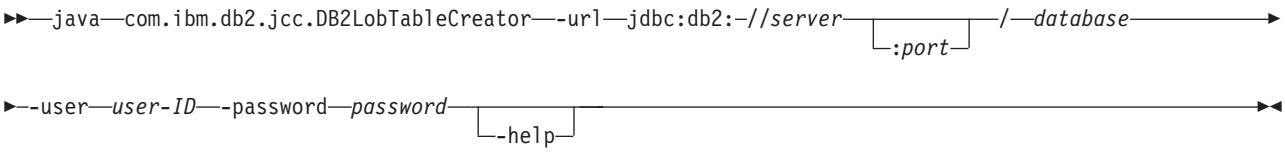
表 4. ターゲット・オペレーティング・システムが Windows の場合の DB2Binder 戻りコード (続き)

戻りコード	意味
-212	-tracelevel 値が無効です。
-307	-dbprotocol 値はターゲット・データベース・サーバーによりサポートされていません。
-308	-keepdynamic 値はターゲット・データベース・サーバーによりサポートされていません。
-310	-reopt 値はターゲット・データベース・サーバーによりサポートされていません。
-313	-optprofile 値はターゲット・データベース・サーバーによりサポートされていません。
-401	Binder クラスが検出されませんでした。
-402	データベース・サーバーへの接続が失敗しました。
-403	データベース・サーバーの DatabaseMetaData 検索が失敗しました。
-501	クラスターで使用可能なパッケージはこれ以上ありません。
-502	既存のパッケージは無効です。
-503	バインド処理がエラーを戻しました。
-999	文書化されていないバインド・オプションの処理中にエラーが発生しました。

DB2LobTableCreator ユーティリティー

DB2LobTableCreator ユーティリティーは、 DB2 for z/OS データベース・サーバー上に表を作成します。それらの表は、DBCLOB または CLOB 列のデータにアクセスするために LOB ロケーターを使用する JDBC または SQLJ アプリケーションにより必要とされます。

DB2LobTableCreator 構文



DB2LobTableCreator オプションの説明

-url

DB2LobTableCreator が実行するデータ・ソースを指定します。 -url 値の可変部は、以下のとおりです。

jdbc:db2:
接続先が DB2 ファミリーのサーバーであることを示します。

server
データベース・サーバーのドメイン名または IP アドレス。

port
データベース・サーバーに割り当てられている TCP/IP サーバーのポート番号。これは 0 から 65535 の間の整数です。デフォルトは 446 です。

database
データベース・サーバーの名前。

database は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

-user
DB2LobTableCreator を実行するユーザー ID を指定します。このユーザーは DSNATPDB データベース内に表を作成する権限を持っている必要があります。

-password
ユーザー ID のパスワードを指定します。

-help
DB2LobTableCreator ユーティリティーがサポートする各オプションを記述することを指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

IBM Data Server Driver for JDBC and SQLJ の構成プロパティーのカスタマイズ

IBM Data Server Driver for JDBC and SQLJ 構成プロパティーを使用して、ドライバー全体にわたる有効範囲を持つプロパティー値を設定することができます。これらの設定は、アプリケーションおよび DataSource インスタンス全体に適用されます。アプリケーションのソース・コードまたは DataSource 特性を変更せずに、この設定を変更することができます。

それぞれの IBM Data Server Driver for JDBC and SQLJ 構成プロパティーの設定は以下の形式になります。

```
property=value
```

構成プロパティーは以下の方法で設定できます。

- 構成プロパティーを Java システム・プロパティーとして設定します。Java システム・プロパティーとして設定される構成プロパティー値は、別の方法で設定されるすべての構成プロパティー値をオーバーライドします。

  スタンドアロン Java アプリケーションの場合、java コマンドを実行する際に構成プロパティーごとに -D*property*=*value* を指定することによって、構成プロパティーを Java システム・プロパティーとして設定できます。
- db2.jcc.propertiesFile Java システム・プロパティーで指定した名前を持つリソースに構成プロパティーを設定します。例えば、db2.jcc.propertiesFile 値に絶対パス名を指定することができます。

スタンドアロン Java アプリケーションの場合、java コマンドを実行する際に -Ddb2.jcc.propertiesFile=*path* オプションを指定することによって、構成プロパティーを設定できます。

- DB2JccConfiguration.properties という名前のリソースに構成プロパティーを設定します。標準 Java リソース検索は、DB2JccConfiguration.properties を検索するのに使用されます。 IBM Data Server Driver for JDBC and SQLJ がこのリソースを検索するのは、db2.jcc.propertiesFile Java システム・プロパティーを設定していない場合のみです。

  DB2JccConfiguration.properties はスタンドアロン・ファイルにするか、あるいは JAR ファイルに組み込むことができます。

  DB2JccConfiguration.properties ファイルが ISO 8859-1 (Latin-1) コード化スキームであるか、あるいは一部の Unicode エンコード (¥*udddd*) 文字を含む Latin-1 コード化スキームである場合、 IBM Data Server Driver for JDBC and SQLJ がそのファイルを使用する前に文字変換を行う必要はありません。
  DB2JccConfiguration.properties ファイルがその他のコード化スキームである場合、Java native2ascii コンバーターを使用して、ファイルの内容を Latin-1 または Unicode エンコード文字に変換する必要があります。

  DB2JccConfiguration.properties がスタンドアロン・ファイルである場合、DB2JccConfiguration.properties のパスは CLASSPATH 連結内に存在している必要があります。

  DB2JccConfiguration.properties が JAR ファイルにある場合、その JAR ファイルは CLASSPATH 連結内に存在している必要があります。

Java プログラムから DB2 for z/OS サーバーにアクセスするための特別なセットアップ

DB2 for z/OS データベース・サーバーにアクセスする JDBC アプリケーションまたは SQLJ アプリケーションを作成する計画がある場合は、 IBM Data Server Driver for JDBC and SQLJ のインストール・プロセスで追加の手順が必要になります。

DB2 for z/OS サーバーへの接続を可能にするには、以下の手順を実行します。

1. DB2 for z/OS バージョン 7 またはバージョン 8 のデータベース・サーバーに接続する計画がある場合は、それらのデータベース・サーバーに以下の PTF をインストールします。

表 5. DB2 for z/OS ストアード・プロシージャーのための PTF

DB2 for z/OS	PTF 番号または APAR 番号
バージョン 7	UQ72083、UQ93889、UK21848
バージョン 8	UQ93890、UK21849
バージョン 9	PK44166

2. com.ibm.db2.jcc.DB2Binder ユーティリティーを実行して、サーバーで IBM Data Server Driver for JDBC and SQLJ が使用する DB2 パッケージをバインドします。

3. DB2 for z/OS データベース・サーバーでジョブ DSNTIJMS をカスタマイズして実行します。

   DSNTIJMS は、データ・セット *prefix*.SDSNSAMP にあります。このジョブは、以下の機能を実行します。

   - DatabaseMetaData のメソッド、トレース、エラー・メッセージ・フォーマットをサポートするために、以下のストアード・プロシージャーを作成します。
     - SQLCOLPRIVILEGES
     - SQLCOLUMNS
     - SQLFOREIGNKEYS
     - SQLFUNCTIONS
     - SQLFUNCTIONCOLUMNS
     - SQLGETTYPEINFO
     - SQLPRIMARYKEYS
     - SQLPROCEDURECOLS
     - SQLPROCEDURES
     - SQLSPECIALCOLUMNS
     - SQLSTATISTICS
     - SQLTABLEPRIVILEGES
     - SQLTABLES
     - SQLUDTS
     - SQLCAMESSAGE
   - CLOB 列または DBCLOB 列にデータを効率的に格納し、CLOB または DBCLOB の検索で LOB ロケーターを使用するために、以下の表を作成します。
     - SYSIBM.SYSDUMMYU
     - SYSIBM.SYSDUMMYA
     - SYSIBM.SYSDUMMYE

     これらの表を作成する別の方法として、それぞれの DB2 for z/OS サーバーに対してクライアントで com.ibm.db2.jcc.DB2LobTableCreator ユーティリティーを実行することも可能です。

4. OS/390® サーバーと z/OS サーバーで Unicode サポートを使用可能にします。

   SQLJ プログラムまたは JDBC プログラムで IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用して DB2 for z/OS バージョン 7 サーバーに接続する場合は、OS/390 または z/OS のオペレーティング・システムで Unicode UTF-8 コード化スキームをサポートする必要があります。そのサポートのためには、OS/390 バージョン 2 リリース 9 APAR OW44581、あるいは OS/390 または z/OS のそれ以降のリリースと、OS/390 R8/R9/R10 Unicode サポートが必要です。追加情報については、情報 APAR II13048 と II13049 を参照してください。

5. IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用して DB2 for z/OS バージョン 7 サーバーに分散トランザクションをインプリメントする計画がある場合は、それぞれの DB2 for z/OS バージョン 7 サーバーに対して DB2T4XAIndoubtUtil ユーティリティーを 1 回ずつ実行します。

DB2 UDB (OS/390 および z/OS 版) バージョン 7 サーバーを使用する分散トランザクション用の DB2T4XAIndoubtUtil

IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用して、DB2 UDB (OS/390 および z/OS 版) バージョン 7 サーバーを組み込んだ分散トランザクションをインプリメントすることを計画している場合、それらのサーバーに対して DB2T4XAIndoubtUtil ユーティリティーを実行する必要があります。

DB2T4XAIndoubtUtil により、XA 仕様をインプリメントする分散トランザクションの組み込みサポートを持たないバージョン 7 サーバーは、サポートをエミュレートすることができます。

DB2T4XAIndoubtUtil は、次のタスクの一方または両方を実行します。

- SYSIBM.INDOUBT という名前の表および関連する索引を作成します。
- T4XAIN01、T4XAIN02、T4XAIN03、および T4XAIN04 という名前の DB2 パッケージをバインドします。

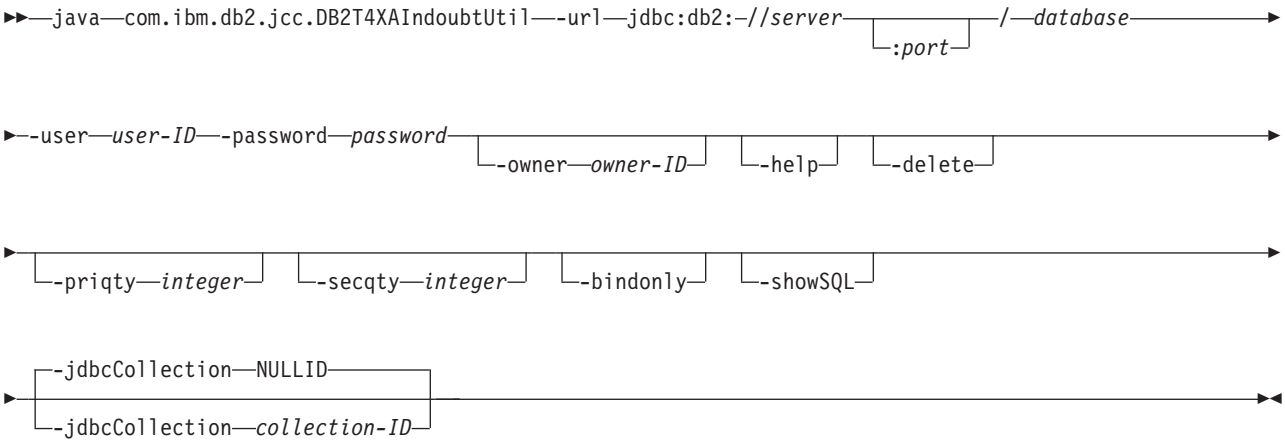
パッケージ T4XAIN01、T4XAIN02、T4XAIN03、および T4XAIN04 の作成およびドロップを、DB2T4XAIndoubtUtil を実行するだけで行います。
SYSTEM.INDOUBT とその索引は手動で作成およびドロップできますが、このユーティリティーを使用することをお勧めします。これらのオブジェクトを手動で作成する方法についての説明は、『DB2T4XAIndoubtUtil の使用上の注意』を参照してください。

DB2T4XAIndoubtUtil の許可

DB2T4XAIndoubtUtil ユーティリティーを実行して、SYSTEM.INDOUBT の作成とパッケージ T4XAIN01、T4XAIN02、T4XAIN03、および T4XAIN04 のバインドを行うには、SYSADM 権限が必要です。

DB2T4XAIndoubtUtil を、パッケージ T4XAIN01、T4XAIN02、T4XAIN03、および T4XAIN04 をバインドするためだけに実行するには、パッケージに対する BIND 権限が必要です。

DB2T4XAIndoubtUtil の構文



DB2T4XAIndoubtUtil のパラメーターの説明

-url

DB2T4XAIndoubtUtil が実行するデータ・ソースを指定します。 -url 値の可変部は、以下のとおりです。

jdbc:db2:
接続先が DB2 ファミリーのサーバーであることを示します。

server
データベース・サーバーのドメイン名または IP アドレス。

port
データベース・サーバーに割り当てられている TCP/IP サーバーのポート番号。これは 0 から 65535 の間の整数です。デフォルトは 446 です。

database
データベース・サーバーの名前。

database は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

-user

DB2T4XAIndoubtUtil を実行するユーザー ID を指定します。このユーザーは、SYSADM 権限を持っているか、または SYSADM 権限がある 2 次許可 ID に対応する RACF® グループのメンバーでなければなりません。

-password

ユーザー ID のパスワードを指定します。

-owner

SYSADM 権限を持つ 2 次許可 ID を指定します。 -user パラメーター値が SYSADM 権限を持たない場合は、-owner パラメーターを使用します。 -user パラメーター値は、名前が *owner-ID* の RACF グループのメンバーでなければなりません。

-owner パラメーターの指定時に、DB2T4XAIndoubtUtil は *owner-ID* を以下のものとして使用します。

- SYSIBM.INDOUBT 表を作成するための許可 ID。
- T4XAIN01、T4XAIN02、T4XAIN03、および T4XAIN04 パッケージの所有者の許可 ID。それらのパッケージ内の SQL ステートメントは、*owner-ID* の権限を使用して実行されます。

-help

DB2T4XAIndoubtUtil ユーティリティーがサポートする各オプションを記述することを指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

-delete

DB2T4XAIndoubtUtil の以前の実行時に作成されたオブジェクトを、DB2T4XAIndoubtUtil ユーティリティーが削除することを指定します。

-priqty
SYSIBM.INDOUBT 表を含む表スペースの 1 次スペース割り振りを、K バイト (KB) 単位で指定します。 -priqty のデフォルト値は 1000 です。

重要: ページ・サイズで除算した、SYSIBM.INDOUBT が置かれる表スペースの -priqty 値は、一定の時間に許可される未確定トランザクションの最大数よりも大きくなければなりません。例えば、4 KB のページ・サイズの場合、デフォルトの -priqty 値である 1000 では、およそ 250 の並行未確定トランザクションが可能です。

-secqty
SYSIBM.INDOUBT 表を含む表スペースの 2 次スペース割り振りを、K バイト (KB) 単位で指定します。 -secqty のデフォルト値は 0 です。

推奨事項: -secqty 値には必ずデフォルト値 0 を使用し、-priqty 値には並行未確定トランザクションの最大数に対応する十分な大きさの値を指定してください。

-bindonly
DB2T4XAIndoubtUtil ユーティリティーで T4XAIN01、T4XAIN02、T4XAIN03、および T4XAIN04 パッケージをバインドし、そのパッケージを実行する許可を PUBLIC に与えるが、SYSIBM.INDOUBT 表は作成しないように指定します。

-showSQL
DB2T4XAIndoubtUtil ユーティリティーが、実行する SQL ステートメントを表示することを指定します。

-jdbcCollection *collection-name*|**NULLID**
IBM Data Server Driver for JDBC and SQLJ パッケージが DB2Binder ユーティリティーでバインドされるときに使用された -collection パラメーターの値を指定します。明示的または暗黙に指定された -collection パラメーターの値が NULLID でなかった 場合は、-jdbcCollection パラメーターを指定しなければなりません。

デフォルトは -jdbcCollection NULLID です。

DB2T4XAIndoubtUtil の使用上の注意

SYSTEM.INDOUBT 表およびその索引を手動で作成するには、以下の SQL ステートメントを使用します。

```
CREATE TABLESPACE INDBTTS
 USING STOGROUP
 LOCKSIZE ROW
 BUFFERPOOL BP0
 SEGSIZE 32
 CCSID EBCDIC;

CREATE TABLE SYSIBM.INDOUBT(indbtXid VARCHAR(140) FOR BIT DATA NOT NULL,
                            uowId VARCHAR(25) FOR BIT DATA NOT NULL,
                            pSyncLog VARCHAR(150) FOR BIT DATA,
                            cSyncLog VARCHAR(150) FOR BIT DATA)
 IN INDBTTS;

CREATE UNIQUE INDEX INDBTIDX ON SYSIBM.INDOUBT(indbtXid, uowId);
```

DB2T4XAIndoubtUtil の例

DB2T4XAIndoubtUtil を実行して、IP アドレス mvs1、ポート番号 446、および DB2 ロケーション名 SJCEC1 を持つ DB2 for OS/390 and z/OS バージョン 7 サブシステムが、XA 分散トランザクションに参加できるようにします。

```
java com.ibm.db2.jcc.DB2T4XAIndoubtUtil -url jdbc:db2://mvs1:446/SJCEC1 ¥
  -user SYSADM -password mypass
```

HP-UX 環境で Java ルーチンを実行するための特別なセットアップ

PA-RISC プロセッサー上の HP-UX オペレーティング・システムの場合、Java ストアード・プロシージャーおよびユーザー定義関数を実行するための特別な前提条件があります。

『 IBM Data Server Driver for JDBC and SQLJ のインストール』にある前提条件に加えて、以下の前提条件のステップを実行する必要があります。

1. コマンド行で以下のコマンドを実行して、db2hpjv ツールを使用可能にします。

   ```
   db2hpjv -e
   db2stop
   db2start
   ```

   db2hpjv を使用不可にする必要がある場合は、以下のコマンドを実行します。

   ```
   db2hpjv -d
   db2stop
   db2start
   ```

   db2hpjv -e を発行する前に、**必ず** SDK for Java をオペレーティング・システムにインストールしなければなりません。
2. HP-UX ランタイム・リンカーが Java 共用ライブラリーにアクセスできるようにします。

   Java ストアード・プロシージャーまたはユーザー定義関数を実行するには、HP-UX ランタイム・リンカーが特定の Java 共用ライブラリーにアクセスできなければならず、 DB2 システムはこれらのライブラリーおよび JVM をロードできなければなりません。そのロードを行うプログラムは、setuid 特権のもとで実行されるので、/usr/lib/pa20_64 内の従属ライブラリー**だけ** が探索されることになります。 Java 共用ライブラリーへのアクセスを作成するには、以下のメソッドの 1 つを選択します。

   - Java 共用ライブラリーへのシンボリック・リンクを作成します。これを実行するには、ルートとしてログインし、以下のコマンドを実行して Java 共用ライブラリーへのシンボリック・リンクを作成します。

     ```
     ln -s /opt/java1.4/jre/lib/PA_RISC2.0W/*.sl /usr/lib/pa20_64
     ln -s /opt/java1.4/jre/lib/PA_RISC2.0W/hotspot/*.sl /usr/lib/pa20_64
     ```

     これらのコマンドにより以下のライブラリーへのシンボリック・リンクが作成されます。

     ```
     /opt/java1.4/jre/lib/PA_RISC2.0W/libnet.sl
     /opt/java1.4/jre/lib/PA_RISC2.0W/libzip.sl
     /opt/java1.4/jre/lib/PA_RISC2.0W/librmi.sl
     /opt/java1.4/jre/lib/PA_RISC2.0W/libnio.sl
     /opt/java1.4/jre/lib/PA_RISC2.0W/libverify.sl
     /opt/java1.4/jre/lib/PA_RISC2.0W/libmlib_image.sl
     ```

```
/opt/java1.4/jre/lib/PA_RISC2.0W/libhprof.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjaas_unix.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libawt.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libcmm.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libdcpr.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libdt_socket.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libfontmanager.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libioser12.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libmawt.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjsound.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjava.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjawt.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjcov.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjcpm.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjdwp.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/libjpeg.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/hotspot/libjsig.sl
/opt/java1.4/jre/lib/PA_RISC2.0W/hotspot/libjvm.sl
```

- /opt/java1.4/jre/lib/PA_RISC2.0W ディレクトリーと opt/java1.4/jre/lib/PA_RISC2.0W/hotspot ディレクトリーを、/etc/dld.sl.conf ファイル**および** SHLIB_PATH 環境に追加します。

3. HP-UX ランタイム・リンカーが Java 共用ライブラリーにアクセスできるようにします。

   DB2 サーバーが、Java ルーチンの実行時に共用 Java ライブラリーを検出できない場合、-4300 エラーを生成します。

第 3 章 JDBC アプリケーション・プログラミング

JDBC アプリケーションの作成は、他の言語での SQL アプリケーションの作成と多くの点で共通しています。

一般に、以下のことを行う必要があります。
- JDBC メソッドが含まれている Java パッケージにアクセスする。
- DB2 表へのデータの送信用またはこの表からのデータの取得用の変数を宣言する。
- データ・ソースに接続する。
- SQL ステートメントを実行する。
- SQL エラーおよび警告を処理する。
- データ・ソースへの接続を切断する。

実行する必要のあるタスクは他の言語のものと同様ですが、これらのタスクを実行する方法は多少異なります。

簡単な JDBC アプリケーションの例

この簡単な JDBC アプリケーションでは、JDBC アプリケーションに含める必要がある基本要素を示しています。

図 1. 簡単な JDBC アプリケーション

```
import java.sql.*;                                                          1

public class EzJava
{
  public static void main(String[] args)
  {
    String urlPrefix = "jdbc:db2:";
    String url;
    String empNo;                                                           2
    Connection con;
    Statement stmt;
    ResultSet rs;

    System.out.println ("**** Enter class EzJava");

    // Check the that first argument has the correct form for the portion
    // of the URL that follows jdbc:db2:,
    // as described
    // in the Connecting to a data source using the DriverManager
    // interface with the IBM Data Server Driver for JDBC and SQLJ topic.
    // For example, for IBM Data Server Driver for
    // JDBC and SQLJ type 2 connectivity,
    // args[0] might be MVS1DB2M. For
    // type 4 connectivity, args[0] might
    // be //stlmvs1:10110/MVS1DB2M.

    if (args.length==0)
    {
      System.err.println ("Invalid value. First argument appended to "+
       "jdbc:db2: must specify a valid URL.");
```

```
  System.exit(1);
}
url = urlPrefix + args[0];

try
{
  // Load the driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                              3a
  System.out.println("**** Loaded the JDBC driver");

  // Create the connection using the IBM Data Server Driver for JDBC and SQLJ
  con = DriverManager.getConnection (url);                                 3b
  // Commit changes manually
  con.setAutoCommit(false);
  System.out.println("**** Created a JDBC connection to the data source");

  // Create the Statement
  stmt = con.createStatement();                                            4a
  System.out.println("**** Created JDBC Statement object");

  // Execute a query and generate a ResultSet instance
  rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                    4b
  System.out.println("**** Created JDBC ResultSet object");

  // Print all of the employee numbers to standard output device
  while (rs.next()) {
    empNo = rs.getString(1);
    System.out.println("Employee number = " + empNo);
  }
  System.out.println("**** Fetched all rows from JDBC ResultSet");
  // Close the ResultSet
  rs.close();
  System.out.println("**** Closed JDBC ResultSet");

  // Close the Statement
  stmt.close();
  System.out.println("**** Closed JDBC Statement");

  // Connection must be on a unit-of-work boundary to allow close
  con.commit();
  System.out.println ( "**** Transaction committed" );

  // Close the connection
  con.close();                                                             6
  System.out.println("**** Disconnected from data source");

  System.out.println("**** JDBC Exit from class EzJava - no errors");

}

catch (ClassNotFoundException e)
{
  System.err.println("Could not load JDBC driver");
  System.out.println("Exception: " + e);
  e.printStackTrace();
}

catch(SQLException ex)                                                     5
{
  System.err.println("SQLException information");
  while(ex!=null) {
    System.err.println ("Error msg: " + ex.getMessage());
    System.err.println ("SQLSTATE: " + ex.getSQLState());
    System.err.println ("Error code: " + ex.getErrorCode());
    ex.printStackTrace();
    ex = ex.getNextException(); // For drivers that support chained exceptions
```

```
      }
    }
  }  // End main
}    // End EzJava
```

29 ページの図 1 の注:

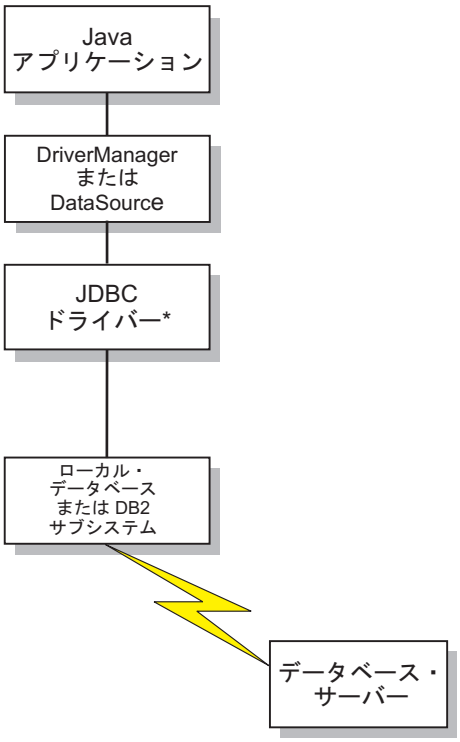
注	説明
1	このステートメントにより、JDBC コア API が含まれている java.sql パッケージがインポートされます。アクセスが必要になる可能性のある他の Java パッケージについては、『JDBC サポート用の Java パッケージ』を参照してください。
2	String 変数 empNo により、ホスト変数の機能が実行されます。つまり、この変数は SQL 照会から検索されたデータを保持するために使用されます。詳細については、『JDBC アプリケーションでの変数』を参照してください。
3a および 3b	これら 2 つのステートメントのセットでは、使用可能な 2 つのインターフェースのいずれかを使用して、データ・ソースに接続する方法が示されています。詳細については、『JDBC アプリケーションによるデータ・ソースへの接続の方法』を参照してください。 手順 3a (JDBC ドライバーのロード) は、JDBC 4.0 を使用する場合は不要です。
4a および 4b	これら 2 つのステートメントのセットでは、JDBC で SELECT を実行する方法が示されています。他の SQL 操作を実行する方法については、『SQL を実行するための JDBC インターフェース』を参照してください。
5	この try/catch ブロックでは、SQL エラー処理用の SQLException クラスの使用が示されています。 SQL エラーの処理の詳細については、『IBM Data Server Driver for JDBC and SQLJ 使用時の SQLException の処理』を参照してください。 SQL 警告の処理については、『IBM Data Server Driver for JDBC and SQLJ 使用時の SQLWarning の処理』を参照してください。
6	このステートメントにより、アプリケーションがデータ・ソースから切断されます。『JDBC アプリケーションでのデータ・ソースからの切断』を参照してください。

JDBC アプリケーションによるデータ・ソースへの接続の方法

SQL プログラムで SQL ステートメントを実行する前に、データ・ソースに接続する必要があります。

IBM Data Server Driver for JDBC and SQLJ では、タイプ 2 接続およびタイプ 4 接続がサポートされます。 DB2 データベースへの接続では、タイプ 2 またはタイプ 4 の接続を使用することができます。 IBM Informix Dynamic Server (IDS) データベースへの接続では、タイプ 4 の接続を使用することができます。

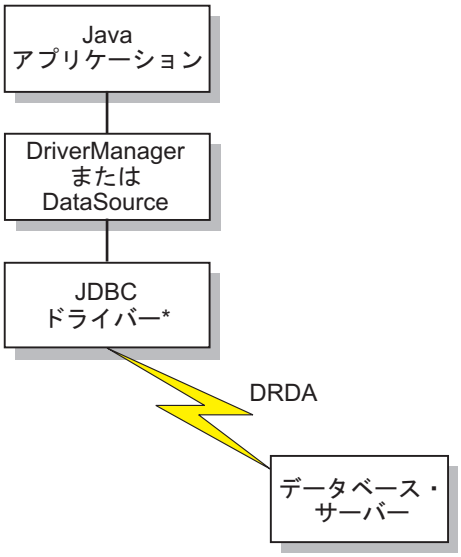
次の図では、Java アプリケーションによる IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用したデータ・ソースへの接続方法が示されています。



*JVM で実行される Java バイトコードと
ネイティブ・コード

図 2. IBM Data Server Driver for JDBC and SQLJ Type 2 接続での Java アプリケーションの流れ

次の図では、Java アプリケーションによる IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用したデータ・ソースへの接続方法が示されています。



*JVM で実行される Java バイトコード

図 3. IBM Data Server Driver for JDBC and SQLJ Type 4 接続での Java アプリケーションの流れ

DB2 アプリケーションによる DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続

JDBC アプリケーションでは、java.sql パッケージの一部である JDBC DriverManager インターフェースを使用して、データ・ソースへの接続を確立することができます。

Java アプリケーションでは、最初に Class.forName メソッドが呼び出されて、JDBC ドライバーがロードされます。アプリケーションによりドライバーがロードされた後、DriverManager.getConnection メソッドが呼び出されて、データ・ソースに接続されます。

Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の場合、次の引数で Class.forName メソッドを呼び出して、このドライバーをロードすることができます。

```
COM.ibm.db2.jdbc.app.DB2Driver
```

以下のコードでは、DB2 JDBC Type 2 ドライバーのロードが示されています。

```
try {
  // Load the DB2 JDBC Type 2 Driver with DriverManager
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

catch ブロックは、ドライバーが見つからなかった場合のエラーの出力に使用されます。

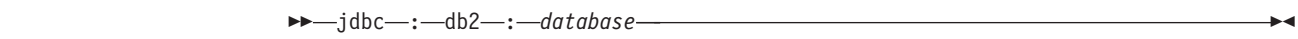
ドライバーのロード後、DriverManager.getConnection メソッドを呼び出して、データ・ソースに接続します。以下のいずれかの形式の getConnection を使用できます。

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

url 引数により、データ・ソースが表されます。

DB2 JDBC Type 2 ドライバーの場合、次の形式の URL を指定します。

DB2 JDBC Type 2 ドライバーの URL の構文



URL の各部の意味は、以下のとおりです。

jdbc:db2:
jdbc:db2: は、接続が DB2 データ・ソースに対するものであることを示します。

database
データベース別名。この別名は、 DB2 クライアント上の DB2 のデータベース・カタログ項目を指します。

info 引数は、接続用の一連のドライバー・プロパティーを含んだタイプ java.util.Properties のオブジェクトです。*info* 引数の指定は、URL での *property=value* ストリングの指定の代わりとなります。

接続用のユーザー ID およびパスワードの指定: 接続用のユーザー ID およびパスワードを指定するには、次の方法があります。

- *user* および *password* を指定する getConnection メソッドの形式を使用します。
- java.util.Properties オブジェクトで user および password プロパティーを設定してから、*info* を指定する getConnection メソッドの形式を使用します。

例: user および password パラメーターへのユーザー ID およびパスワードの設定

```
String url = "jdbc:db2:toronto";
                                        // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Create connection
```

例: java.util.Properties オブジェクトへのユーザー ID およびパスワードの設定

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2:toronto";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                        // Create connection
```

DriverManager インターフェースと IBM Data Server Driver for JDBC and SQLJ を使用したデータ・ソースへの接続

JDBC アプリケーションでは、java.sql パッケージの一部である JDBC DriverManager インターフェースを使用して、データ・ソースへの接続を確立することができます。

接続を確立するための手順は、以下のとおりです。

1. Class.forName メソッドを呼び出して、JDBC ドライバーをロードします。

   JDBC 4.0 を使用している場合は、JDBC ドライバーを明示的にロードする必要はありません。

   IBM Data Server Driver for JDBC and SQLJ では、Class.forName メソッドを呼び出すときに以下の引数を指定してドライバーをロードします。

   ```
   com.ibm.db2.jcc.DB2Driver
   ```

   以前の JDBC ドライバーとの互換性を維持するために、代わりに以下の引数を使用することもできます。

   ```
   COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
   ```

IBM Data Server Driver for JDBC and SQLJ をロードするコードを以下に示します。

```
try {
  // Load the IBM Data Server Driver for JDBC and SQLJ with DriverManager
  Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

catch ブロックは、ドライバーが見つからなかった場合のエラーの出力に使用されます。

2. DriverManager.getConnection メソッドを呼び出して、データ・ソースに接続します。

   以下のいずれかの形式の getConnection を使用できます。

   ```
   getConnection(String url);
   getConnection(String url, user, password);
   getConnection(String url, java.util.Properties info);
   ```

   IBM Data Server Driver for JDBC and SQLJ Type 4 接続では、パラメーターまたはプロパティー値によって getConnection メソッドにユーザー ID とパスワードを指定する必要があります。

   url 引数はデータ・ソースに対応しており、どのタイプの JDBC 接続を使用するのかをこの引数で示します。

   info 引数は、接続用の一連のドライバー・プロパティーを含んだタイプ java.util.Properties のオブジェクトです。*info* 引数を指定すれば、URL で *property=value;* ストリングを指定する代わりになります。指定できるプロパティーについては、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』を参照してください。

   接続用のユーザー ID とパスワードを指定するには、いくつかの方法があります。

   - *url* に property=*value*; 節を指定した形式の getConnection メソッドを使用して、URL に user と password のプロパティーを組み込みます。
   - *user* および *password* を指定する getConnection メソッドの形式を使用します。
   - java.util.Properties オブジェクトで user および password プロパティーを設定してから、*info* を指定する getConnection メソッドの形式を使用します。

例: 接続を確立し、URL でユーザー ID とパスワードを設定する:

```
String url = "jdbc:db2://myhost:5021/mydb:" +
  "user=dbadm;password=dbadm;";

                                        // Set URL for data source
Connection con = DriverManager.getConnection(url);
                                        // Create connection
```

例: 接続を確立し、user と password のパラメーターでユーザー ID とパスワードを設定する:

```
String url = "jdbc:db2://myhost:5021/mydb";
                                        // Set URL for data source
String user = "dbadm";
String password = "dbadm";
Connection con = DriverManager.getConnection(url, user, password);
                                        // Create connection
```

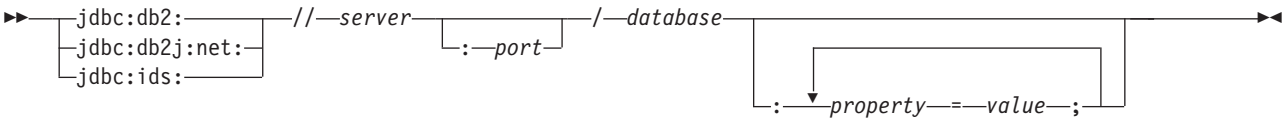
例: 接続を確立し、java.util.Properties オブジェクトでユーザー ID とパスワードを設定する:

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "dbadm");         // Set user ID for connection
properties.put("password", "dbadm");     // Set password for connection
String url = "jdbc:db2://myhost:5021/mydb";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                        // Create connection
```

IBM Data Server Driver for JDBC and SQLJ Type 4 接続の URL 形式

JDBC アプリケーションでタイプ 4 接続を使用しており、DriverManager インターフェースを使用して接続を行っている場合、DriverManager.getConnection 呼び出しでタイプ 4 接続を示す URL を指定する必要があります。

IBM Data Server Driver for JDBC and SQLJ Type 4 接続 URL 構文



IBM Data Server Driver for JDBC and SQLJ Type 4 接続 URL オプションの説明

URL の各部の意味は、以下のとおりです。

jdbc:db2: または jdbc:db2j:net:
URL の最初の部分の意味は、以下のとおりです。

jdbc:db2:
接続先が DB2 for z/OS、 DB2 Database for Linux, UNIX, and Windows であることを示します。

また、アプリケーションの移植性のために、jdbc:db2: も IBM Informix Dynamic Server (IDS) データベースへの接続に使用できます。

jdbc:db2j:net:
接続先がリモートの IBM Cloudscape サーバーであることを示します。

jdbc:ids:
接続先が IDS データ・ソースであることを示します。jdbc:informix-sqli: も接続先が IDS データ・ソースであることを示しますが、jdbc:ids: を使用してください。

server
データ・ソースのドメイン名または IP アドレス。

port
データ・ソースに割り当てられている TCP/IP サーバーのポート番号。これは 0 から 65535 の間の整数です。デフォルトは 446 です。

database
データ・ソースの名前。

- 接続を DB2 for z/OS サーバーに対して行う場合、*database* は、インストール中に定義される DB2 ロケーション名です。DB2 ロケーション名の中の文字はすべて大文字でなければなりません。 IBM Data Server Driver for JDBC and SQLJ は、データベース値の小文字を IBM Data Server Driver for JDBC and SQLJ Type 4 接続用の大文字に変換しません。

  サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- 接続を DB2 for z/OS サーバーまたは DB2 for i サーバーに対して行う場合、*database* 内のすべての文字を大文字にしなければなりません。
- 接続を DB2 Database for Linux, UNIX, and Windows サーバーに対して行う場合、*database* は、インストール中に定義されるデータベース名です。
- 接続を IDS サーバーに対して行う場合、*database* は、データベース名です。この名前は大/小文字を区別しません。サーバーにより、名前は小文字に変換されます。
- 接続を IBM Cloudscape サーバーに対して行う場合、*database* は、データベースを含むファイルの完全修飾名です。この名前は、二重引用符 (″) で囲まなければなりません。以下に例を示します。

  ```
  "c:/databases/testdb"
  ```

property=*value*;
JDBC 接続のプロパティーとその値。1 つ以上のプロパティーと値の対を指定できます。プロパティーと値の各対 (最後のものを含む) はセミコロン (;) で終わらなければなりません。プロパティーと値のストリングのリスト内のどこにも、スペースまたはその他の空白文字を組み込まないでください。

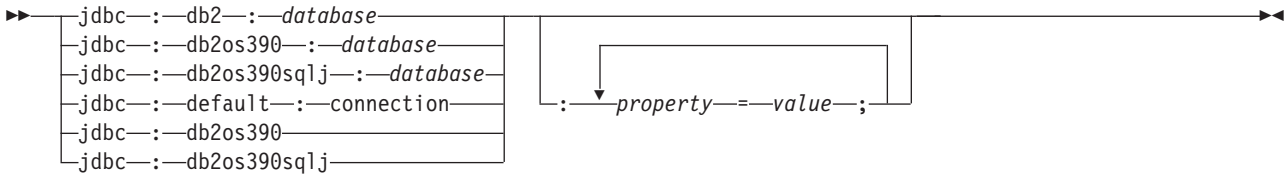
int データ・タイプのプロパティーには、事前定義定数フィールド値を持つものがあります。定数フィールド値は、*url* パラメーターで使用する前に、整数値に解決する必要があります。例えば、
com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL を *url* パラメーターに使用することはできません。しかし、com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL を含む URL ストリングを作成して、その URL ストリングをストリング変数に割り当てることができます。その後、以下のようにして、そのストリング変数を *url* パラメーターで使用できます。

```
String url =
 "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
 "user=dbadm;password=dbadm;" +
 "traceLevel=" +
 (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";";
 Connection con =
   java.sql.DriverManager.getConnection(url);
```

IBM Data Server Driver for JDBC and SQLJ Type 2 接続の URL 形式

JDBC アプリケーションでタイプ 2 接続を使用しており、DriverManager インターフェースによって接続を行っている場合、DriverManager.getConnection 呼び出し内の URL がタイプ 2 接続を示すことを指定する必要があります。

IBM Data Server Driver for JDBC and SQLJ Type 2 接続 URL 構文



IBM Data Server Driver for JDBC and SQLJ Type 2 接続 URL オプションの説明

URL の各部の意味は、以下のとおりです。

jdbc:db2: または jdbc:db2os390: または jdbc:db2os390sqlj: または jdbc:default:connection

URL の最初の部分の意味は、以下のとおりです。

jdbc:db2: または jdbc:db2os390: または jdbc:db2os390sqlj:

DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows サーバーへの接続を示します。 jdbc:db2os390: および jdbc:db2os390sqlj: は、旧式のドライバー用に作成されたプログラムの互換性を目的としており、 DB2 for z/OS への IBM Data Server Driver for JDBC and SQLJ Type 2 接続 にのみ適用されます。

jdbc:default:connection

URL が、CICS®、IMS™、または Java ストアード・プロシージャー環境によって制御される DB2 スレッドを介したローカル・サブシステムへの接続のためのものであることを示します。

database

データベース・サーバーの名前。

- serverName 接続プロパティーの値がヌルの場合、*database* は、インストール中に定義されるデータベース名になります。 serverName プロパティーの値がヌルではない場合、*database* はデータベース別名になります。

***property*=*value*;**

JDBC 接続のプロパティーとその値。1 つ以上のプロパティーと値の対を指定できます。プロパティーと値の各対 (最後のものを含む) はセミコロン (;) で終わらなければなりません。プロパティーと値のストリングのリスト内のどこにも、スペースまたはその他の空白文字を組み込まないでください。

int データ・タイプのプロパティーには、事前定義定数フィールド値を持つものがあります。定数フィールド値は、*url* パラメーターで使用する前に、整数値に解決する必要があります。例えば、
com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL を *url* パラメーターに使用することはできません。しかし、com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL

を含む URL ストリングを作成して、その URL ストリングをストリング変数に割り当てることができます。その後、以下のようにして、そのストリング変数を *url* パラメーターで使用できます。

```
String url =
 "jdbc:db2:STLEC1" +
 "user=dbadm;password=dbadm;" +
 "traceLevel=" +
 (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";";
 Connection con =
   java.sql.DriverManager.getConnection(url);
```

DataSource インターフェースを使用したデータ・ソースへの接続

さまざまなデータ・ソースで移植可能なアプリケーションを作成する必要がある場合は、DataSource インターフェースを使用します。

DriverManager を使用してデータ・ソースに接続すると、アプリケーションは、特定の JDBC ドライバー・クラス名とドライバー URL を識別しなければならなくなるので、移植性が低下します。ドライバー・クラス名とドライバー URL は、JDBC ベンダー、ドライバーの実装、データ・ソースによって異なります。

DataSource インターフェースによってデータ・ソースに接続する場合は、DataSource オブジェクトを使用します。

DataSource オブジェクトを使用するための最も簡単な方法は、DriverManager インターフェースの場合のように、オブジェクトの作成と使用を同じアプリケーションで実行することです。ただし、この方法では移植性が得られません。

DataSource オブジェクトを使用する最善の方法は、システム管理者が WebSphere Application Server などのツールを使用して、オブジェクトの作成と管理を別途行うようにすることです。DataSource オブジェクトを作成して管理するプログラムでは、Java Naming and Directory Interface (JNDI) も使用して、論理名を DataSource オブジェクトに割り当てます。DataSource オブジェクトを使用する JDBC アプリケーションでは、その論理名でオブジェクトを参照できるので、基礎になっているデータ・ソースの情報が不要になります。さらに、システム管理者がデータ・ソースの属性を変更する場合でも、アプリケーション・プログラムを使用する側でアプリケーション・プログラムを変更する必要はありません。

WebSphere を使用して DataSource オブジェクトをデプロイする方法の詳細については、Web で以下の URL を参照してください。

`http://www.ibm.com/software/webservers/appserv/`

DataSource オブジェクトを自分でデプロイする方法については、『DataSource オブジェクトの作成およびデプロイ』を参照してください。

DataSource インターフェースと DriverManager インターフェースを同じアプリケーションで使用することは可能ですが、移植性を最大限に高めるために、DataSource インターフェースだけを使用して接続を取得することをお勧めします。

システム管理者が既に作成して論理名を割り当てた DataSource オブジェクトを使用して接続を取得するには、以下の手順を実行します。

1. システム管理者から、接続する必要があるデータ・ソースの論理名を取得します。
2. 次の手順で使用する Context オブジェクトを作成します。 Context インターフェースは、JDBC ではなく Java Naming and Directory Interface (JNDI) に組み込まれています。
3. アプリケーション・プログラムで JNDI を使用して、論理データ・ソース名に関連した DataSource オブジェクトを取得します。
4. DataSource.getConnection メソッドを使用して接続を取得します。

   以下のいずれかの形式の getConnection メソッドを使用できます。

   ```
   getConnection();
   getConnection(String user, String password);
   ```

   接続のユーザー ID とパスワードとして、DataSource のデプロイ時に指定されているユーザー ID とパスワード以外の値を指定する必要がある場合は、2 番目の形式を使用します。

システム管理者が作成した DataSource オブジェクトを使用して接続を取得する例:
この例で接続する必要があるデータ・ソースの論理名は、jdbc/sampledb です。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();                          2
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");     3
Connection con=ds.getConnection();                         4
```

図 4. DataSource オブジェクトによる接続の取得

DataSource オブジェクトの作成と使用を同じアプリケーションで実行する例:

図 5. DataSource オブジェクトの作成と使用を同じアプリケーションで実行する

```
import java.sql.*;        // JDBC base
import javax.sql.*;       // Addtional methods for JDBC
import com.ibm.db2.jcc.*; // IBM Data Server Driver for JDBC and SQLJ   1
                          // interfaces
DB2SimpleDataSource dbds=new DB2SimpleDataSource();            2
dbds.setDatabaseName("dbloc1");                                3
                          // Assign the location name
dbds.setDescription("Our Sample Database");
                          // Description for documentation
dbds.setUser("john");
                          // Assign the user ID
dbds.setPassword("dbadm");
                          // Assign the password
Connection con=dbds.getConnection();                           4
                          // Create a Connection object
```

注	説明
1	DataSource インターフェースの実装が含まれているパッケージをインポートします。

注	説明
2	DB2SimpleDataSource オブジェクトを作成します。DB2SimpleDataSource は、DataSource インターフェースの IBM Data Server Driver for JDBC and SQLJ 実装の 1 つです。DB2 の DataSource 実装については、『DataSource オブジェクトの作成およびデプロイ』を参照してください。
3	setDatabaseName メソッド、setDescription メソッド、setUser メソッド、setPassword メソッドによって、DB2SimpleDataSource オブジェクトに属性を割り当てます。IBM Data Server Driver for JDBC and SQLJ 使用時に DB2SimpleDataSource オブジェクトに設定できる属性については、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』を参照してください。
4	DB2SimpleDataSource オブジェクト dbds に対応するデータ・ソースへの接続を確立します。

使用する IBM Data Server Driver for JDBC and SQLJ 接続のタイプを判別する方法

IBM Data Server Driver for JDBC and SQLJ は、タイプ 2 接続とタイプ 4 接続という 2 種類のタイプの接続をサポートしています。

DriverManager インターフェースの場合、DriverManager.getConnection メソッドの URL を介して接続のタイプを指定します。 DataSource インターフェースの場合、`driverType` プロパティーを介して接続のタイプを指定します。

次の表は、タイプ 2 接続とタイプ 4 接続の違いを要約しています。

表 6. IBM Data Server Driver for JDBC and SQLJ Type 2 接続と IBM Data Server Driver for JDBC and SQLJ Type 4 接続の比較

関数	IBM Data Server Driver for JDBC and SQLJ Type 2 接続のサポート	IBM Data Server Driver for JDBC and SQLJ Type 4 接続のサポート
Sysplex ワークロード・バランシングおよび接続コンセントレーター	DB2 Connect によるサポート	単一の JVM 内の接続用ドライバーによる直接サポート 複数の JVM にわたる DB2 Connect を介したサポート
通信プロトコル	TCP/IP	TCP/IP
パフォーマンス	ローカル DB2 サーバーへのアクセスに適している	リモート DB2 サーバーへのアクセスに適している
インストール	Java クラスに加えてネイティブ・ライブラリーのインストールが必要	Java クラスのみのインストールが必要
ストアード・プロシージャー	ストアード・プロシージャーの呼び出しまたは実行に使用できる	ストアード・プロシージャーの呼び出しにのみ使用できる
分散トランザクション処理 (XA)	サポートされる	サポートされる
J2EE 1.4 準拠	準拠	準拠

以下の点は、使用する接続のタイプを判別するのに役立ちます。

以下の環境では、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用します。

- JDBC または SQLJ アプリケーションは大抵の場合ローカルで実行される。

ローカル・アプリケーションでは、タイプ 2 接続を使用するとパフォーマンスが向上します。

- Java ストアード・プロシージャーを実行している。

  ストアード・プロシージャー環境は 2 つの部分から成ります。1 つはクライアント・プログラムで、そこからストアード・プロシージャーを呼び出します。もう 1 つはサーバー・プログラムで、これがストアード・プロシージャーです。ストアード・プロシージャーは、タイプ 2 またはタイプ 4 接続を使用する JDBC または SQLJ プログラムで呼び出すことができますが、Java ストアード・プロシージャーはタイプ 2 接続を使用して実行する必要があります。

以下の環境では、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用します。

- JDBC または SQLJ アプリケーションは大抵の場合リモートで実行される。

  リモート・アプリケーションでは、タイプ 4 接続を使用するとパフォーマンスが向上します。

- IBM Data Server Driver for JDBC and SQLJ 接続コンセントレーターおよび Sysplex ワークロード・バランシング・サポートを使用している。

JDBC 接続オブジェクト

いずれかの接続メソッドによってデータ・ソースに接続する場合、データ・ソースへの接続を表す Connection オブジェクトを作成します。

この Connection オブジェクトを使用して、以下の事柄を行います。

- SQL ステートメントを実行するための Statement、PreparedStatement、および CallableStatement オブジェクトを作成します。このことについては、『JDBC アプリケーションでの SQL ステートメントの実行』で解説されています。
- 接続先のデータ・ソースに関する情報を収集します。このプロセスについては、『DatabaseMetaData メソッドを使用した、データ・ソースに関する情報の取得』で解説されています。
- トランザクションをコミットまたはロールバックします。トランザクションのコミットは手動でも自動でも行えます。これらの操作については、『JDBC トランザクションのコミットまたはロールバック』で解説されています。
- データ・ソースへの接続をクローズします。この操作については、『JDBC アプリケーションでのデータ・ソースからの切断』で解説されています。

DataSource オブジェクトの作成およびデプロイ

JDBC バージョン 2.0 以降には、データ・ソースに接続するための DataSource インターフェースが用意されています。この DataSource インターフェースを使用することが、データ・ソースに接続するための好ましい方法です。

DataSource インターフェースを使用するには、2 部からなる手順が必要です。

- DataSource オブジェクトの作成およびデプロイ。この手順は通常、システム管理者が WebSphere Application Server などのツールを使用して実行します。
- DataSource オブジェクトによる接続の作成。この手順は、アプリケーション・プログラムで実行します。

このトピックでは、DataSource オブジェクトを自分で作成してデプロイするときに必要な情報を取り上げます。

IBM Data Server Driver for JDBC and SQLJ には、以下の DataSource 実装が用意されています。

- com.ibm.db2.jcc.DB2SimpleDataSource。接続プールはサポートしていません。この実装は、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続または IBM Data Server Driver for JDBC and SQLJ Type 4 接続で使用できます。
- com.ibm.db2.jcc.DB2ConnectionPoolDataSource。接続プールをサポートしています。この実装は、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続または IBM Data Server Driver for JDBC and SQLJ Type 4 接続で使用できます。
- com.ibm.db2.jcc.DB2XADataSource。接続プールと分散トランザクションをサポートしています。接続プールは、WebSphere Application Server や他のアプリケーション・サーバーによって用意します。この実装は、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続でのみ使用できます。

DB2 JDBC Type 2 ドライバーには、以下の DataSource 実装が用意されています。

- COM.ibm.db2.jdbc.DB2DataSource。接続プールに対応しています。この実装の場合、接続プールは内部で処理され、アプリケーションからは認識されません。
- COM.ibm.db2.jdbc.DB2XADataSource。分散トランザクションと接続プールの組み込みサポートはありません。この実装の場合は、分散トランザクションと接続プールを自分で管理する必要があります。そのためには、独自のコードを記述するか、WebSphere Application Server などのツールを使用します。

DataSource オブジェクトを作成してデプロイするときには、以下のタスクを実行する必要があります。

1. 適切な DataSource 実装のインスタンスを作成します。
2. DataSource オブジェクトのプロパティーを設定します。
3. そのオブジェクトを Java Naming and Directory Interface (JNDI) ネーム・サービスに登録します。

これらのタスクを実行する例を以下に示します。

```
import java.sql.*;        // JDBC base
import javax.naming.*;    // JNDI Naming Services
import javax.sql.*;       // Additional methods for JDBC
import com.ibm.db2.jcc.*; // IBM Data Server Driver for
                          // JDBC and SQLJ
                          // implementation of JDBC
                          // standard extension APIs

DB2SimpleDataSource dbds = new com.ibm.db2.jcc.DB2SimpleDataSource();   1

dbds.setDatabaseName("db2loc1");                                        2
dbds.setDescription("Our Sample Database");
dbds.setUser("john");
dbds.setPassword("mypw");
...
Context ctx=new InitialContext();                                       3
Ctx.bind("jdbc/sampledb",dbds);                                         4
```

図 6. DataSource オブジェクトを作成してデプロイする例

注	説明
1	DB2SimpleDataSource クラスのインスタンスを作成します。
2	このステートメントとその後の 3 つのステートメントによって、この DB2SimpleDataSource オブジェクトのプロパティーの値を設定します。
3	JNDI で使用するコンテキストを作成します。
4	DBSimple2DataSource オブジェクト dbds を論理名 jdbc/sampledb に関連付けます。このオブジェクトを使用するアプリケーションは、その名前 jdbc/sampledb でオブジェクトを参照できます。

JDBC サポート用の Java パッケージ

JDBC メソッドを呼び出すには、それらのメソッドを含む様々な Java パッケージのすべてまたは一部にあらかじめアクセスできるようにしておく必要があります。

パッケージまたは特定のクラスをインポートするか、または完全修飾クラス名を使用することにより、これを行えます。 JDBC プログラムについて以下のパッケージまたはクラスが必要になることがあります。

java.sql
コア JDBC API が含まれます。

javax.naming
DataSource をインプリメントするためにしばしば使用される Java Naming and Directory Interface (JNDI) のクラスおよびインターフェースが含まれています。

javax.sql
Java を使用してサーバー・サイドのアプリケーションを作成するためのメソッドが含まれています。

javax.transaction
Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) 用の分散トランザクションの JDBC サポートが含まれています。

com.ibm.db2.jcc
IBM Data Server Driver for JDBC and SQLJ 用の JDBC のインプリメンテーションが含まれています。

COM.ibm.db2.jdbc
DB2 JDBC Type 2 ドライバー用の JDBC のインプリメンテーションが含まれています。

DatabaseMetaData メソッドを使用した、データ・ソースに関する情報の取得

DatabaseMetaData インターフェースには、データ・ソースの情報を取得するためのメソッドが含まれています。それらのメソッドは、さまざまなデータ・ソースにアクセスできる汎用アプリケーションを作成するときに便利です。

さまざまなデータ・ソースにアクセスできる汎用アプリケーションを実行する場合は、まずデータ・ソースがさまざまなデータベース操作を処理できるかどうかをテ

ストする必要があります。例えば、JDBC 3.0 のメソッドをデータ・ソースのドライバーに対して呼び出すのであれば、まずそのドライバーが JDBC 3.0 のレベルかどうかを確認する必要があります。

DatabaseMetaData メソッドでは、以下のタイプの情報を確認できます。
- データ・ソースがサポートしているフィーチャー (ANSI SQL レベルなど)
- JDBC ドライバーに関する具体的な情報 (ドライバーのレベルなど)
- 制限事項 (索引に組み込める列の最大数など)
- データ・ソースがデータ定義ステートメント (CREATE、ALTER、DROP、GRANT、REVOKE) をサポートしているかどうか
- データ・ソースのオブジェクト・リスト (表、索引、プロシージャーなど)
- データ・ソースがさまざまな JDBC 機能 (バッチ更新、スクロール可能 ResultSet など) をサポートしているかどうか
- ドライバーがサポートしているスカラー関数のリスト

DatabaseMetaData メソッドを呼び出すには、以下の基本手順を実行する必要があります。

1. 接続に対して getMetaData メソッドを呼び出して、DatabaseMetaData オブジェクトを作成します。
2. DatabaseMetaData メソッドを呼び出して、データ・ソースに関する情報を取得します。
3. メソッドから ResultSet が返された場合は、以下のようにします。
   a. ループの中で、next メソッドを使用してカーソルを配置し、getXXX メソッドを使用して、ResultSet オブジェクトの現在行の各列からデータを取得します。
   b. close メソッドを呼び出して ResultSet オブジェクトをクローズします。

例: DatabaseMetaData メソッドを使用して、ドライバーのバージョンを確認し、データ・ソースで使用できるストアード・プロシージャーのリストを取得し、ドライバーがサポートしている日時関数のリストを取得するためのコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

図 7. DatabaseMetaData メソッドによるデータ・ソース情報の取得

```
Connection con;
DatabaseMetaData dbmtadta;
ResultSet rs;
int mtadtaint;
String procSchema;
String procName;
String dtfnList;
...
dbmtadta = con.getMetaData();     // Create the DatabaseMetaData object  1
mtadtaint = dmtadta.getDriverVersion();                                  2
                                  // Check the driver version
System.out.println("Driver version: " + mtadtaint);
rs = dbmtadta.getProcedures(null, null, "%");
                                  // Get information for all procedures
while (rs.next()) {               // Position the cursor                 3a
 procSchema = rs.getString("PROCEDURE_SCHEM");
                                  // Get procedure schema
 procName = rs.getString("PROCEDURE_NAME");
```

```
                                    // Get procedure name
 System.out.println(procSchema + "." + procName);
                                    // Print the qualified procedure name
}
dtfnList = dbmtadta.getTimeDateFunctions();
                                    // Get list of supported datetime functions
System.out.println("Supported datetime functions:");
System.out.println(dtfnList);       // Print the list of datetime functions
rs.close();                         // Close the ResultSet                3b
```

データ・ソースのタイプを識別するための DatabaseMetaData メソッド

DatabaseMetaData.getDatabaseProductName および DatabaseMetaData.getProductVersion メソッドを使用して、接続先のデータベース・マネージャーのタイプとレベル、およびデータベース・マネージャーが稼働しているオペレーティング・システムを識別することができます。

DatabaseMetaData.getDatabaseProductName は、データベース・マネージャーとオペレーティング・システムを識別するストリングを戻します。ストリングは、以下の形式のいずれかです。

database-productdatabase-product/operating-system

以下の表では、DatabaseMetaData.getDatabaseProductName によって戻される値の例を示します。

表 7. DatabaseMetaData.getDatabaseProductName の値の例

getDatabaseProductName の値	**データベース製品**
DB2	DB2 for z/OS
DB2/LINUXX8664	x86 版 Linux 上の DB2 Database for Linux, UNIX, and Windows
IDS/UNIX64	UNIX 上の IBM Informix Dynamic Server (IDS)

DatabaseMetaData.getDatabaseVersionName は、データベース製品の標識と、データ・ソースのバージョン番号、リリース番号、およびメンテナンス・レベルを含むストリングを戻します。

以下の表では、DatabaseMetaData.getDatabaseProductVersion によって戻される値の例を示します。

表 8. DatabaseMetaData.getDatabaseProductVersion の値の例

getDatabaseProductVersion の値	**データベース製品のバージョン**
DSN09015	DB2 for z/OS バージョン 9.1 (新関数モード)
SQL09010	DB2 Database for Linux, UNIX, and Windows バージョン 9.1
IFX11100	IDS バージョン 11.10

JDBC アプリケーションでの変数

他の Java アプリケーションと同様に、JDBC アプリケーションを作成するときには変数を宣言します。 Java アプリケーションでは、そのような変数は Java ID として知られます。

これらの ID のいくつかは他の言語のホスト変数と同じ関数を持っています。これはデータベース表に受け渡すデータや、データベース表から取り出すデータを保持します。以下のコードの ID empNo は EMPNO 表列から取り出すデータを保持します。このデータのデータ・タイプは CHAR です。

```
String empNo;
// Execute a query and generate a ResultSet instance
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");
while (rs.next()) {
 String empNo = rs.getString(1);
 System.out.println("Employee number = " + empNo);
}
```

Java データ・タイプを選択すると、パフォーマンスに影響を与えることがあります。なぜなら、Java 変数のデータ・タイプが DB2 データ・タイプに緊密にマップされている場合、 DB2 は最適なアクセス・パスを選択するからです。

SQL を実行するための JDBC インターフェース

従来の SQL プログラムでは、SQL ステートメントを実行して、表のデータの更新、表からのデータの取り出し、あるいはストアード・プロシージャーの呼び出しを行います。 JDBC プログラムで同じ機能を実行するには、メソッドを呼び出します。

これらのメソッドは以下のインターフェースで定義されます。

- Statement インターフェースはすべての SQL ステートメントの実行をサポートします。以下のインターフェースは Statement インターフェースからメソッドを継承します。
  - PreparedStatement インターフェースは、入力パラメーター・マーカーを含む SQL ステートメントをサポートします。パラメーター・マーカーは入力変数を表します。また、PreparedStatement インターフェースはパラメーター・マーカーがない SQL ステートメントにも使用できます。

    IBM Data Server Driver for JDBC and SQLJ では、PreparedStatement インターフェースは、入力パラメーターを持っていても出力パラメーターを持たないために結果セットを戻さないストアード・プロシージャーを呼び出すために使用できます。ただし、推奨されるインターフェースは CallableStatement です。
  - CallableStatement インターフェースは、ストアード・プロシージャーの呼び出しをサポートします。

    CallableStatement インターフェースは、入力パラメーター、出力パラメーター、入出力パラメーターのいずれかを持つか、またはパラメーターを持たないストアード・プロシージャーを呼び出すために使用できます。 IBM Data Server Driver for JDBC and SQLJ では、Statement インターフェースを使用してストアード・プロシージャーを呼び出すこともできますが、そのストアード・プロシージャーにはパラメーターは指定できません。

- ResultSet インターフェースは、照会が生成する結果にアクセスできるようにします。 ResultSet インターフェースには、他の言語の SQL アプリケーションで使用されるカーソルと同じ目的があります。

Statement.executeUpdate メソッドによるデータベース・オブジェクトの作成と変更

JDBC メソッドの 1 つである Statement.executeUpdate を使用して、表を更新したり、ストアード・プロシージャーを呼び出したりできます。

Statement.executeUpdate メソッドを使用して、以下の操作を実行できます。

- データ定義ステートメント (CREATE、ALTER、DROP、GRANT、REVOKE など) の実行
- パラメーター・マーカーのない INSERT、UPDATE、DELETE、MERGE の各ステートメントの実行
- IBM Data Server Driver for JDBC and SQLJ では、CALL ステートメントを実行して、パラメーターがなく結果セットを返さないストアード・プロシージャーを呼び出せます。

これらの SQL ステートメントを実行するには、以下の手順を実行する必要があります。

1. Connection.createStatement メソッドを呼び出して、Statement オブジェクトを作成します。
2. Statement.executeUpdate メソッドを呼び出して、SQL 操作を実行します。
3. Statement.close メソッドを呼び出して、Statement オブジェクトを閉じます。

以下の SQL ステートメントを実行するとしましょう。

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

Statement オブジェクト `stmt` を作成し、その UPDATE ステートメントを実行し、更新された行の数を `numUpd` で返すコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
Connection con;
Statement stmt;
int numUpd;
...
stmt = con.createStatement();                // Create a Statement object  1
numUpd = stmt.executeUpdate(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");              2
                                             // Perform the update
stmt.close();                                // Close Statement object     3
```

図 8. Statement.executeUpdate の使用

PreparedStatement.executeUpdate メソッドを使用した表のデータの更新

DB2 表を定数値で更新する場合は、Statement.executeUpdate メソッドを使用できます。ところが、更新時には、変数の値を DB2 表に渡さなければならない場合が多くあります。その場合に使用できるのが、PreparedStatement.executeUpdate メソッドです。

入力パラメーターがあり、出力パラメーターがなく、結果セットを返さないストアード・プロシージャーであれば、 IBM Data Server Driver for JDBC and SQLJ で PreparedStatement.executeUpdate を使用して、そのようなストアード・プロシージャーを呼び出すこともできます。

DB2 for z/OS は、CALL ステートメントの動的実行をサポートしていません。DB2 for z/OS データ・ソースに格納されているストアード・プロシージャーの呼び出しでは、パラメーターとしてパラメーター・マーカーやリテラルを使用できますが、式は使用できません。以下のタイプのリテラルがサポートされています。

- Integer
- Double
- Decimal
- Character
- Hexadecimal
- Graphic

IBM Informix Dynamic Server データ・ソースに格納されているストアード・プロシージャーの呼び出しでは、PreparedStatement オブジェクトとして CALL ステートメントまたは EXECUTE PROCEDURE ステートメントを使用できます。

1 つの SQL ステートメントを何度も実行する場合は、 SQL ステートメントを PreparedStatement として作成する方がパフォーマンスが良くなります。

例えば、以下の UPDATE ステートメントでは、従業員表の 1 つの電話番号と 1 つの従業員番号だけを更新します。

```
UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'
```

この操作を一般化して、従業員表の電話番号と従業員番号のあらゆる組み合わせを更新するとしましょう。その場合は、定数の電話番号と従業員番号を変数に置き換える必要があります。

```
UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?
```

この形式の変数のことをパラメーター・マーカーといいます。パラメーター・マーカーを指定した SQL ステートメントを実行するには、以下の手順を実行する必要があります。

1. Connection.prepareStatement メソッドを呼び出して、PreparedStatement オブジェクトを作成します。
2. PreparedStatement.setXXX メソッドを呼び出して、入力変数に値を渡します。

このステップでは、標準パラメーター・マーカーを使用することを想定しています。別の方法として名前付きパラメーター・マーカーを使用する場合、 IBM Data Server Driver for JDBC and SQLJ のみのメソッドを使用して、値を入力パラメーターに渡します。

3. PreparedStatement.executeUpdate メソッドを呼び出して、変数値で表を更新します。
4. PreparedStatement オブジェクトの作業が完了した時点で、PreparedStatement.close メソッドを呼び出してそのオブジェクトを閉じます。

これまでに取り上げた手順を実行して、従業員番号「000010」の従業員の電話番号を「4657」に変更するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                               // Create a PreparedStatement object       1
pstmt.setString(1,"4657");     // Assign first value to first parameter   2
pstmt.setString(2,"000010");   // Assign first value to second parameter
numUpd = pstmt.executeUpdate(); // Perform first update                   3
pstmt.setString(1,"4658");     // Assign second value to first parameter
pstmt.setString(2,"000020");   // Assign second value to second parameter
numUpd = pstmt.executeUpdate(); // Perform second update
pstmt.close();                 // Close the PreparedStatement object      4
```

図 9. パラメーター・マーカーを指定した SQL ステートメントのために PreparedStatement.executeUpdate を使用する

パラメーター・マーカーのないステートメントの場合も、PreparedStatement.executeUpdate メソッドを使用できます。パラメーター・マーカーのない PreparedStatement オブジェクトを実行する手順は、パラメーター・マーカーを指定した PreparedStatement オブジェクトを実行する手順とよく似ていますが、この場合は、手順 2 (49 ページ) をスキップします。そのための手順を以下の例で示します。

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");
                               // Create a PreparedStatement object  1
numUpd = pstmt.executeUpdate(); // Perform the update                 3
pstmt.close();                 // Close the PreparedStatement object 4
```

図 10. パラメーター・マーカーのない SQL ステートメントのために PreparedStatement.executeUpdate を使用する

DB2 for z/OS サーバーに対する JDBC executeUpdate メソッド

JDBC 標準は、executeUpdate メソッドが行カウントまたは 0 を戻すことを示していますが、executeUpdate メソッドが DB2 for z/OS サーバーに対して実行されると、値 -1 が戻される可能性があります。

DB2 for z/OS サーバーに対する executeUpdate ステートメントの場合、戻される値は、実行中の SQL ステートメントのタイプによって異なります。

- INSERT、UPDATE、DELETE、または MERGE ステートメントなどの更新カウントを持つことができる SQL ステートメントの場合、戻される値は影響を受ける行の数になります。以下のようになります。
  - 正数 (操作によって影響を受ける行の数が正であり、その操作がセグメント表スペースでの一括削除ではない場合)。
  - 0 (操作によって影響を受ける行がない場合)。
  - -1 (操作がセグメント表スペースでの一括削除である場合)。
- SQL CALL ステートメントの場合、データ・ソースは影響を受ける行の数を判別できないため、値 -1 が戻されます。 CALL ステートメントに対して getUpdateCount または getMoreResults を呼び出した場合にも -1 が戻されます。
- その他の SQL ステートメントの場合、値 -1 が戻されます。

JDBC アプリケーションでのバッチ更新の作成

表の各行を 1 行ずつ更新する代わりにバッチ更新を使用すれば、JDBC によって一群の更新を同時に実行できます。同じバッチ更新に組み込めるステートメントのことを*バッチ可能* ステートメントといいます。

入力パラメーターまたはホスト式が含まれているステートメントをバッチに組み込めるのは、そのバッチが同じステートメントの他のインスタンスだけで構成されている場合に限られます。このタイプのバッチのことを*同種バッチ* といいます。入力パラメーターのないステートメントをバッチに組み込めるのは、そのバッチに含まれている他のステートメントにも入力パラメーターやホスト式がない場合に限られます。このタイプのバッチのことを*異種バッチ* といいます。2 つのステートメントを同じバッチに組み込める場合、それらのステートメントは*バッチ互換* であるといいます。

SQL バッチ更新の作成、実行、除去には、以下の Statement メソッドを使用します。

- addBatch
- executeBatch
- clearBatch

以下の PreparedStatement メソッドと CallableStatement メソッドを使用してパラメーターのバッチを作成すれば、1 つのステートメントをバッチの中で複数回実行し、各実行時に別々のパラメーター・セットを適用する、という操作を実行できます。

- addBatch

バッチの中でステートメントを実行するときの制限:

- バッチの中で SELECT ステートメントを実行しようとすると、BatchUpdateException がスローされます。
- バッチの中で実行する CallableStatement オブジェクトには、出力パラメーターを組み込めます。ただし、出力パラメーターの値を取得することはできません。そうしようとすると、BatchUpdateException がスローされます。
- バッチの中で実行する CallableStatement オブジェクトから ResultSet オブジェクトを取得することはできません。BatchUpdateException はスローされませんが、getResultSet メソッド呼び出しでは NULL 値が返されます。

入力パラメーターのない複数のステートメントを使用するバッチ更新を作成するには、以下の基本手順を実行します。

1. バッチの中で実行する各 SQL ステートメントごとに、addBatch メソッドを呼び出します。
2. executeBatch メソッドを呼び出して、ステートメントのバッチを実行します。
3. エラーを確認します。エラーがなければ、以下のようにします。
   a. executeBatch の呼び出しで返された配列から、各 SQL ステートメントで更新された行の数を取得します。この数には、トリガーで更新された行や参照整合性の適用によって更新された行は含めません。
   b. Connection オブジェクトの AutoCommit が無効になっている場合は、commit メソッドを呼び出して変更内容をコミットします。

      Connection オブジェクトの AutoCommit が有効になっている場合は、バッチの最後に IBM Data Server Driver for JDBC and SQLJ が commit メソッドを追加します。

1 つのステートメントで複数の入力パラメーター・セットを使用するバッチ更新を作成するには、以下の基本手順を実行します。

1. prepareStatement メソッドを呼び出して、PreparedStatement オブジェクトを作成します。
2. 入力パラメーター値の各セットごとに、以下のようにします。
   a. setXXX メソッドを実行して、入力パラメーターに値を割り当てます。
   b. addBatch メソッドを呼び出して、入力パラメーターのセットをバッチに追加します。
3. executeBatch メソッドを呼び出して、すべてのパラメーター・セットでステートメントを実行します。
4. エラーがなければ、以下のようにします。
   a. executeBatch の呼び出しで返された配列から、SQL ステートメントのそれぞれの実行で更新された行の数を取得します。
   b. Connection オブジェクトの AutoCommit が無効になっている場合は、commit メソッドを呼び出して変更内容をコミットします。

      Connection オブジェクトの AutoCommit が有効になっている場合は、バッチの最後に IBM Data Server Driver for JDBC and SQLJ が commit メソッドを追加します。
   c. PreparedStatement オブジェクトから自動生成キーが返される場合、DB2PreparedStatement.getDBGeneratedKeys を呼び出して、それらの自動生成キーを含む ResultSet オブジェクトの配列を取得します。

返された配列の長さをチェックします。返された配列の長さが 0 なら、自動生成キー取得時にエラーが発生しています。

5. エラーが発生した場合は、BatchUpdateException を処理します。

以下のコード断片では、2 つのパラメーター・セットをバッチの中に組み込みます。 1 つの UPDATE ステートメントを 2 つの入力パラメーター・セットのそれぞれに基づいて 2 回実行します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
try {
...
  PreparedStatement prepStmt = con.prepareStatement(
    "UPDATE DEPT SET MGRNO=? WHERE DEPTNO=?");          1
  prepStmt.setString(1,mgrnum1);                        2a
  prepStmt.setString(2,deptnum1);
  prepStmt.addBatch();                                  2b

  prepStmt.setString(1,mgrnum2);
  prepStmt.setString(2,deptnum2);
  prepStmt.addBatch();
  int [] numUpdates=prepStmt.executeBatch();            3
  for (int i=0; i < numUpdates.length; i++) {           4a
    if (numUpdates[i] == SUCCESS_NO_INFO)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
    else
      System.out.println("Execution " + i +
        "successful: " numUpdates[i] + " rows updated");
  }
  con.commit();                                         4b
} catch(BatchUpdateException b) {                       5
  // process BatchUpdateException
}
```

以下のコード断片は、バッチ・ステートメントから自動生成キーを返す部分です。

```
try {
...
  PreparedStatement pStmt = con.prepareStatement(       1
    "INSERT INTO DEPT (DEPTNO, DEPTNAME, ADMRDEPT) " +
    "VALUES (?,?,?)",
    Statement.RETURN_GENERATED_KEYS);
  pStmt.setString(1,"X01");                             2a
  pStmt.setString(2,"Finance");
  pStmt.setString(3,"A00");
  pStmt.addBatch();                                     2b
  pStmt.setString(1,"Y01");
  pStmt.setString(2,"Accounting");
  pStmt.setString(3,"A00");
  pStmt.addBatch();

  int [] numUpdates=prepStmt.executeBatch();            3

  for (int i=0; i < numUpdates.length; i++) {           4a
    if (numUpdates[i] == SUCCESS_NO_INFO)
      System.out.println("Execution " + i +
        ": unknown number of rows updated");
    else
      System.out.println("Execution " + i +
        "successful: " numUpdates[i] + " rows updated");
  }
  con.commit();                                         4b
  ResultSet[] resultList =
    ((DB2PreparedStatement)pStmt).getDBGeneratedKeys(); 4c
  if (resultList.length != 0) {
```

```
    for (i = 0; i < resultList.length; i++) {
        while (resultList[i].next()) {
          java.math.BigDecimal idColVar = rs.getBigDecimal(1);
                               // Get automatically generated key
                               // value
          System.out.println("Automatically generated key value = "
            + idColVar);
        }
    }
  }
  else {
    System.out.println("Error retrieving automatically generated keys");
  }
} catch(BatchUpdateException b) {                          5
  // process BatchUpdateException
}
```

ParameterMetaData メソッドを使用した、PreparedStatement のパラメーターに関する情報の取得

IBM Data Server Driver for JDBC and SQLJ には、ParameterMetaData インターフェースのサポートが組み込まれています。ParameterMetaData インターフェースには、PreparedStatement オブジェクトのパラメーター・マーカーに関する情報を取得するためのメソッドが含まれています。

ParameterMetaData メソッドでは、以下のタイプの情報を確認できます。

- パラメーターのデータ・タイプ (10 進パラメーターの精度やスケールなど)。
- パラメーターのデータベース固有のタイプ名。特殊タイプで定義されている表列に対応するパラメーターの場合は、特殊タイプ名になります。
- パラメーターが NULL 可能かどうか。
- 入力パラメーターか出力パラメーターか。
- 数値パラメーターの値に符号を付けられるかどうか。
- PreparedStatement.setObject がパラメーター値を設定するときに使用する完全修飾 Java クラス名。

ParameterMetaData メソッドを呼び出すには、以下の基本手順を実行する必要があります。

1. Connection.prepareStatement メソッドを呼び出して、PreparedStatement オブジェクトを作成します。
2. PreparedStatement.getParameterMetaData メソッドを呼び出して、ParameterMetaData オブジェクトを取得します。
3. ParameterMetaData.getParameterCount を呼び出して、PreparedStatement に含まれているパラメーターの数を確認します。
4. 個々のパラメーターに対して ParameterMetaData メソッドを呼び出します。

ParameterMetaData メソッドを使用して、SQL の UPDATE ステートメントに含まれているパラメーターの数とデータ・タイプを確認するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
Connection con;
ParameterMetaData pmtadta;
int mtadtacnt;
String sqlType;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=? WHERE EMPNO=?");
                                  // Create a PreparedStatement object   1
pmtadta = pstmt.getParameterMetaData();                                  2
                                  // Create a ParameterMetaData object
mtadtacnt = pmtadta.getParameterCount();                                 3
                                  // Determine the number of parameters
System.out.println("Number of statement parameters: " + mtadtacnt);
for (int i = 1; i <= mtadtacnt; i++) {
  sqlType = pmtadta.getParameterTypeName(i);                             4
                                  // Get SQL type for each parameter
 System.out.println("SQL type of parameter " + i " is " + sqlType);
}
...
pstmt.close();                       // Close the PreparedStatement
```

図 11. ParameterMetaData メソッドを使用して PreparedStatement に関する情報を取得する

JDBC アプリケーションにおけるデータの取り出し

JDBC アプリケーションでは、ResultSet オブジェクトを使用してデータを取り出します。 ResultSet は照会の結果セットを表します。

Statement.executeQuery メソッドを使用して表からデータを取得する

パラメーター・マーカーのない SELECT ステートメントを使用して表からデータを取得する場合は、Statement.executeQuery メソッドを使用できます。

このメソッドは、結果表を ResultSet オブジェクトとして返します。結果表を取得したら、ResultSet メソッドを使用して結果表の中を移動し、各行の個々の列値を取得する必要があります。

IBM Data Server Driver for JDBC and SQLJ では、Statement.executeQuery メソッドを使用して、ストアード・プロシージャー呼び出しの結果セットを取得することもできます。ただし、その操作が可能なのは、そのストアード・プロシージャーが 1 つの結果セットだけを返す場合に限られます。ストアード・プロシージャーが複数の結果セットを返す場合は、Statement.execute メソッドを使用する必要があります。

このトピックでは、最もシンプルな ResultSet、つまり読み取り専用の ResultSet を取り上げます。この結果セットでは、1 行ずつ前方に移動することしかできません。 IBM Data Server Driver for JDBC and SQLJ では、更新可能の ResultSet とスクロール可能の ResultSet もサポートされています。

パラメーター・マーカーのない SELECT ステートメントを使用して表から行を取得するには、以下の手順を実行する必要があります。

1. Connection.createStatement メソッドを呼び出して、Statement オブジェクトを作成します。

2. Statement.executeQuery メソッドを呼び出して、SELECT ステートメントの結果表を ResultSet オブジェクトとして取得します。
3. ループの中で、next メソッドを使用してカーソルを配置し、getXXX メソッドを使用して、ResultSet オブジェクトの現在行の各列からデータを取得します。XXX は、データ・タイプに相当します。
4. ResultSet.close メソッドを呼び出して、ResultSet オブジェクトを閉じます。
5. Statement オブジェクトの作業が完了した時点で、Statement.close メソッドを呼び出してそのオブジェクトを閉じます。

従業員表からすべての行を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
String empNo;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement();     // Create a Statement object          1
rs = stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE");                   2
                                  // Get the result table from the query
while (rs.next()) {               // Position the cursor               3
 empNo = rs.getString(1);             // Retrieve only the first column value
 System.out.println("Employee number = " + empNo);
                                  // Print the column value
}
rs.close();                       // Close the ResultSet               4
stmt.close();                     // Close the Statement               5
```

図 12. Statement.executeQuery の使用

PreparedStatement.executeQuery メソッドを使用した表からのデータの取得

パラメーター・マーカーを指定した SELECT ステートメントを使用して表からデータを取得するには、PreparedStatement.executeQuery メソッドを使用します。

このメソッドは、結果表を ResultSet オブジェクトとして返します。結果表を取得したら、ResultSet メソッドを使用して結果表の中を移動し、各行の個々の列値を取得する必要があります。

IBM Data Server Driver for JDBC and SQLJ では、PreparedStatement.executeQuery メソッドを使用して、ストアード・プロシージャー呼び出しの結果セットを取得することもできます。ただし、その操作が可能なのは、そのストアード・プロシージャーが 1 つの結果セットだけを返し、入力パラメーターだけを使用する場合に限られます。ストアード・プロシージャーが複数の結果セットを返す場合は、PreparedStatement.execute メソッドを使用する必要があります。

パラメーター・マーカーのないステートメントの場合も、PreparedStatement.executeQuery メソッドを使用できます。1 つの照会を何度も実行する場合は、SQL ステートメントを PreparedStatement として作成する方がパフォーマンスが良くなります。

パラメーター・マーカーを指定した SELECT ステートメントを使用して表から行を取得するには、以下の手順を実行する必要があります。

1. Connection.prepareStatement メソッドを呼び出して、PreparedStatement オブジェクトを作成します。
2. PreparedStatement.setXXX メソッドを呼び出して、入力パラメーターに値を渡します。
3. PreparedStatement.executeQuery メソッドを呼び出して、SELECT ステートメントの結果表を ResultSet オブジェクトとして取得します。
4. ループの中で、ResultSet.next メソッドを使用してカーソルを配置し、getXXX メソッドを使用して、ResultSet オブジェクトの現在行の各列からデータを取得します。
5. ResultSet.close メソッドを呼び出して、ResultSet オブジェクトを閉じます。
6. PreparedStatement オブジェクトの作業が完了した時点で、PreparedStatement.close メソッドを呼び出してそのオブジェクトを閉じます。

従業員表から特定の従業員の行を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
String empnum, phonenum;
Connection con;
PreparedStatement pstmt;
ResultSet rs;
...
pstmt = con.prepareStatement(
  "SELECT EMPNO, PHONENO FROM EMPLOYEE WHERE EMPNO=?");
                                  // Create a PreparedStatement object    1
pstmt.setString(1,"000010");      // Assign value to input parameter      2

rs = pstmt.executeQuery();        // Get the result table from the query  3
while (rs.next()) {               // Position the cursor                  4
 empnum = rs.getString(1);        // Retrieve the first column value
 phonenum = rs.getString(2);      // Retrieve the first column value
 System.out.println("Employee number = " + empnum +
   "Phone number = " + phonenum);
                                  // Print the column values
}
rs.close();                       // Close the ResultSet                  5
pstmt.close();                    // Close the PreparedStatement          6
```

図 13. PreparedStatement.executeQuery の使用例

JDBC アプリケーションでのバッチ照会の作成

IBM Data Server Driver for JDBC and SQLJ には、同種バッチでバッチ照会を実行するために、 IBM Data Server Driver for JDBC and SQLJ のみの DB2PreparedStatement インターフェースが用意されています。

1 つのステートメントで複数の入力パラメーター・セットを使用するバッチ照会を作成するには、以下の基本手順を実行します。

1. prepareStatement メソッドを呼び出して、入力パラメーターのある SQL ステートメントの PreparedStatement オブジェクトを作成します。
2. 入力パラメーター値の各セットごとに、以下のようにします。

a. PreparedStatement.setXXX メソッドを実行して、入力パラメーターに値を割り当てます。

b. PreparedStatement.addBatch メソッドを呼び出して、入力パラメーターのセットをバッチに追加します。

3. PreparedStatement オブジェクトを DB2PreparedStatement オブジェクトにキャストし、DB2PreparedStatement.executeDB2QueryBatch メソッドを呼び出して、すべてのパラメーター・セットを使ってステートメントを実行します。
4. PreparedStatement オブジェクトに関連した最初の ResultSet オブジェクトを検索します。
5. 最初の ResultSet オブジェクトからすべての行を検索します。
6. ループの中で、以下のように ResultSet オブジェクトの残りを検索します。

   a. 各 ResultSet オブジェクトを検索します。

   b. 各 ResultSet オブジェクトからすべての行を検索します。

例: 以下のコード断片では、2 つのパラメーター・セットをバッチの中に組み込みます。1 つの SELECT ステートメントを 1 つの入力パラメーターのそれぞれのパラメーター値に基づいて 2 回実行します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
java.sql.Connection con = java.sql.DriverManager.getConnection(url, properties);
java.sql.Statement s = con.createStatement();
// Clean up from previous executions
try {
  s.executeUpdate ("drop table TestQBatch");
}
catch (Exception e) {
}

// Create and populate a test table
s.executeUpdate ("create table TestQBatch (col1 int, col2 char(10))");
s.executeUpdate ("insert into TestQBatch values (1, 'test1')");
s.executeUpdate ("insert into TestQBatch values (2, 'test2')");
s.executeUpdate ("insert into TestQBatch values (3, 'test3')");
s.executeUpdate ("insert into TestQBatch values (4, 'test4')");
s.executeUpdate ("insert into TestQBatch values (1, 'test5')");
s.executeUpdate ("insert into TestQBatch values (2, 'test6')");

try {
  PreparedStatement pstmt =                                    1
    con.prepareStatement("Select * from TestQBatch where col1 = ?");
  pstmt.setInt(1,1);                                           2a
  pstmt.addBatch();                                            2b
  // Add some more values to the batch
  pstmt.setInt(1,2);
  pstmt.addBatch();
  pstmt.setInt(1,3);
  pstmt.addBatch();
  pstmt.setInt(1,4);
  pstmt.addBatch();
  ((com.ibm.db2.jcc.DB2PreparedStatement)prepStmt).executeDB2QueryBatch();
                                                               3
} catch(BatchUpdateException b) {
  // process BatchUpdateException
}
ResultSet rs = pstmt.getResultSet();                           4
while(rs.next()) {                                             5
  System.out.print (rs.getInt (1) + "  ");
  System.out.println (rs.getString (2));
}
```

```
rs.close ();
while(pstmt.getMoreResults()) {                                    6
  rs = pstmt.getResultSet();                                       6a
  while (rs.next()) {                                              6b
    System.out.print (rs.getInt (1) + "  ");
    System.out.println (rs.getString (2));
  }
  System.out.println ();
  rs.close ();
}
// Clean up
s.close ();
pstmt.close ();
con.close ();
```

ResultSetMetaData のメソッドを使用した ResultSet の情報入手

表または結果セットの列の数やデータ・タイプがどんな場合にもわかるとは限りません。リモート・データ・ソースからデータを取得する場合は、特にそういえます。

不明な ResultSet を取得するプログラムを作成するときには、ResultSet からデータを取得する前に、ResultSetMetaData メソッドを使用して結果セットの特性を確認する必要があります。

ResultSetMetaData メソッドでは、以下のタイプの情報を確認できます。

- ResultSet の列の数
- ResultSet の基礎表の修飾子
- 列に関する情報 (データ・タイプ、長さ、精度、スケール、NULL 可能など)
- 列が読み取り専用かどうか

executeQuery メソッドを呼び出して、表の照会の ResultSet を生成したら、以下の基本手順を実行して、ResultSet の内容を確認します。

1. ResultSet オブジェクトに対して getMetaData メソッドを呼び出し、ResultSetMetaData オブジェクトを作成します。
2. getColumnCount メソッドを呼び出して、ResultSet の列の数を確認します。
3. ResultSet の各列ごとに ResultSetMetaData メソッドを呼び出して、列の特性を確認します。

   ResultSetMetaData.getColumnName 呼び出しの結果は、データ・ソースに関して DB2 のカタログに格納されている列名情報を反映しています。

従業員表のすべての列のデータ・タイプを確認するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
String s;
Connection con;
Statement stmt;
ResultSet rs;
ResultSetMetaData rsmtadta;
int colCount
int mtadtaint;
int i;
String colName;
String colType;
...
stmt = con.createStatement();     // Create a Statement object
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Get the ResultSet from the query
rsmtadta = rs.getMetaData();      // Create a ResultSetMetaData object  1
colCount = rsmtadta.getColumnCount();                                  2
                                  // Find number of columns in EMP
for (i=1; i<= colCount; i++) {                                         3
 colName = rsmtadta.getColumnName();    // Get column name
 colType = rsmtadta.getColumnTypeName();
                                        // Get column data type
 System.out.println("Column = " + colName +
  " is data type " + colType);
                                        // Print the column value
}
```

図 14. ResultSetMetaData メソッドを使用して ResultSet に関する情報を取得する

IBM Data Server Driver for JDBC and SQLJ 使用時の JDBC ResultSet の特性

IBM Data Server Driver for JDBC and SQLJ では、両方向スクロール可能、更新可能、および保持可能なカーソルのサポートが提供されます。

ResultSet を順方向に 1 行ずつ移動することに加えて、以下のことも実行することができます。

- 逆方向に移動する、または特定の行へ直接移動する
- ResultSet に対して行を更新、削除、または挿入する
- COMMIT 後に ResultSet をオープンしたままにしておく

以下の各用語により、ResultSet の特性が説明できます。

スクロール可能性
ResultSet のカーソルを順方向のみ、1 つ以上の行を順方向、1 つ以上の行を逆方向、または特定の行に移動できるかどうか。

ResultSet のカーソルが両方向にスクロール可能な場合、このカーソルには感度属性もあり、カーソルが基礎表に対する変更にセンシティブであるかどうかが表されます。

更新可能性
カーソルが行の更新または削除に使用できるかどうか。この特性は、ストアード・プロシージャーの ResultSet が更新できないため、ストアード・プロシージャーから返された ResultSet には当てはまりません。

保持可能性
COMMIT 後にカーソルがオープンしたままかどうか。

Connection.prepareStatement または Connection.createStatement メソッド内のパラメーターによって、ResultSet の更新可能性、スクロール可能性、および保持可能性の特性を設定します。ResultSet 設定は、データベース内のカーソルの属性にマップされます。以下の表に、JDBC のスクロール可能性、更新可能性、保持可能性の設定、および対応するカーソルの属性をリストしています。

表 9. JDBC ResultSet の特性および SQL カーソルの属性

JDBC 設定	DB2 カーソル設定	IBM Informix Dynamic Server カーソル設定
CONCUR_READ_ONLY	FOR READ ONLY	FOR READ ONLY
CONCUR_UPDATABLE	FOR UPDATE	FOR UPDATE
HOLD_CURSORS_OVER_COMMIT	WITH HOLD	WITH HOLD
TYPE_FORWARD_ONLY	SCROLL 指定なし	SCROLL 指定なし
TYPE_SCROLL_INSENSITIVE	INSENSITIVE SCROLL	SCROLL
TYPE_SCROLL_SENSITIVE	cursorSensitvity Connection および DataSource プロパティーに基づく、SENSITIVE STATIC、SENSITIVE DYNAMIC、または ASENSITIVE	サポートされていない

JDBC の ResultSet が静的である場合は、カーソルがオープンされた後も、結果表のサイズや結果表の行の順序は変わりません。つまり、行を基礎表に挿入した場合も、静的 ResultSet の結果表は変化しません。結果表の行を削除すると削除ホールが生じます。削除ホールを更新または削除することはできません。

JDBC アプリケーションにおける ResultSets の更新可能性、スクロール可能性、および保持可能性の指定:

ResultSet の更新可能性、スクロール可能性、および保持可能性を指定するには、Connection.prepareStatement または Connection.createStatement メソッド内の特殊なパラメーターを使用します。

デフォルトでは、ResultSet オブジェクトはスクロール可能ではなく、更新可能でもありません。デフォルトの保持可能性は、データ・ソースによって異なります。保持可能性を確認するには、DatabaseMetaData.getResultSetHoldability メソッドを使用します。 ResultSet のスクロール可能性、更新可能性、保持可能性の属性を変更するには、以下の手順を実行します。

1. ResultSet を定義する SELECT ステートメントに入力パラメーターが含まれていない場合、createStatement メソッドを呼び出して Statement オブジェクトを作成します。それ以外の場合は、prepareStatement メソッドを呼び出して PreparedStatement オブジェクトを作成します。 *resultSetType*、*resultSetConcurrency*、または *resultSetHoldability* パラメーターが含まれる createStatement または prepareStatement メソッドの形式を指定する必要があります。

   スクロール可能性および更新可能性をサポートする createStatement メソッドの形式は次のようになります。

   ```
   createStatement(int resultSetType, int resultSetConcurrency);
   ```

スクロール可能性、更新可能性、および保持可能性をサポートする createStatement メソッドの形式は次のようになります。

```
createStatement(int resultSetType, int resultSetConcurrency,
 int resultSetHoldability);
```

スクロール可能性および更新可能性をサポートする prepareStatement メソッドの形式は次のようになります。

```
prepareStatement(String sql, int resultSetType,
 int resultSetConcurrency);
```

スクロール可能性、更新可能性、および保持可能性をサポートする prepareStatement メソッドの形式は次のようになります。

```
prepareStatement(String sql, int resultSetType,
 int resultSetConcurrency, int resultSetHoldability);
```

以下の表には、*resultSetType* および *resultSetConcurrency* の有効値のリストが含まれています。

表 10. ResultSets の resultSetType および resultSetConcurrency の有効な組み合わせ

resultSetType **値**	*resultSetConcurrency* **値**
TYPE_FORWARD_ONLY	CONCUR_READ_ONLY
TYPE_FORWARD_ONLY	CONCUR_UPDATABLE
TYPE_SCROLL_INSENSITIVE	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE[1]	CONCUR_READ_ONLY
TYPE_SCROLL_SENSITIVE[1]	CONCUR_UPDATABLE

注:

1. この値は IBM Informix Dynamic Server への接続には適用されません。

resultSetHoldability には、HOLD_CURSORS_OVER_COMMIT および CLOSE_CURSORS_AT_COMMIT の 2 つの可能な値が含まれます。これらの値はいずれも、*resultSetConcurrency* および *resultSetHoldability* の有効な組み合わせを使用して指定できます。設定した値は、接続のデフォルトの保持可能性をオーバーライドします。

制約事項: ResultSet がスクロール可能であり、 DB2 Database for Linux, UNIX, and Windows サーバーで ResultSet を使用して表から列を選択する場合、ResultSet を定義する SELECT ステートメントの SELECT リストには、以下のデータ・タイプの列を含めることはできません。

- LONG VARCHAR
- LONG VARGRAPHIC
- BLOB
- CLOB
- XML
- このリスト内の前述のデータ・タイプのいずれかに基づいた特殊タイプ
- 構造化タイプ

2. SELECT ステートメントに入力パラメーターが含まれる場合、setXXX メソッドを呼び出して値を入力パラメーターに渡します。

3. executeQuery メソッドを呼び出して、ResultSet オブジェクトの SELECT ステートメントから結果表を取得します。
4. アクセスする各行ごとに以下のようにします。
   a. 以下の表にリストされているメソッドのいずれかを使用してカーソルを置きます。

表 11. 両方向スクロール・カーソルを置くための ResultSet メソッド

メソッド	カーソルの位置
first[1]	ResultSet の最初の行
last[1]	ResultSet の最後の行
next[2]	ResultSet の次の行
previous[1,3]	ResultSet の前の行
`absolute(int n)`[1,4]	*n*>0 の場合は ResultSet の *n* 行。 *n*<0 であり、*m* が ResultSet の行数である場合は、ResultSet の *m+n+1* 行。
`relative(int n)`[1,5,6,]	*n*>0 の場合、現在行の *n* 行後ろにある行。 *n*<0 の場合、現在行の *n* 行前にある行。 *n*=0 の場合は現在行。
afterLast[1]	ResultSet の最後の行の後ろ
beforeFirst[1]	ResultSet の最初の行の前

注:

1. この方法は、 IBM Informix Dynamic Server への接続には適用されません。
2. カーソルが ResultSet の最初の行の前にある場合、このメソッドではカーソルは最初の行に置かれます。
3. カーソルが ResultSet の最後の行の後ろにある場合、このメソッドではカーソルは最後の行に置かれます。
4. このメソッドでは、*n* の絶対値が結果セットの行数よりも大きい場合、*n* が正であれば最後の行の後ろ、*n* が負であれば最初の行の前にカーソルが置かれます。
5. このメソッドを使用する前に、カーソルは ResultSet の有効な行を示している必要があります。カーソルが最初の行の前あるいは最後の行の後ろにある場合、メソッドは SQLException をスローします。
6. *m* が ResultSet の行数、そして *x* が ResultSet の現在行の番号だとします。 *n*>0 であり、*x+n>m* である場合、ドライバーは最後の行の後ろにカーソルを置きます。 *n*<0 であり、*x+n*<1 である場合、ドライバーは最初の行の前にカーソルを置きます。

   b. 現行カーソル位置を知る必要がある場合には、getRow、isFirst、isLast、`isBeforeFirst`、または isAfterLast メソッドを使用してこの情報を取得します。
   c. ステップ 1 (61 ページ) で、TYPE_SCROLL_SENSITIVE の *resultSetType* 値を指定しており、現在行の最新の値を参照する必要がある場合には、`refreshRow` メソッドを呼び出します。

      推奨: ResultSet の行をリフレッシュするとご使用のアプリケーションのパフォーマンスに悪影響を及ぼす可能性があるため、最新のデータを参照する必要がある場合にのみ、refreshRow を呼び出してください。
   d. 以下の操作の 1 つ以上を実行します。

- ResultSet オブジェクトの現在行の各列からデータを取り出すには、getXXX メソッドを使用します。
- 基礎表から現在行を更新するには、updateXXX メソッドを使用して列値を ResultSet の現在行に割り当てます。次に、updateRow を使用して基礎表の対応する行を更新します。基礎表を更新しない場合は、updateRow メソッドの代わりに cancelRowUpdates メソッドを呼び出します。

  これらのメソッドを使用するには、ResultSet の *resultSetConcurrency* 値は CONCUR_UPDATABLE でなければなりません。
- 基礎表から現在行を削除するには、deleteRow メソッドを使用します。deleteRow を呼び出すと、ドライバーは ResultSet の現在行をホールに置き換えます。

  このメソッドを使用するには、ResultSet の *resultSetConcurrency* 値は CONCUR_UPDATABLE でなければなりません。

5. close メソッドを呼び出して ResultSet オブジェクトをクローズします。
6. close メソッドを呼び出して Statement または PreparedStatement オブジェクトをクローズします。

以下のコードは、従業員表からすべての行を逆順で取り出し、従業員番号 ″000010″ の電話番号を更新する方法について説明しています。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
String s;
String stmtsrc;
Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_UPDATABLE);                1
                                       // Create a Statement object
                                       // for a scrollable, updatable
                                       // ResultSet
stmtsrc = "SELECT EMPNO, PHONENO FROM EMPLOYEE " +
          "FOR UPDATE OF PHONENO";
rs = stmt.executeQuery(stmtsrc);       // Create the ResultSet         3
rs.afterLast();                        // Position the cursor at the end of
                                       // the ResultSet                4a
while (rs.previous()) {                // Position the cursor backward
 s = rs.getString("EMPNO");            // Retrieve the employee number 4d
                                       // (column 1 in the result
                                       // table)
 System.out.println("Employee number = " + s);
                                       // Print the column value
 if (s.compareTo("000010") == 0) {     // Look for employee 000010
  rs.updateString("PHONENO","4657");   // Update their phone number
  rs.updateRow();                      // Update the row
 }
}
rs.close();                            // Close the ResultSet          5
stmt.close();                          // Close the Statement          6
```

図 15. 両方向スクロール・カーソルの使用

IBM Data Server Driver for JDBC and SQLJ を使用した複数行の SQL 操作:

IBM Data Server Driver for JDBC and SQLJ では、複数行の INSERT、UPDATE、およびこれらの操作をサポートするデータ・ソースへの接続用の FETCH がサポートされます。

複数行 INSERT

複数行 FETCH では、各 FETCH ステートメントを使って 1 行を取り出す場合よりも良好なパフォーマンスを提供できます。 DB2 for z/OS の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 では、複数行 FETCH は前方スクロール・カーソルと両方向スクロール・カーソルで使用できます。 IBM Data Server Driver for JDBC and SQLJ Type 4 接続の場合、複数行 FETCH は以下の状況で使用できます。

- JDBC または SQLJ プログラムでは両方向スクロール・カーソルの場合
- カスタマイズ SQLJ プログラムでは前方スクロール・カーソルの場合

複数行 INSERT ステートメントを JDBC アプリケーション内に組み込んで、複数行の挿入操作を実行することはできません。

複数行 FETCH

複数行 FETCH では、各 FETCH ステートメントを使って 1 行を取り出す場合よりも良好なパフォーマンスを提供できます。 DB2 for z/OS の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 では、複数行 FETCH は前方スクロール・カーソルと両方向スクロール・カーソルで使用できます。 IBM Data Server Driver for JDBC and SQLJ Type 4 接続の場合、複数行 FETCH は両方向スクロール・カーソルでのみ使用できます。

アプリケーションでデータを取り出す際、 IBM Data Server Driver for JDBC and SQLJ は、以下のいくつかの要因に応じて、複数行 FETCH を使用するかどうか決定します。

- enableRowsetSupport および useRowsetCursor プロパティーの設定
- 使用されている IBM Data Server Driver for JDBC and SQLJ 接続のタイプ
- IBM Data Server Driver for JDBC and SQLJ のバージョン

DB2 for z/OS への IBM Data Server Driver for JDBC and SQLJ Type 4 接続の場合、複数行 FETCH を使用するには、次の条件セットのいずれかを満たしている必要があります。

- 最初の条件セットは、次のとおりです。
  - IBM Data Server Driver for JDBC and SQLJ バージョンが 3.51 以降である。
  - enableRowsetSupport プロパティー値が com.ibm.db2.jcc.DB2BaseDataSource.YES (1) である。**または** enableRowsetSupport プロパティー値が com.ibm.db2.jcc.DB2BaseDataSource.NOT_SET (0) で、useRowsetCursor プロパティー値が com.ibm.db2.jcc.DB2BaseDataSource.YES (1) である。
  - FETCH 操作で両方向スクロール・カーソルを使用する。

前方スクロール・カーソルの場合、DRDA ブロック FETCH を介して複数行のフェッチが行われる場合があります。しかし、この動作では、データ・ソースの複数行 FETCH 機能が使用されません。

- 2 番目の条件セットは、次のとおりです。
  - IBM Data Server Driver for JDBC and SQLJ バージョンが 3.1 である。
  - useRowsetCursor プロパティー値が com.ibm.db2.jcc.DB2BaseDataSource.YES (1) である。
  - FETCH 操作で両方向スクロール・カーソルを使用する。

    前方スクロール・カーソルの場合、DRDA ブロック FETCH を介して複数行のフェッチが行われる場合があります。しかし、この動作では、データ・ソースの複数行 FETCH 機能が使用されません。

DB2 for z/OS への IBM Data Server Driver for JDBC and SQLJ Type 2 接続の場合、複数行 FETCH を使用するには、次の条件を満たしている必要があります。

- IBM Data Server Driver for JDBC and SQLJ バージョンが 3.51 以降である。
- enableRowsetSupport プロパティー値が com.ibm.db2.jcc.DB2BaseDataSource.YES (1) である。
- FETCH 操作で両方向スクロール・カーソルまたは前方スクロール・カーソルを使用する。

DB2 for z/OS の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 では、maxRowsetSize プロパティーを設定することによりステートメントごとの行セットの最大サイズを制御することができます。

複数行の位置指定 UPDATE または位置指定 DELETE

IBM Data Server Driver for JDBC and SQLJ では、JDBC 1 標準に従った位置指定の更新または削除操作を実行するための手法がサポートされます。この手法には、ResultSet.getCursorName メソッドを使用して、ResultSet のカーソルの名前を取得して、以下の形式の位置指定 UPDATE ステートメントまたは位置指定 DELETE ステートメントを定義することが含まれます。

```
UPDATE table SET col1=value1,...coln=valueN WHERE CURRENT OF cursorname
DELETE FROM table WHERE CURRENT OF cursorname
```

useRowsetCursor が `true` に設定される場合の複数行 UPDATE または DELETE:
JDBC 1 手法を使用して、複数行 FETCH をサポートするデータベース・サーバーでデータを更新または削除するときに複数行 FETCH が useRowsetCursor プロパティーによって使用可能になっていると、単一行を更新または削除しようとしても、位置指定 UPDATE または DELETE ステートメントが複数の行を更新または削除してしまう場合があります。予期しない更新または削除が実行されないようにするために、下記の処置のいずれかを実行することができます。

- 前の例に示されているように、更新可能な `ResultSet` を使用して、一度に 1 行を取り出して更新する。
- useRowsetCursor を `false` に設定する。

enableRowsetSupport が *com.ibm.db2.jcc.DB2BaseDataSource.YES (1)* に設定されている場合の複数行 *UPDATE* または *DELETE:* データを更新または削除するための JDBC 1 手法には、enableRowsetSupport プロパティーによって使用可能にされる複数行 FETCH との互換性がありません。

推奨: アプリケーションが JDBC 1 手法を使用する場合、位置指定による更新または削除アクティビティーで JDBC 2.0 ResultSet.updateRow または ResultSet.deleteRow メソッドを使用するように、それらのアプリケーションを更新してください。

JDBC アプリケーションで ResultSet の現在行が削除ホールまたは更新ホールになっているかどうかをテストする:

ResultSet に TYPE_SCROLL_SENSITIVE 属性があり、基礎になっているカーソルが SENSITIVE STATIC の場合は、ResultSet の行を取得する前に、削除ホールまたは更新ホールがあるかどうかをテストする必要があります。

SENSITIVE STATIC の ResultSet を開いた後も、サイズ変更はありません。削除された行は、プレースホルダーに置き換えられているからです。そのような行のことを ホール といいます。更新された行が ResultSet の基準に適合しなくなった場合も、ホールになります。ホールになった行を取得することはできません。

ResultSet の現在行が削除ホールまたは更新ホールになっているかどうかをテストするには、以下の手順を実行します。

1. DatabaseMetaData.deletesAreDetected メソッドまたは DatabaseMetaData.updatesAreDetected メソッドに TYPE_SCROLL_SENSITIVE 引数を指定して呼び出し、データ・ソースが TYPE_SCROLL_SENSITIVE の ResultSet でホールを作成するかどうかを確認します。
2. DatabaseMetaData.deletesAreDetected または DatabaseMetaData.updatesAreDetected から `true` が返された場合は、データ・ソースがホールを作成できるという意味なので、ResultSet.rowDeleted メソッドまたは ResultSet.rowUpdated メソッドを呼び出して、現在行が削除ホールまたは更新ホールになっているかどうかを確認します。 メソッドから `true` が返された場合は、現在行がホールになっています。

現在行が削除ホールになっているかどうかをテストするコードを以下に示します。

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
  ResultSet.CONCUR_UPDATABLE);
                                     // Create a Statement object
                                     // for a scrollable, updatable
                                     // ResultSet
ResultSet rs =
  stmt.executeQuery("SELECT EMPNO FROM EMPLOYEE FOR UPDATE OF PHONENO");
                                     // Create the ResultSet
DatabaseMetaData dbmd = con.getMetaData();
                                     // Create the DatabaseMetaData object
boolean dbSeesDeletes =
  dbmd.deletesAreDetected(ResultSet.TYPESCROLL_SENSITIVE);
                                     // Can the database see delete holes?
rs.afterLast();                      // Position the cursor at the end of
                                     // the ResultSet
while (rs.previous()) {              // Position the cursor backward
 if (dbSeesDeletes) {                // If delete holes can be detected
   if (!(rs.rowDeleted()))           // If this row is not a delete hole
```

```
    {
      s = rs.getString("EMPNO");         // Retrieve the employee number
      System.out.println("Employee number = " + s);
                                         // Print the column value
    }
   }
}
rs.close();                              // Close the ResultSet
stmt.close();                            // Close the Statement
```

JDBC アプリケーションで ResultSet に行を挿入する:

ResultSet の *resultSetConcurrency* 属性が CONCUR_UPDATABLE の場合は、ResultSet に行を挿入できます。

ResultSet に行を挿入するには、以下の手順を実行します。

1. 挿入する行それぞれに対して、以下のステップを実行します。
   a. ResultSet.moveToInsertRow メソッドを呼び出して、挿入する行を作成します。行は、ResultSet の外にあるバッファーに作成されます。

      既存の挿入バッファーがある場合は、バッファーから以前の値すべてがクリアされます。
   b. ResultSet.updateXXX メソッドを呼び出して、挿入する行に値を割り当てます。

      ResultSet 内の少なくとも 1 つの列に値を割り当てる必要があります。そうしない場合、その行が ResultSet に挿入されるときに、SQLException がスローされます。

      ResultSet の列に値を割り当てない場合、基礎表の更新時にデータ・ソースはその関連表の列にデフォルト値を挿入します。

      NOT NULL として定義されている列に NULL 値を割り当てた場合、JDBC ドライバーは SQLException をスローします。
   c. ResultSet.insertRow を呼び出して、その行を ResultSet に挿入します。

      ResultSet.insertRow を呼び出すと、ResultSet.insertRow が失敗したとしても、挿入バッファーから常にすべての値がクリアされます。
2. ResultSet 内にカーソルを位置変更します。

   カーソルを挿入行から ResultSet に移動するには、ResultSet.first、ResultSet.absolute、または ResultSet.relative といった、カーソルを特定の行に配置する任意のメソッドを呼び出します。あるいは、ResultSet.moveToCurrentRow を呼び出して、ResultSet 内の、挿入操作が実行される前に現在行であった行にカーソルを移動できます。

   ResultSet.moveToCurrentRow の呼び出し後には、すべての値が挿入バッファーからクリアされます。

例: 以下のコードは、サンプル DEPARTMENT 表にあるすべての行から成る ResultSet に 1 つの行を挿入する例を示します。行を挿入した後、このコードは、

挿入操作前にカーソルが置かれていた ResultSet 内の位置にカーソルを配置します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
               ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM DEPARTMENT");
rs.moveToInsertRow();                                          1a
rs.updateString("DEPT_NO", "M13");                             1b
rs.updateString("DEPTNAME", "TECHNICAL SUPPORT");
rs.updateString("MGRNO", "000010");
rs.updateString("ADMRDEPT", "A00");
rs.insertRow();                                                1c
rs.moveToCurrentRow();                                         2
```

JDBC アプリケーションで現在行が ResultSet に挿入されたかどうかをテストする:

ResultSet が動的であれば、行を挿入できます。行を ResultSet に挿入した後に、どの行が挿入されたかを確認しなければならない場合もあります。

現在行が ResultSet に挿入されたかどうかをテストするには、以下の手順を実行します。

1. DatabaseMetaData.ownInsertsAreVisible メソッドと DatabaseMetaData.othersInsertsAreVisible メソッドを呼び出して、このタイプの ResultSet で挿入内容を参照できるかどうかを確認します。
2. この ResultSet で挿入内容を参照できるようであれば、DatabaseMetaData.insertsAreDetected メソッドを呼び出して、このタイプの ResultSet で挿入内容を検出できるかどうかを確認します。
3. この ResultSet で挿入内容を検出できるようであれば、ResultSet.rowInserted メソッドを呼び出して、現在行が挿入されたかどうかを確認します。

JDBC アプリケーションでのストアード・プロシージャーの呼び出し

ストアード・プロシージャーを呼び出すには、CallableStatement クラスのメソッドを呼び出します。

標準の CallableStatement メソッドを使用してストアード・プロシージャーを呼び出すための基本手順は、以下のとおりです。

1. CALL ステートメントを引数として Connection.prepareCall メソッドを呼び出し、CallableStatement オブジェクトを作成します。

   パラメーターは、標準パラメーター・マーカー (?)、名前付きパラメーター・マーカー、または名前付きパラメーターを使用して表すことができます。同じ CALL ステートメントで名前付きパラメーターと標準パラメーター・マーカーを混用することはできますが、 名前付きパラメーター・マーカーと標準パラメーター・マーカーまたは名前付きパラメーターを混用することはできません。

   制約事項: 使用できるパラメーター・タイプは、データ・ソースが CALL ステートメントの動的実行をサポートしているかどうかによって異なります。 DB2 for z/OS は、CALL ステートメントの動的実行をサポートしていません。 DB2 for z/OS データベース・サーバーに格納されているストアード・プロシージャー

の呼び出しでは、パラメーターとしてパラメーター・マーカーやリテラルを使用できますが、式は使用できません。サポートされているリテラルのタイプとそれに対応する JDBC タイプを以下の表にまとめます。

表 12. DB2 for z/OS のストアード・プロシージャー呼び出しのパラメーターでサポートされているリテラル・タイプ

リテラル・パラメーター・タイプ	JDBC タイプ	例
整数	java.sql.Types.INTEGER	-122、40022、+27
浮動小数点 10 進数	java.sql.Types.DOUBLE	23E12、40022E-4、+2723E+15、1E+23、0E0
固定小数点 10 進数	java.sql.Types.DECIMAL	-23.12、40022.4295、0.0、+2723.23、10000000000
文字	java.sql.Types.VARCHAR	'Grantham Lutz'、'O'、'Conner'、'ABcde?z?'
16 進数	java.sql.Types.VARBINARY	X'C1C30427'、X'00CF18E0'
Unicode ストリング	java.sql.Types.VARCHAR	UX'0041'、UX'0054006500730074'

2. CallableStatement.setXXX メソッドを呼び出して、入力パラメーター (CREATE PROCEDURE ステートメント内で IN または INOUT として定義されているパラメーター) に値を渡します。

   このステップでは、標準パラメーター・マーカーまたは名前付きパラメーターを使用することを想定しています。別の方法として名前付きパラメーター・マーカーを使用する場合、 IBM Data Server Driver for JDBC and SQLJ のみのメソッドを使用して、値を入力パラメーターに渡します。

   制約事項: データ・ソースが CALL ステートメントの動的実行をサポートしていない場合は、CALL ステートメントの入力パラメーターのデータ・タイプをストアード・プロシージャー定義で指定されている**とおりに**指定する必要があります。

3. CallableStatement.registerOutParameter メソッドを呼び出して、CREATE PROCEDURE ステートメント内で OUT として定義されているパラメーターのデータ・タイプを登録します。

   このステップでは、標準パラメーター・マーカーまたは名前付きパラメーターを使用することを想定しています。別の方法として名前付きパラメーター・マーカーを使用する場合、 IBM Data Server Driver for JDBC and SQLJ のみのメソッドを使用して、特定のデータ・タイプで OUT パラメーターを登録します。

   制約事項: データ・ソースが CALL ステートメントの動的実行をサポートしていない場合は、CALL ステートメントの OUT、IN、または INOUT パラメーターのデータ・タイプをストアード・プロシージャー定義で指定されている**とおりに**指定する必要があります。

4. 以下のいずれかのメソッドを呼び出して、ストアード・プロシージャーを呼び出します。

   CallableStatement.executeUpdate
   ストアード・プロシージャーが結果セットを返さない場合は、このメソッドを呼び出します。

CallableStatement.executeQuery
ストアード・プロシージャーが 1 つの結果セットを返す場合は、このメソッドを呼び出します。

CallableStatement.execute
ストアード・プロシージャーが複数の結果セットを返す場合や、結果セットの数が不明な場合は、このメソッドを呼び出します。

制約事項: IBM Informix Dynamic Server (IDS) データ・ソースは、複数の結果セットをサポートしていません。

5. ストアード・プロシージャーが複数の結果セットを返す場合は、それらの結果セットを取得します。

   制約事項: IDS データ・ソースは、複数の結果セットをサポートしていません。

6. CallableStatement.getXXX メソッドを呼び出して、OUT パラメーターまたは INOUT パラメーターから値を取得します。
7. CallableStatement オブジェクトの作業が完了した時点で、CallableStatement.close メソッドを呼び出してそのオブジェクトを閉じます。

例: 1 つの入力パラメーター、4 つの出力パラメーターがあり、ResultSet を返さないストアード・プロシージャーを呼び出すコード例を以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
int ifcaret;
int ifcareas;
int xsbytes;
String errbuff;
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
cstmt = con.prepareCall("CALL DSN8.DSN8ED2(?,?,?,?,?)");                1
                                  // Create a CallableStatement object
cstmt.setString (1, "DISPLAY THREAD(*)");                                2
                                  // Set input parameter (DB2 command)
cstmt.registerOutParameter (2, Types.INTEGER);                           3
                                  // Register output parameters
cstmt.registerOutParameter (3, Types.INTEGER);
cstmt.registerOutParameter (4, Types.INTEGER);
cstmt.registerOutParameter (5, Types.VARCHAR);
cstmt.executeUpdate();            // Call the stored procedure           4
ifcaret = cstmt.getInt(2);        // Get the output parameter values     6
ifcareas = cstmt.getInt(3);
xsbytes = cstmt.getInt(4);
errbuff = cstmt.getString(5);
cstmt.close();                                                           7
```

JDBC アプリケーションの CALL ステートメントでの名前付きパラメーターの使用

IBM Data Server Driver for JDBC and SQLJ では、ストアード・プロシージャーを呼び出す際にいくつかの方法で名前付きパラメーターを使用できます。名前付きパラメーターは、名前付きパラメーター・マーカーとは異なる構文を使用します。

JDBC アプリケーション内の以下のいずれか (または両方) の場所で名前付きパラメーターを使用することができます。

- CALL ステートメントの中

  名前付きパラメーターを使用する場合、ストアード・プロシージャー定義内と同じ順序で CALL ステートメント内のパラメーターを指定する必要はありません。さらに、すべてのパラメーターを CALL ステートメントで指定する必要はありません。指定されないパラメーターは、ストアード・プロシージャー定義で指定されたデフォルト値になります。

- CallableStatement.setXXX、CallableStatement.getXXX、および CallableStatement.registerOutParameter メソッドの中

  (定義内のパラメーターの位置ではなく) ストアード・プロシージャー定義で表されるとおりにパラメーター名を指定することで、見やすいプログラムになります。

名前付きパラメーターを CALL ステートメントで使用するには、以下の手順に従います。

1. CALL ステートメントを引数として Connection.prepareCall メソッドを呼び出し、CallableStatement オブジェクトを作成します。

   各パラメーターを示すには、パラメーター・マーカー (?) または次のような構文を使用できます。

   ```
   parameter-name=>?
   ```

   この *parameter-name* は CREATE PROCEDURE ステートメント内のパラメーターを識別します。

   DEFAULT キーワードまたは NULL キーワードを指定することで、名前付きパラメーターにデフォルト値や NULL 値を明示的に割り当てることができます。デフォルト値が CREATE PROCEDURE ステートメントで指定されるパラメーターの場合、CALL ステートメントでこれらのパラメーターを省略することで、名前付きパラメーターに暗黙的にデフォルト値を割り当てることができます。省略されるすべてのパラメーターのデフォルト値がストアード・プロシージャー定義で指定される場合に限り、パラメーターを省略できます。

   名前付きパラメーターと名前付きパラメーター・マーカーを同じ CALL ステートメント内で混用することはできません。

2. CallableStatement.setXXX メソッドを呼び出して、入力パラメーター (CREATE PROCEDURE ステートメント内で IN または INOUT として定義されているパラメーター) に値を渡します。

   以下のいずれかの方法で、値を割り当てることができます。

   - CallableStatement.setXXX(parameterIndex, ...) を使って位置指定する
   - CallableStatement.setXXX(parameterName, ...) を使って名前を指定する

     parameterName は二重引用符で囲まれたストリングで、その値は CREATE PROCEDURE ステートメント内のパラメーター名と一致します。

3. CallableStatement.registerOutParameter メソッドを呼び出して、CREATE PROCEDURE ステートメント内で OUT として定義されているパラメーターのデータ・タイプを登録します。

4. CallableStatement.executeUpdate、CallableStatement.executeQuery、または CallableStatement.execute を呼び出してストアード・プロシージャーを実行します。
5. ストアード・プロシージャーが複数の結果セットを返す場合は、それらの結果セットを取得します。

   以下のいずれかの方法で、出力パラメーターを登録することができます。

   - CallableStatement.registerOutParameter(parameterIndex, ...) を使って位置指定する
   - CallableStatement.registerOutParameter(parameterName, ...) を使って名前を指定する

     parameterName は二重引用符で囲まれたストリングで、その値は CREATE PROCEDURE ステートメント内のパラメーター名と一致します。
6. CallableStatement.getXXX メソッドを呼び出して、OUT パラメーターまたは INOUT パラメーターから値を取得します。

   以下のいずれかの方法で、値を取得することができます。

   - CallableStatement.getXXX(parameterIndex, ...) を使って位置指定する
   - CallableStatement.getXXX(parameterName, ...) を使って名前を指定する

     parameterName は二重引用符で囲まれたストリングで、その値は CREATE PROCEDURE ステートメント内のパラメーター名と一致します。
7. CallableStatement オブジェクトの作業が完了した時点で、CallableStatement.close メソッドを呼び出してそのオブジェクトを閉じます。

以下のコードでは、次のような定義のストアード・プロシージャーが呼び出されます。

```
CREATE PROCEDURE SALS (
  OUT retcode INTEGER,
  IN lowsal DOUBLE,
  IN medsal DOUBLE,
  IN highsal DOUBLE DEFAULT 100000,
  IN department CHAR(3) DEFAULT '---')
SPECIFIC JDBC_SALS
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO DBINFO
FENCED
THREADSAFE
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'MYJAR:MyClass.sals'
```

CALL ステートメント内の入力パラメーターは名前付きパラメーターで表されます。 3 番目と 4 番目のパラメーターは、ストアード・プロシージャーのデフォルト値を使って呼び出されます。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
int hvRetCode;                      // Host variable for output parameter
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
```

```
cstmt = con.prepareCall(
  "CALL SALS(retcode=>?,lowsal=>?,medsal=>?,highsal=>DEFAULT)");   1
                                  // Prepare the Call statement.
                                  // Implicitly use the default
                                  // value for the last parameter
                                  // by omitting it.
cstmt.setDouble ("lowsal", 10000);                                 2
cstmt.setDouble ("medsal", 50000);
cstmt.registerOutParameter ("retcode", Types.INTEGER);             3
                                  // Register output parameter
cstmt.executeUpdate();            // Call the stored procedure     4
hvRetCode = cstmt.getInt("retcode");                               6
System.out.println("Return code from SALS call: " + hvRetCode);
cstmt.close();                                                     7
```

JDBC アプリケーションでのカーソル出力パラメーターからのデータの取得

DB2 Database for Linux, UNIX, and Windows ストアード・プロシージャーでは、カーソル・タイプの OUT パラメーターを使用できます。 JDBC アプリケーションでこれらのパラメーターからデータを取得するには、ResultSet オブジェクトを使用します。

カーソル変数からデータを取得するには、以下の手順に従います。

1. カーソル・データ・タイプの OUT パラメーターごとに ResultSet オブジェクトをそれぞれ定義します。
2. CALL ステートメントを引数として Connection.prepareCall メソッドを呼び出し、CallableStatement オブジェクトを作成します。
3. CallableStatement.registerOutParameter メソッドを呼び出して、CREATE PROCEDURE ステートメント内で OUT として定義されているパラメーターのデータ・タイプを登録します。

   カーソル・タイプの出力パラメーターのデータ・タイプは com.ibm.db2.jcc.DB2Types.CURSOR です。
4. ストアード・プロシージャーを呼び出します。
5. それぞれの OUT カーソル・パラメーターの ResultSet を取得するために CallableStatement.getObject メソッドを呼び出します。

   カーソル・パラメーターに対して CallableStatement.getObject または CallableStatement.getString だけを呼び出すことができます。 CallableStatement.getString を呼び出した場合、パラメーターに関して戻される結果セットに関連した名前が戻されます。

   複数の OUT カーソル・パラメーターがデータ・ソース上で同じカーソルを参照する場合、すべてのパラメーターに関して同じ ResultSet インスタンスが戻されます。
6. それぞれの OUT カーソル・パラメーターに関して ResultSet オブジェクトから行を取得します。
7. ResultSetを閉じます。

自動コミット値が true である場合、カーソル・タイプ出力パラメーターまたはストアード・プロシージャーによって戻される**すべての**結果セットがクローズしている場合に限り、コミット操作が発生します。

カーソル・データ・タイプおよびストアード・プロシージャーの定義は次のとおりです。

```
CREATE TYPE myRowType AS ROW (name VARCHAR(128))
CREATE TYPE myCursorType AS myRowType CURSOR
CREATE PROCEDURE MYPROC(IN pempNo VARCHAR(6), OUT pcv1 myCursorType)
  RESULT SETS 0
  LANGUAGE SQL
  BEGIN
     DECLARE c1 CURSOR WITH RETURN FOR
       SELECT  empno FROM EMPLOYEE;
     OPEN c1;
     SET pcv1 = CURSOR FOR SELECT name FROM employee WHERE empNo = pempNo;
     OPEN pcv1;
  END
```

以下のコードでは、ストアード・プロシージャー MYPROC を呼び出し、ResultSet オブジェクトを使用してカーソル pcv1 からデータを取得します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
Connection con;
ResultSet rs = null;                  // Output parameter              1
...
CallableStatement cstmt = conn.prepareCall("CALL MYPROC(?, ?)");        2
String hvEmpNo="000500";
cstmt.setString (1, hvEmpNo);
cstmt.registerOutParameter (2, DB2Types.CURSOR);                        3
cstmt.executeUpdate();                // Call the stored procedure      4
String hvEmpName = null;
rs = (java.sql.ResultSet)cstmt.getObject(2);                            5
while (rs.next()) {                   // Retrieve result set rows       6
  hvEmpName=rs.getString(1);
  System.out.println("Employee name for " + hvEmpNo
    + ": " + hvEmpName);
}
rs.close();                           // Close the ResultSet            7
```

JDBC アプリケーションでのストアード・プロシージャーからの複数の結果セットの検索

結果セットを返すストアード・プロシージャーを呼び出す場合は、結果セットを取得するためのコードを組み込む必要があります。

実行する手順は、返される結果セットの数や内容がわかっているかどうかによって異なります。

JDBC アプリケーションでストアード・プロシージャーから既知の数の結果セットを取得する:

ストアード・プロシージャーから既知の数の結果セットを取得する手順は、結果セットの数が不明な場合の手順よりもシンプルになります。

結果セットの数と内容がわかっている場合に結果セットを取得するには、以下の手順を実行します。

1. Statement.execute メソッド、PreparedStatement.execute メソッド、CallableStatement.execute メソッドのいずれかを呼び出して、ストアード・プロシージャーを呼び出します。

   ストアード・プロシージャーに入力パラメーターがある場合は、PreparedStatement.execute を使用します。
2. getResultSet メソッドを呼び出して、ResultSet オブジェクトに含まれている最初の結果セットを取得します。
3. ループの中で、next メソッドを使用してカーソルを配置し、getXXX メソッドを使用して、ResultSet オブジェクトの現在行の各列からデータを取得します。
4. 結果セットが *n* 個ある場合は、以下の手順を *n*-1 回繰り返します。
   a. getMoreResults メソッドを呼び出して、現在の結果セットを閉じ、次の結果セットを参照します。
   b. getResultSet メソッドを呼び出して、ResultSet オブジェクトに含まれている次の結果セットを取得します。
   c. ループの中で、next メソッドを使用してカーソルを配置し、getXXX メソッドを使用して、ResultSet オブジェクトの現在行の各列からデータを取得します。

例: 以下のコード例では、2 つの結果セットを取得します。最初の結果セットには INTEGER 列が含まれていて、2 番目の結果セットには CHAR 列が含まれています。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
CallableStatement cstmt;
ResultSet rs;
int i;
String s;
...
cstmt.execute();                          // Call the stored procedure      1
rs = cstmt.getResultSet();                // Get the first result set       2
while (rs.next()) {                       // Position the cursor            3
 i = rs.getInt(1);                        // Retrieve current result set value
 System.out.println("Value from first result set = " + i);
                                          // Print the value
}
cstmt.getMoreResults();                   // Point to the second result set 4a
                                          // and close the first result set
rs = cstmt.getResultSet();                // Get the second result set      4b
while (rs.next()) {                       // Position the cursor            4c
 s = rs.getString(1);                     // Retrieve current result set value
 System.out.println("Value from second result set = " + s);
                                          // Print the value
}
rs.close();                               // Close the result set
cstmt.close();                            // Close the statement
```

JDBC アプリケーションでストアード・プロシージャーから不明な数の結果セットを取得する:

ストアード・プロシージャーから不明な数の結果セットを取得する手順は、結果セットの数がわかっている場合の手順よりも複雑になります。

結果セットの数と内容がわからない場合に結果セットを取得するには、ResultSet が返されなくなるまで ResultSet を取得する必要があります。それぞれの ResultSet ごとに ResultSetMetaData メソッドを使用すれば、結果セットの内容を確認できます。

ストアード・プロシージャーを呼び出した後に、以下の基本手順を実行して、不明な数の結果セットの内容を取得します。

1. ストアード・プロシージャーを呼び出した execute ステートメントから返された値をチェックします。

   戻り値が `true` であれば、少なくとも 1 つの結果セットが存在しているので、次の手順に進む必要があります。
2. ループの中で以下の手順を繰り返します。
   a. getResultSet メソッドを呼び出して、ResultSet オブジェクトに含まれている結果セットを取得します。このメソッドを呼び出すと、前の結果セットが閉じます。
   b. ResultSetMetaData メソッドを使用して、ResultSet の内容を確認し、ResultSet からデータを取得します。
   c. getMoreResults メソッドを呼び出して、さらに結果セットがあるかどうかを確認します。getMoreResults から `true` が返された場合は、手順 1 に進んで次の結果セットを取得します。

例: 結果セットの数と内容がわからない場合に結果セットを取得するためのコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
CallableStatement cstmt;
ResultSet rs;
...
boolean resultsAvailable = cstmt.execute(); // Call the stored procedure
while (resultsAvailable) {                  // Test for result sets       1
 ResultSet rs = cstmt.getResultSet();       // Get a result set           2a
 ...                                          // Process the ResultSet
                                            // as you would process
                                            // a ResultSet from a table
 resultsAvailable = cstmt.getMoreResults(); // Check for next result set  2c
                                            // (Also closes the
                                            // previous result set)
}
```

JDBC アプリケーションでストアード・プロシージャーから複数の結果セットを取得するときに結果セットを開いたままにしておく:

getMoreResults メソッドには、次の ResultSet を開くときに現在の ResultSet を開いたままにしておくための形式があります。

結果セットを開いたままにするかどうかを指定するには、以下の手順を実行します。

getMoreResults を呼び出して、次の ResultSet をチェックするときは、以下の形式を使用します。

```
CallableStatement.getMoreResults(int current);
```

- 次の ResultSet をチェックするときに現在の ResultSet を開いたままにしておくには、*current* に `Statement.KEEP_CURRENT_RESULT` の値を指定します。

- 次の ResultSet をチェックするときに現在の ResultSet を閉じるには、*current* に Statement.CLOSE_CURRENT_RESULT の値を指定します。
- **すべての** ResultSet オブジェクトを閉じるには、*current* に Statement.CLOSE_ALL_RESULTS の値を指定します。

例: 最後の ResultSet を取得するまですべての ResultSet を開いたままにしておき、その後すべての ResultSet を閉じるコードを以下に示します。

```
CallableStatement cstmt;
...
boolean resultsAvailable = cstmt.execute(); // Call the stored procedure
if (resultsAvailable==true) {               // Test for result set
 ResultSet rs1 = cstmt.getResultSet();      // Get a result set
 ...
 resultsAvailable = cstmt.getMoreResults(Statement.KEEP_CURRENT_RESULT);
                                            // Check for next result set
                                            // but do not close
                                            // previous result set
 if (resultsAvailable==true) {              // Test for another result set
  ResultSet rs2 = cstmt.getResultSet();     // Get next result set
  ...                                         // Process either ResultSet
 }
}
resultsAvailable = cstmt.getMoreResults(Statement.CLOSE_ALL_RESULTS);
                                            // Close the result sets
```

IBM Data Server Driver for JDBC and SQLJ 使用時の JDBC アプリケーションでの LOB

IBM Data Server Driver for JDBC and SQLJ は、表の BLOB、CLOB、DBCLOB 列のデータを更新および検索したり、 BLOB パラメーターまたは CLOB パラメーターを指定してストアード・プロシージャーやユーザー定義関数を呼び出したりするためのメソッドをサポートしています。

IBM Data Server Driver for JDBC and SQLJ での連続ストリーミング

連続ストリーミング (動的データ形式ともいう) がデータ・ソースでサポートされる場合は、 IBM Data Server Driver for JDBC and SQLJ で連続ストリーミングを使用して LOB または XML 列内のデータを検索できます。

DB2 for z/OS バージョン 9.1 以降では、LOB および XML オブジェクトの連続ストリーミングがサポートされます。 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、 IBM Informix Dynamic Server (IDS) バージョン 11.50 以降、および DB2 for i V6R1 以降では、LOB の連続ストリーミングがサポートされます。

連続ストリーミングを使用すると、LOB または XML オブジェクトのサイズに基づいて、LOB または XML のデータを戻すのに最も効率のよいモードがデータ・ソースで動的に判別されます。

連続ストリーミングは以下の環境でのデフォルトの動作です。

IBM Data Server Driver for JDBC and SQLJ の最小バージョン	データ・サーバーの最小バージョン	オブジェクトのタイプ
3.53	DB2 for i V6R1	LOB、XML
3.50	DB2 Database for Linux, UNIX, and Windows バージョン 9.5	LOB
3.50	IDS バージョン 11.50	LOB
3.2	DB2 for z/OS バージョン 9	LOB、XML

新規接続での連続ストリーミングの動作は、 IBM Data Server Driver for JDBC and SQLJ progressiveStreaming プロパティーを使用して設定します。

DB2 for z/OS バージョン 9.1 以降のデータ・ソースまたは DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降のデータ・ソースの場合、**DB2Connection.setDBProgressiveStreaming(DB2BaseDataSource.YES)** メソッドを使用して、既存の接続に対して連続ストリーミングの動作を設定できます。**DB2Connection.setDBProgressiveStreaming(DB2BaseDataSource.YES)**を呼び出す場合、接続に関して作成されるすべての ResultSet オブジェクトでは、連続ストリーミングの動作が使われます。

連続ストリーミングを使用可能に設定すると、JDBC ドライバーが streamBufferSize プロパティーで LOB をいつマテリアライズするかをコントロールできます。LOB または XML オブジェクトが streamBufferSize 値以下であるとき、オブジェクトはマテリアライズされます。

重要: 連続ストリーミングでは、LOB または XML の値を ResultSet からアプリケーション変数に取り出す場合、カーソルを移動するまで、または ResultSet 上のカーソルをクローズするまでは、そのアプリケーション変数の内容を操作できます。その後は、アプリケーション変数の内容は操作できなくなります。アプリケーション変数内の LOB に対して何らかのアクションを実行すると、SQLException を受け取ります。例えば、連続ストリーミングが有効で、以下のようなステートメントを実行するとします。

```
...
ResultSet rs = stmt.executeQuery("SELECT CLOBCOL FROM MY_TABLE");
rs.next();                // Retrieve the first row of the ResultSet
Clob clobFromRow1  = rs.getClob(1);
                          // Put the CLOB from the first column of
                          // the first row in an application variable
String substr1Clob = clobFromRow1.getSubString(1,50);
                          // Retrieve the first 50 bytes of the CLOB
rs.next();                // Move the cursor to the next row.
                          // clobFromRow1 is no longer available.
// String substr2Clob = clobFromRow1.getSubString(51,100);
                          // This statement would yield an SQLException
Clob clobFromRow2  = rs.getClob(1);
                          // Put the CLOB from the first column of
                          // the second row in an application variable
rs.close();               // Close the ResultSet.
                          // clobFromRow2 is also no longer available.
```

rs.next() を実行してカーソルを ResultSet の 2 番目の行に置いた後は、clobFromRow1 内の CLOB 値は使用できなくなります。同様に、rs.close() を実行して ResultSet をクローズした後、clobFromRow1 および clobFromRow2 内の値は使用できなくなります。

連続ストリーミングを無効として設定した場合、 IBM Data Server Driver for JDBC and SQLJ が LOB を扱う方法は、fullyMaterializeLobData プロパティーの値によって決まります。

連続ストリーミングの使用は、LOB または XML データ検索に推奨される方式です。

IBM Data Server Driver for JDBC and SQLJ での LOB ロケーター

IBM Data Server Driver for JDBC and SQLJ では、LOB ロケーターを使用して LOB 列のデータを検索できます。

JDBC で LOB ロケーターを使用して、LOB 列からデータを検索するには、fullyMaterializeLobData プロパティーを false に設定し、progressiveStreaming プロパティーを NO (アプリケーション・プログラム内では DB2BaseDataSource.NO) に設定する必要があります。

fullyMaterializeLobData の効果は、次のように、データ・ソースが連続ストリーミングをサポートするかどうか、および progressiveStreaming プロパティーの値によって異なります。

- データ・ソースが連続ロケーターをサポートしない場合:

  fullyMaterializeLobData の値が true の場合、行がフェッチされるときに LOB データは JDBC ドライバー内で完全にマテリアライズされます。この値が false の場合、LOB データはストリームされます。ドライバーは内部でロケーターを使用し、必要に応じて LOB データをチャンクごとに検索します。大量のデータを収容している LOB を検索する場合、この値を false に設定するように強くお勧めします。デフォルトは true です。
- データ・ソースが連続ストリーミング (動的データ形式ともいう) をサポートする場合、以下のようになります。

  progressiveStreaming プロパティーが YES (アプリケーション・プログラム内では DB2BaseDataSource.YES) に設定されているか、または設定されていない場合、JDBC ドライバーは fullyMaterializeLobData の値を無視します。

fullyMaterializeLobData は、ストアード・プロシージャー・パラメーターには影響を与えません。

他の言語の場合と同様、Java アプリケーションにおける LOB ロケーターは、単一のデータ・ソースと関連付けられています。単一の LOB ロケーターを使用して、2 つの異なるデータ・ソースの間でデータを移動させることはできません。 LOB データを 2 つのデータ・ソース間で移動するには、最初のデータ・ソースにある表から LOB データを検索するときにそれをマテリアライズし、次いでそのデータを 2 番目のデータ・ソースにある表に挿入する必要があります。

IBM Data Server Driver for JDBC and SQLJ 使用時の LOB 操作

IBM Data Server Driver for JDBC and SQLJ は、表の BLOB、CLOB、DBCLOB 列のデータを更新および検索したり、 BLOB パラメーターまたは CLOB パラメーターを指定してストアード・プロシージャーやユーザー定義関数を呼び出したりするためのメソッドをサポートしています。

IBM Data Server Driver for JDBC and SQLJ で LOB データに関して実行できる操作は次のとおりです。

- BLOB 列または CLOB 列からデータを検索するには、BLOB 列または CLOB 列を以下の ResultSet メソッドの引数として指定します。

  BLOB 列の場合:
  - getBinaryStream
  - getBlob
  - getBytes

  CLOB 列の場合:
  - getAsciiStream
  - getCharacterStream
  - getClob
  - getString
- 更新可能な ResultSet で BLOB 列または CLOB 列を更新するには、以下の ResultSet メソッドを呼び出します。

  BLOB 列の場合:
  - updateBinaryStream
  - updateBlob

  CLOB 列の場合:
  - updateAsciiStream
  - updateCharacterStream
  - updateClob

  上にリストしたいずれかのメソッドで *length* パラメーターに -1 を指定すると、IBM Data Server Driver for JDBC and SQLJ は、入力データがなくなるまでデータを読み取ります。
- 以下の PreparedStatement メソッドを使用すると、 BLOB 列または CLOB 列に対応するパラメーターの値が設定されます。

  BLOB 列の場合:
  - setBytes
  - setBlob
  - setBinaryStream
  - setObject (*Object* パラメーターの値は InputStream)

  CLOB 列の場合:
  - setString
  - setAsciiStream

- setClob
- setCharacterStream
- setObject (*Object* パラメーター値は Reader)。

length に -1 を指定すると、 IBM Data Server Driver for JDBC and SQLJ は、入力データがなくなるまでデータを読み取ります。

- `CallableStatement.getString` メソッドを使用して、JDBC CLOB パラメーターの値を検索します。

制約事項: IBM Data Server Driver for JDBC and SQLJ Type 2 接続では、DBCLOB OUT または INOUT パラメーターを持つストアード・プロシージャーを呼び出すことはできません。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降を使用する場合は、このほかに以下の操作を実行できます。

- ResultSet.updateXXX メソッドまたは PreparedStatement.setXXX メソッドで、BLOB または CLOB を更新します。*length* 値に指定できるのは、最大で 2 GB です。例えば、BLOB には以下のメソッドが定義されます。

```
ResultSet.updateBlob(int columnIndex, InputStream x, long length)
ResultSet.updateBlob(String columnLabel, InputStream x, long length)
ResultSet.updateBinaryStream(int columnIndex, InputStream x, long length)
ResultSet.updateBinaryStream(String columnLabel, InputStream x, long length)
PreparedStatement.setBlob(int columnIndex, InputStream x, long length)
PreparedStatement.setBlob(String columnLabel, InputStream x, long length)
PreparedStatement.setBinaryStream(int columnIndex, InputStream x, long length)
PreparedStatement.setBinaryStream(String columnLabel, InputStream x, long length)
```

- BLOB または CLOB の更新時に *length* パラメーターを指定せずに ResultSet.updateXXX メソッドまたは PreparedStatement.setXXX メソッドを実行すると、 IBM Data Server Driver for JDBC and SQLJ は入力データがなくなるまでデータを読み取ります。以下に例を示します。

```
ResultSet.updateBlob(int columnIndex, InputStream x)
ResultSet.updateBlob(String columnLabel, InputStream x)
ResultSet.updateBinaryStream(int columnIndex, InputStream x)
ResultSet.updateBinaryStream(String columnLabel, InputStream x)
PreparedStatement.setBlob(int columnIndex, InputStream x)
PreparedStatement.setBlob(String columnLabel, InputStream x)
PreparedStatement.setBinaryStream(int columnIndex, InputStream x)
PreparedStatement.setBinaryStream(String columnLabel, InputStream x)
```

- Connection.createBlob メソッドまたは Connection.createClob メソッドを使用して、データを含まない Blob オブジェクトまたは Clob オブジェクトを作成します。
- Blob.getBinaryStream メソッドまたは Clob.getCharacterStream メソッドを使用して、連続ストリーミングまたはロケーターの使用中に、クライアント上の Blob オブジェクトまたは `Clob` オブジェクトをマテリアライズします。
- Blob.free メソッドまたは Clob.free メソッドで、 Blob オブジェクトまたは Clob オブジェクトが保持するリソースを解放します。

JDBC アプリケーションでの LOB 列データの検索または更新のための Java データ・タイプ

JDBC ドライバーで、LOB 列とともに使用されるパラメーターのデータ・タイプがすぐに判別できない場合は、LOB データ・タイプと互換性のあるパラメーターのデータ・タイプを選択する必要があります。

deferPrepares プロパティーが true に設定され、 IBM Data Server Driver for JDBC and SQLJ により PreparedStatement.setXXX 呼び出しが処理される場合、ドライバーにデータ・タイプの判別のための追加的な処理が必要になることがあります。この追加処理は、パフォーマンスに影響を与えるおそれがあります。

BLOB 列の入力パラメーター

BLOB 列の IN パラメーター、または BLOB 列への入力に使用される INOUT パラメーターの場合、以下の技法のいずれかを使用できます。

- 次のように、BLOB 列と完全に一致する java.sql.Blob 入力変数を使用します。

  ```
  cstmt.setBlob(parmIndex, blobData);
  ```

- 次のように、ターゲット・データ・タイプとして BLOB を指定する CallableStatement.setObject 呼び出しを使用します。

  ```
  byte[] byteData = {(byte)0x1a, (byte)0x2b, (byte)0x3c};
  cstmt.setObject(parmInd, byteData, java.sql.Types.BLOB);
  ```

- CallableStatement.setBinaryStream 呼び出しで、java.io.ByteArrayInputStream タイプの入力パラメーターを使用します。 java.io.ByteArrayInputStream オブジェクトは、BLOB データ・タイプと互換性があります。この呼び出しの場合、入力データの正確な長さを指定する必要があります。

  ```
  java.io.ByteArrayInputStream byteStream =
    new java.io.ByteArrayInputStream(byteData);
  int numBytes = byteData.length;
  cstmt.setBinaryStream(parmIndex, byteStream, numBytes);
  ```

BLOB 列の出力パラメーター

BLOB 列の OUT パラメーター、または BLOB 列からの出力に使用される INOUT パラメーターの場合、次の技法を使用できます。

- CallableStatement.registerOutParameter 呼び出しを使用して、出力パラメーターを BLOB タイプに指定します。これにより、パラメーター値を BLOB データ・タイプと互換性のあるデータ・タイプのすべての変数に取り出すことができます。例えば、次のコードでは BLOB 値が byte[] 変数に取り出されます。

  ```
  cstmt.registerOutParameter(parmIndex, java.sql.Types.BLOB);
  cstmt.execute();
  byte[] byteData = cstmt.getBytes(parmIndex);
  ```

CLOB 列の入力パラメーター

CLOB 列の IN パラメーター、または CLOB 列への入力に使用される INOUT パラメーターの場合、以下の技法のいずれかを使用できます。

- 次のように、CLOB 列と完全に一致する java.sql.Clob 入力変数を使用します。

  ```
  cstmt.setClob(parmIndex, clobData);
  ```

- 次のように、ターゲット・データ・タイプとして CLOB を指定する CallableStatement.setObject 呼び出しを使用します。

```
String charData = "CharacterString";
cstmt.setObject(parmInd, charData, java.sql.Types.CLOB);
```

- 以下のいずれかのタイプのストリーム入力パラメーターを使用します。
  - cstmt.setCharacterStream 呼び出しでの java.io.StringReader 入力パラメーター:

```
java.io.StringReader reader = new java.io.StringReader(charData);
cstmt.setCharacterStream(parmIndex, reader, charData.length);
```

  - cstmt.setAsciiStream 呼び出しでの java.io.ByteArrayInputStream パラメーター (ASCII データ用):

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
cstmt.setAsciiStream(parmIndex, byteStream, charDataBytes.length);
```

  これらの呼び出しでは、入力データの正確な長さを指定する必要があります。
- 次のように、cstmt.setString 呼び出しで String の入力パラメーターを使用します。

```
cstmt.setString(parmIndex, charData);
```

  データの長さが 32 KB より大きく、JDBC ドライバーにこのパラメーターのデータ・タイプに関する DESCRIBE 情報がない場合は、JDBC ドライバーにより CLOB データ・タイプが入力データに割り当てられます。
- 次のように、cstmt.setObject 呼び出しで String の入力パラメーターを使用し、ターゲット・データ・タイプを VARCHAR または LONGVARCHAR で指定します。

```
cstmt.setObject(parmIndex, charData, java.sql.Types.VARCHAR);
```

  データの長さが 32 KB より大きく、JDBC ドライバーにこのパラメーターのデータ・タイプに関する DESCRIBE 情報がない場合は、JDBC ドライバーにより CLOB データ・タイプが入力データに割り当てられます。

CLOB 列の出力パラメーター

CLOB 列の OUT パラメーター、または CLOB 列からの出力に使用される INOUT パラメーターの場合、以下の技法のいずれかを使用できます。

- CallableStatement.registerOutParameter 呼び出しを使用して、出力パラメーターを CLOB タイプに指定します。これにより、パラメーター値を Clob 変数に取り出すことができます。以下に例を示します。

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.CLOB);
cstmt.execute();
Clob clobData = cstmt.getClob(parmIndex);
```

- CallableStatement.registerOutParameter 呼び出しを使用して、出力パラメーターを VARCHAR または LONGVARCHAR タイプに指定します。

```
cstmt.registerOutParameter(parmIndex, java.sql.Types.VARCHAR);
cstmt.execute();
String charData = cstmt.getString(parmIndex);
```

この技法は、取り出されるデータの長さが 32 KB 以下であることが分かっている場合にのみ使用してください。 32 KB を超える場合、データは切り捨てられます。

IBM Data Server Driver for JDBC and SQLJ 使用時の JDBC での ROWID

DB2 for z/OSおよび DB2 for i は、データベース表内の列に対して ROWID データ・タイプをサポートします。ROWID とは、表内の行を一意的に識別する値です。

IBM Informix Dynamic Server (IDS) では ROWID もサポートされますが、それらの ROWID のデータ・タイプは INTEGER です。 4 バイトの整数データ・タイプを使用して IDS ROWID 列を変数に選択することができます。

以下の ResultSet メソッドを使用して、データを ROWID 列から取得することができます。
- getRowId (JDBC 4.0 以降)
- getBytes
- getObject

次の ResultSet メソッドを使用して、更新可能な ResultSet の ROWID 列を更新することができます。
- updateRowId (JDBC 4.0 以降)

  updateRowId は、ターゲット・データベース・システムで ROWID 列の更新がサポートされる場合にのみ有効です。

JDBC 3.0 を使用している場合、getObject では、 IBM Data Server Driver for JDBC and SQLJ により、 IBM Data Server Driver for JDBC and SQLJ のみのクラスである com.ibm.db2.jcc.DB2RowID のインスタンスが返されます。

JDBC 4.0 を使用している場合、getObject では、 IBM Data Server Driver for JDBC and SQLJ により、クラス java.sql.RowId のインスタンスが返されます。

以下の PreparedStatement メソッドを使用して、ROWID 列に関連付けられたパラメーターの値を設定することができます。
- setRowId (JDBC 4.0 以降)
- setBytes
- setObject

JDBC 3.0 を使用している場合、setObject では、パラメーターのターゲット・タイプとして、 IBM Data Server Driver for JDBC and SQLJ のみのタイプである `com.ibm.db2.jcc.Types.ROWID` または `com.ibm.db2.jcc.DB2RowID` クラスのインスタンスを使用します。

JDBC 4.0 を使用している場合、setObject では、パラメーターのターゲット・タイプとして、タイプ `java.sql.Types.RowId` または `java.sql.ROWID` クラスのインスタンスを使用します。

以下の CallableStatement メソッドを使用して、ROWID 列をストアード・プロシージャー呼び出しからの出力パラメーターとして取得することができます。
- getRowId (JDBC 4.0 以降)
- getObject

ROWID 出力パラメーターで定義されたストアード・プロシージャーを呼び出すには、そのパラメーターを java.sql.Types.ROWID タイプで登録します。

ROWID 値が有効な期間は、それらの ROWID 値が定義されたデータ・ソースによって異なります。 DatabaseMetaData.getRowIdLifetime メソッドを使用して、ROWID 値が有効な期間を判別します。データ・ソースに返される値が、以下の表にリストされています。

表 13. サポートされるデータ・ソースの DatabaseMetaData.getRowIdLifetime 値

データベース・サーバー	DatabaseMetaData.getRowIdLifetime
DB2 for z/OS	ROWID_VALID_TRANSACTION
DB2 Database for Linux, UNIX, and Windows	ROWID_UNSUPPORTED
DB2 for i	ROWID_VALID_FOREVER
IDS	ROWID_VALID_FOREVER

例: PreparedStatement.setRowId の java.sql.RowId ターゲット・タイプでの使用: rwid が RowId オブジェクトであると仮定します。パラメーター 1 を設定するには、次の形式の setRowId メソッドを使用します。

```
ps.setRowId(1, rid);
```

例: ROWID 値をデータ・ソースから取得するための ResultSet.getRowId の使用: ROWID 値を結果セットの最初の列から RowId オブジェクトの rwid に取り出すには、次の形式の ResultSet.getRowId メソッドを使用します。

```
java.sql.RowId rwid = rs.getRowId(1);
```

例: CallableStatement.registerOutParameter の java.sql.Types.ROWID パラメーター・タイプでの使用: CALL ステートメントのパラメーター 1 を java.sql.Types.ROWID データ・タイプで登録するには、次の形式の registerOutParameter メソッドを使用します。

```
cs.registerOutParameter(1, java.sql.Types.ROWID)
```

JDBC アプリケーションでの特殊タイプ

特殊タイプとは、組み込みの SQL データ・タイプとして内部的に表されるユーザー定義のデータ・タイプのことです。特殊タイプは、SQL ステートメントの CREATE DISTINCT TYPE を実行して作成します。

JDBC プログラムでは、特殊タイプは、executeUpdate メソッドを使用して CREATE DISTINCT TYPE ステートメントを実行することで作成することができます。また、executeUpdate を使用して、このタイプの列が含まれた表を作成することもできます。このタイプの列からデータを取得するか、またはこのタイプの列を更新する場合は、特殊タイプのベースとなる組み込みタイプに対応するデータ・タイプと共に、Java ID を使用します。

以下の例では、INTEGER タイプをベースにした特殊タイプを作成し、このタイプの列を持つ表を作成して、行をその表に挿入し、さらにその表から行を取得します。

```
Connection con;
Statement stmt;
ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
stmt = con.createStatement();                // Create a Statement object
stmt.executeUpdate(
  "CREATE DISTINCT TYPE SHOESIZE AS INTEGER");
                                             // Create distinct type
stmt.executeUpdate(
  "CREATE TABLE EMP_SHOE (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)");
                                             // Create table with distinct type
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");                   // Insert a row
rs=stmt.executeQuery("SELECT EMPNO, EMP_SHOE_SIZE FROM EMP_SHOE);
                                             // Create ResultSet for query
while (rs.next()) {
  empNumVar = rs.getString(1);               // Get employee number
  shoeSizeVar = rs.getInt(2);                // Get shoe size (use int
                                             // because underlying type
                                             // of SHOESIZE is INTEGER)
  System.out.println("Employee number = " + empNumVar +
    " Shoe size = " + shoeSizeVar);
}
rs.close();                                  // Close ResultSet
stmt.close();                                // Close Statement
```

図 16. 特殊タイプの作成および使用

JDBC アプリケーションでの ARRAY パラメーターを使用したストアード・プロシージャーの呼び出し

IBM Data Server Driver for JDBC and SQLJ で実行される JDBC アプリケーションでは、ARRAY パラメーターを持つストアード・プロシージャーを呼び出すことができます。 ARRAY パラメーターは、 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降のストアード・プロシージャーでサポートされています。

java.sql.Array オブジェクトは、ストアード・プロシージャーで IN、OUT、または INOUT パラメーターとして使用できます。

IN または INOUT パラメーターの場合、DB2Connection.createArrayOf メソッド (JDBC 3.0 またはそれ以前) または Connection.createArrayOf メソッド (JDBC 4.0 またはそれ以降) を使用して、java.sql.Array オブジェクトを作成します。 CallableStatement.setArray メソッドまたは CallableStatement.setObject メソッドを使用して、java.sql.Array オブジェクトを ARRAY ストアード・プロシージャーのパラメーターに割り当てることができます。

CallableStatement.registerOutParameter 呼び出しでパラメーター・タイプとして java.sql.Types.ARRAY を指定することによって、ストアード・プロシージャー呼び出し用の OUT ARRAY パラメーターを登録することができます。

ARRAY 出力パラメーターからデータを取り出すには、次の 2 つの方法があります。

- CallableStatement.getArray メソッドを使用して、データを java.sql.Array オブジェクトに取り出し、java.sql.Array.getArray メソッドを使用して、java.sql.Array オブジェクトの内容を Java 配列に取り出します。
- CallableStatement.getArray メソッドを使用して、データを java.sql.Array オブジェクトに取り出します。java.sql.Array.getResultSet() メソッドを使用して、データを ResultSet オブジェクトに取り出します。ResultSet メソッドを使用して、配列のエレメントを取り出します。 ResultSet の各行には、次の 2 つの列が含まれています。
  - 配列の指標 (1 から始まります)
  - 配列エレメント

例: ストアード・プロシージャー GET_EMP_DATA の入力パラメーター IN_PHONE と出力パラメーター OUT_PHONE が以下のように定義される配列だとします。

```
CREATE TYPE PHONENUMBERS AS VARCHAR(10) ARRAY[5]
```

GET_EMP_DATA をこの 2 つのパラメーターで呼び出します。

```
Connection con;
CallableStatement cstmt;
ResultSet rs;
java.sql.Array inPhoneData;
...
stmt = con.prepareCall("CALL GET_EMP_DATA(?,?)");
                                  // Create a CallableStatement object
cstmt.setObject (1, inPhoneData); // Set input parameter
cstmt.registerOutParameter (2, java.sql.Types.ARRAY);
                                  // Register out parameters
cstmt.executeUpdate();            // Call the stored procedure
Array outPhoneData = cstmt.getArray(2);
                                  // Get the output parameter array
System.out.println("Parameter values from GET_EMP_DATA call: ");
String [] outPhoneNums = (String [])outPhoneData.getArray();
                          // Retrieve output data from the JDBC Array object
                          // into a Java String array
for(int i=0; i<outPhoneNums.length; i++) {
  System.out.print(outPhoneNums[i]);
  System.out.println();
}
```

JDBC アプリケーションでのセーブポイント

SQL セーブポイントは、作業単位内の特定の時点のデータおよびスキーマの状態を表します。セーブポイントの設定、セーブポイントの解放、およびデータとスキーマのセーブポイントによって表された状態へのリストアを行うための SQL ステートメントがあります。

IBM Data Server Driver for JDBC and SQLJ では、セーブポイントを使用するための以下のメソッドがサポートされます。

Connection.setSavepoint() または Connection.setSavepoint(String *name*)
セーブポイントが設定されます。これらのメソッドにより、この後の releaseSavepoint または rollback 操作で使用される Savepoint オブジェクトが返されます。

これらのメソッドのいずれかを実行すると、 DB2 により ON ROLLBACK RETAIN CURSORS などの SAVEPOINT ステートメントの形式が実行されます。

Connection.releaseSavepoint(Savepoint *savepoint*)
指定したセーブポイント、およびそれ以後に確立されたすべてのセーブポイントが解放されます。

Connection.rollback(Savepoint *savepoint*)
作業が指定したセーブポイントまでロールバックされます。

DatabaseMetaData.supportsSavepoints()
データ・ソースでセーブポイントがサポートされているかどうかが示されます。

メソッド DB2Connection.setSavePointUniqueOption を呼び出すことにより、セーブポイントが固有かどうかを示すことができます。値を `true` にしてこのメソッドを呼び出した場合、アプリケーションで同じリカバリー単位内に同じ名前のセーブポイントを複数設定することはできません。値を `false` (デフォルト) にしてこのメソッドを呼び出すと、同じリカバリー単位内に同じ名前のセーブポイントを複数作成できますが、セーブポイントを作成すると、同じ名前で既に作成されているセーブポイントが破棄されます。

以下の例は、セーブポイントの設定、セーブポイントへのロールバック、およびセーブポイントの解除の方法を示しています。

```
Connection con;
Statement stmt;
ResultSet rs;
String empNumVar;
int shoeSizeVar;
...
con.setAutoCommit(false);                    // set autocommit OFF
stmt = con.createStatement();                // Create a Statement object
...                                            // Perform some SQL
con.commit();                                // Commit the transaction
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000010', 6)");                   // Insert a row
((com.ibm.db2.jcc.DB2Connection)con).setSavePointUniqueOption(true);
                                             // Indicate that savepoints
                                             // are unique within a unit
                                             // of recovery
Savepoint savept = con.setSavepoint("savepoint1");
                                             // Create a savepoint
...
stmt.executeUpdate("INSERT INTO EMP_SHOE " +
  "VALUES ('000020', 10)");                  // Insert another row
conn.rollback(savept);                       // Roll back work to the point
                                             //  after the first insert
...
con.releaseSavepoint(savept);                // Release the savepoint
stmt.close();                                // Close the Statement
conn.commit();                               // Commit the transaction
```

図 17. JDBC アプリケーションでのセーブポイントの設定、ロールバック、および解除

JDBC アプリケーションでの自動生成キーの取得

IBM Data Server Driver for JDBC and SQLJ を使用すれば、JDBC 3.0 のメソッドによって自動生成キーを表から取得できます。

自動生成キーは、ID 列の内容に対応しています。 ID 列は、データ・ソースが各行の数値を自動生成するための手段になる表列です。CREATE TABLE ステートメントまたは ALTER TABLE ステートメントで ID 列を定義するには、スケール 0 の厳密な数値タイプ (スケール 0 の SMALLINT、INTEGER、BIGINT、DECIMAL、またはそのいずれかのタイプに基づく特殊タイプ) の列を定義するときに AS IDENTITY 節を指定します。

表から自動生成キーを取得する操作を有効にするには、行を挿入するときに、自動生成キー値を取得できるように設定する必要があります。そのためには、Connection.prepareStatement、Statement.executeUpdate、Statement.execute のいずれかのメソッド呼び出しで 1 つのフラグを設定します。実行するステートメントは、単一行の INSERT ステートメントまたは複数行の INSERT ステートメント (副選択節を含む INSERT など) でなければなりません。それ以外の場合は、JDBC ドライバーがそのフラグを設定するためのパラメーターを無視します。

制約事項: Connection または DataSource プロパティー atomicMultiRowInsert を `DB2BaseDataSource.YES` (1) に設定すると、自動生成キーの取得のために SQL ステートメントを準備して、その PreparedStatement オブジェクトをバッチ更新で使用することはできません。 IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 以降では、自動生成キーを取得するために準備した PreparedStatement オブジェクトに対して addBatch メソッドまたは executeBatch メソッドを呼び出すと、SQLException がスローされます。

表から自動生成キーを取得するには、以下の手順を実行する必要があります。

1. 以下のいずれかのメソッドを使用して、自動生成キーを返すための設定を行います。
   - PreparedStatement.executeUpdate メソッドを使用して行を挿入する計画がある場合は、以下のいずれかの形式の Connection.prepareStatement メソッドを呼び出して、PreparedStatement オブジェクトを作成します。

     ID 列をサポートしているデータ・ソースの表であれば、以下の形式が有効です。*sql-statement* は、単一行の INSERT ステートメントでなければなりません。

     ```
     Connection.prepareStatement(sql-statement,
       Statement.RETURN_GENERATED_KEYS);
     ```

     以下の形式は、データ・ソースが SELECT ステートメント内の INSERT をサポートしている場合にのみ有効です。 *sql-statement* は、単一行の INSERT ステートメントでも複数行の INSERT ステートメントでもかまいません。最初の形式の場合は、自動生成キーを取得する列の名前を指定します。2 番目の形式の場合は、自動生成キーを取得する列の、表の中での位置を指定します。

     ```
     Connection.prepareStatement(sql-statement, String [] columnNames);
     Connection.prepareStatement(sql-statement, int [] columnIndexes);
     ```

   - Statement.executeUpdate メソッドを使用して行を挿入する場合は、以下のいずれかの形式の Statement.executeUpdate メソッドを呼び出します。

ID 列をサポートしているデータ・ソースの表であれば、以下の形式が有効です。*sql-statement* は、単一行の INSERT ステートメントでなければなりません。

```
Statement.executeUpdate(sql-statement, Statement.RETURN_GENERATED_KEYS);
```

以下の形式は、データ・ソースが SELECT ステートメント内の INSERT をサポートしている場合にのみ有効です。 *sql-statement* は、単一行の INSERT ステートメントでも複数行の INSERT ステートメントでもかまいません。最初の形式の場合は、自動生成キーを取得する列の名前を指定します。2 番目の形式の場合は、自動生成キーを取得する列の、表の中での位置を指定します。

```
Statement.executeUpdate(sql-statement, String [] columnNames);
Statement.executeUpdate(sql-statement, int [] columnIndexes);
```

2. PreparedStatement.getGeneratedKeys メソッドまたは Statement.getGeneratedKeys メソッドを呼び出して、自動生成キー値が含まれている ResultSet オブジェクトを取得します。

   Statement.RETURN_GENERATED_KEYS パラメーターを組み込む場合、ResultSet に含まれている自動生成キーのデータ・タイプは、対応する列のデータ・タイプに関わりなく DECIMAL になります。

ID 列が含まれている表を作成し、その表に 1 つの行を挿入し、ID 列の自動生成キー値を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
import java.sql.*;
import java.math.*;
import com.ibm.db2.jcc.*;

Connection con;
Statement stmt;
ResultSet rs;
java.math.BigDecimal iDColVar;
...
stmt = con.createStatement();         // Create a Statement object

stmt.executeUpdate(
  "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
    "IDENTCOL INTEGER GENERATED ALWAYS AS IDENTITY)");
                                      // Create table with identity column
stmt.executeUpdate("INSERT INTO EMP_PHONE (EMPNO, PHONENO) " +          1
  "VALUES ('000010', '5555')",        // Insert a row
  Statement.RETURN_GENERATED_KEYS);   // Indicate you want automatically
                                      // generated keys
rs = stmt.getGeneratedKeys();         // Retrieve the automatically      2
                                      // generated key value in a ResultSet.
                                      // Only one row is returned.
                                      // Create ResultSet for query
while (rs.next()) {
  java.math.BigDecimal idColVar = rs.getBigDecimal(1);
                                      // Get automatically generated key
                                      // value
  System.out.println("automatically generated key value = " + idColVar);
}
rs.close();                           // Close ResultSet
stmt.close();                         // Close Statement
```

ID 列が含まれている表を作成し、複数行の INSERT ステートメントを使用してその表に 2 つの行を挿入し、ID 列の自動生成キー値を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
import java.sql.*;
import java.math.*;
import com.ibm.db2.jcc.*;

Connection con;
Statement stmt;
ResultSet rs;
...
stmt = con.createStatement();

stmt.executeUpdate(
  "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
  "IDENTCOL INTEGER GENERATED ALWAYS AS IDENTITY)");
                                    // Create table with identity column
String[] id_col = {"IDENTCOL"};
int updateCount =                                                    1
  stmt.executeUpdate("INSERT INTO EMP_PHONE (EMPNO, PHONENO)" +
  "VALUES ('000010', '5555'), ('000020', '5556')", id_col);
                                    // Insert two rows
                                    // Indicate you want automatically
                                    // generated keys
rs = stmt.getGeneratedKeys();       // Retrieve the automatically       2
                                    // generated key values in a ResultSet.
                                    // Two rows are returned.
                                    // Create ResultSet for query
while (rs.next()) {
 int idColVar = rs.getInt(1);
                                    // Get automatically generated key
                                    // values
 System.out.println("automatically generated key value = " + idColVar);
}
stmt.close();
con.close();
```

JDBC アプリケーションでの名前付きパラメーター・マーカーの使用

PreparedStatement および CallableStatement オブジェクトで標準のパラメーター・マーカーの代わりに名前付きパラメーター・マーカーを使用して、入力パラメーター・マーカーに値を割り当てることができます。 CallableStatement オブジェクトで標準のパラメーター・マーカーの代わりに名前付きパラメーター・マーカーを使用して、名前付き OUT パラメーターを登録することもできます。

名前付きパラメーター・マーカーを使用すれば、JDBC アプリケーションをより分かりやすいものにできます。データ・ソースで名前付きパラメーター・マーカーがサポートされていない場合、 IBM Data Server Driver for JDBC and SQLJenableNamedParameterMarkers プロパティーを DB2BaseDataSource.YES (1) に設定することにより、ドライバーが名前付きパラメーター・マーカーを受け入れ、それらを標準のパラメーター・マーカー (疑問符 (?) として示される) としてデータ・ソースに送信するようにできます。

PreparedStatement オブジェクトでの名前付きパラメーター・マーカーの使用

PreparedStatement オブジェクトで標準のパラメーター・マーカーの代わりに名前付きパラメーター・マーカーを使用して、パラメーター・マーカーに値を割り当てることができます。

名前付きパラメーター・マーカーを PreparedStatement オブジェクトで使用するには、以下の手順に従います。

1. 名前付きパラメーター・マーカーを含む SQL ステートメント・ストリングに対して Connection.prepareStatement メソッドを実行します。 名前付きパラメーター・マーカーは、SQL ホスト変数名の規則に従う必要があります。

   同じ SQL ステートメント・ストリングの中で名前付きパラメーター・マーカーと標準のパラメーター・マーカーを混用することはできません。

   名前付きパラメーター・マーカーは大/小文字を区別しません。
2. それぞれの名前付きパラメーター・マーカーに対して DB2PreparedStatement.setJccXXXAtName メソッドを使用して、それぞれの名前付き入力パラメーターに値を割り当てます。

   同じ SQL ステートメント・ストリング内で同じ名前付きパラメーター・マーカーを複数回にわたって使用する場合、そのパラメーター・マーカーに対する setJccXXXAtName メソッドを一度だけ呼び出す必要があります。

   推奨: パラメーター・マーカーへの入力がストリームである場合には、同じ SQL ステートメント・ストリング内でその同じ名前付きパラメーター・マーカーを複数回にわたって使用しないでください。予期しない結果が発生する可能性があります。

   制約事項: 標準の JDBC PreparedStatement.setXXX メソッドを名前付きパラメーター・マーカーで使用することはできません。これが原因で例外がスローされます。
3. PreparedStatement を実行します。

以下のコードでは、名前付きパラメーター・マーカーを使用して、従業員番号「000010」の従業員の電話番号を「4657」に更新します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
Connection con;
PreparedStatement pstmt;
int numUpd;
...
pstmt = con.prepareStatement(
  "UPDATE EMPLOYEE SET PHONENO=:phonenum WHERE EMPNO=:empnum");
                                  // Create a PreparedStatement object     1
((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).setJccStringAtName
  ("phonenum", "4567");
                                  // Assign a value to phonenum parameter  2
((com.ibm.db2.jcc.DB2PreparedStatement)pstmt).setJccStringAtName
  ("empnum", "000010");
                                  // Assign a value to empnum parameter
numUpd = pstmt.executeUpdate();   // Perform the update                     3
pstmt.close();                    // Close the PreparedStatement object
```

CallableStatement オブジェクトでの名前付きパラメーター・マーカーの使用

CallableStatement オブジェクトで標準のパラメーター・マーカーの代わりに名前付きパラメーター・マーカーを使用して、IN または INOUT パラメーターに値を割り当てたり、OUT パラメーターを登録したりすることができます。

名前付きパラメーター・マーカーを CallableStatement オブジェクトで使用するには、以下の手順に従います。

1. 名前付きパラメーター・マーカーを含む SQL ステートメント・ストリングに対して Connection.prepareCall メソッドを実行します。

   名前付きパラメーター・マーカーは、SQL ホスト変数名の規則に従う必要があります。

   同じ SQL ステートメント・ストリングの中で名前付きパラメーター・マーカーと標準のパラメーター・マーカーを混用することはできません。

   名前付きパラメーター・マーカーは大/小文字を区別しません。
2. OUT パラメーターを表すそれぞれの名前付きパラメーター・マーカーに対して、DB2CallableStatement.registerJccOutParameterAtName メソッドを使って OUT パラメーターのデータ・タイプを登録します。

   同じ SQL ステートメント・ストリング内で同じ名前付きパラメーター・マーカーを複数回にわたって使用する場合、そのパラメーター・マーカーに対する registerJccOutParameterAtName メソッドを一度だけ呼び出す必要があります。同じ名前を持つすべてのパラメーターは、同じデータ・タイプとして登録されます。

   制約事項: 標準の JDBC CallableStatement.registerOutParameter メソッドを名前付きパラメーター・マーカーで使用することはできません。これが原因で例外がスローされます。
3. 入力パラメーターであるそれぞれの名前付きパラメーター・マーカーに対して DB2CallableStatement.setJccXXXAtName メソッドを使用して、それぞれの名前付き入力パラメーターに値を割り当てます。

   setJccXXXAtName メソッドは DB2PreparedStatement から継承します。

   同じ SQL ステートメント・ストリング内で同じ名前付きパラメーター・マーカーを複数回にわたって使用する場合、そのパラメーター・マーカーに対する setJccXXXAtName メソッドを一度だけ呼び出す必要があります。

   推奨: パラメーター・マーカーへの入力がストリームである場合には、同じ SQL ステートメント・ストリング内でその同じ名前付きパラメーター・マーカーを複数回にわたって使用しないでください。予期しない結果が発生する可能性があります。
4. CallableStatement を実行します。
5. 出力パラメーター値を取得するために CallableStatement.getXXX メソッドを呼び出します。

以下のコードで呼び出されるストアード・プロシージャーには 1 つの入力 VARCHAR パラメーターと 1 つの出力 INTEGER パラメーターがあり、これらは名前付きパラメーター・マーカーで表されます。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
...
CallableStatement cstmt =
  con.prepareCall("CALL MYSP(:inparm,:outparm)");
                                    // Create a CallableStatement object  1
                                    // Register OUT parameter data type   2
((com.ibm.db2.jcc.DB2CallableStatement)cstmt).
   registerOutParameterAtName("outparm", java.sql.Types.INTEGER);
((com.ibm.db2.jcc.DB2CallableStatement)cstmt).setJccStringAtName("inparm", "4567");
                                  // Assign a value to inparm parameter    3

cstmt.executeUpdate();            // Call the stored procedure             4
int outssid = cstmt.getInt(2);    // Get the output parameter value        5
cstmt.close();
```

IBM Data Server Driver for JDBC and SQLJ のみのメソッドによって拡張クライアント情報をデータ・ソースに提供する

一連の IBM Data Server Driver for JDBC and SQLJ のみのメソッドによって、クライアントに関する追加情報をサーバーに提供できます。その情報は、アカウンティング、ワークロード管理、デバッグに使用できます。

アプリケーションが SQL を実行するなどして、データベース・サーバーにアクセスする操作を実行すると、拡張クライアント情報がデータベース・サーバーに送信されます。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 では、 IBM Data Server Driver for JDBC and SQLJ のみのメソッドは非推奨になっています。代わりに java.sql.Connection.setClientInfo を使用してください。

IBM Data Server Driver for JDBC and SQLJ のみのメソッドを以下の表にまとめます。

表 14. DB2 サーバーにクライアント情報を提供するメソッド

メソッド	提供できる情報
setDB2ClientAccountingInformation	アカウント情報
setDB2ClientApplicationInformation	接続を使用するアプリケーションの名前
setDB2ClientDebugInfo	統合デバッガーの CLIENT DEBUGINFO 接続属性
setDB2ClientProgramId	特定の SQL ステートメントに関連するプログラムを呼び出し元が識別するために役立つ呼び出し元指定ストリング
setDB2ClientUser	接続のためのユーザー名
setDB2ClientWorkstation	接続のためのクライアント・ワークステーション名

拡張クライアント情報を設定するには、以下の手順を実行します。

1. Connection を作成します。

2. java.sql.Connection オブジェクトを com.ibm.db2.jcc.DB2Connection にキャストします。
3. 95 ページの表 14 にまとめられているいずれかのメソッドを呼び出します。
4. SQL ステートメントを実行して、情報を DB2 サーバーに送信します。

これまでに取り上げた手順を実行して、 DB2 サーバーにユーザー名とワークステーション名を渡すコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
public class ClientInfoTest {
  public static void main(String[] args) {
    String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection conn = DriverManager.getConnection(url,        1
        user, password);
      if (conn instanceof DB2Connection) {
        DB2Connection db2conn = (DB2Connection) conn;           2
        db2conn.setDB2ClientUser("Michael L Thompson");         3
        db2conn.setDB2ClientWorkstation("sjwkstn1");
        // Execute SQL to force extended client information to be sent
        // to the server
        conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
          + "WHERE 0 = 1").executeQuery();                      4
      }
    } catch (Throwable e) {
        e.printStackTrace();
      }
  }
}
```

図 18. DB2 サーバーに拡張クライアント情報を渡す例

クライアント情報プロパティーによって拡張クライアント情報をデータ・ソースに提供する

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 では、JDBC 4.0 のクライアント情報プロパティーがサポートされています。それらのプロパティーを使用して、クライアントに関する追加情報をサーバーに提供できます。その情報は、アカウンティング、ワークロード管理、デバッグに使用できます。

アプリケーションが SQL を実行するなどして、データベース・サーバーにアクセスする操作を実行すると、拡張クライアント情報がデータベース・サーバーに送信されます。

アプリケーションでは、Connection.getClientInfo メソッドを使用して、データベース・サーバーからクライアント情報を取得したり、DatabaseMetaData.getClientInfoProperties メソッドを実行して、ドライバーがサポートしているクライアント情報を確認したりすることもできます。

IBM Data Server Driver for JDBC and SQLJ のみのメソッドは非推奨になっているので、代わりに JDBC 4.0 のクライアント情報プロパティーを使用してください。

クライアント情報プロパティーを設定するには、以下の手順を実行します。

1. Connection を作成します。
2. java.sql.setClientInfo メソッドを呼び出して、データベース・サーバーがサポートしているクライアント情報プロパティーを設定します。
3. SQL ステートメントを実行して、情報をデータベース・サーバーに送信します。

これまでに取り上げた手順を実行して、 DB2 サーバーにクライアントのユーザー名とホスト名を渡すコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
public class ClientInfoTest {
  public static void main(String[] args) {
   String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
    try {
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      String user = "db2adm";
      String password = "db2adm";
      Connection conn = DriverManager.getConnection(url,       1
        user, password);
      conn.setClientInfo("ClientUser", "Michael L Thompson"); 2
      conn.setClientInfo("ClientHostname, "sjwkstn1");
      // Execute SQL to force extended client information to be sent
      // to the server
      conn.prepareStatement("SELECT * FROM SYSIBM.SYSDUMMY1"
         + "WHERE 0 = 1").executeQuery();                      3
    } catch (Throwable e) {
      e.printStackTrace();
    }
  }
}
```

図 19. DB2 サーバーに拡張クライアント情報を渡す例

IBM Data Server Driver for JDBC and SQLJ によりサポートされるクライアント情報プロパティー

JDBC 4.0 には、データ・ソースへの接続に関する情報が入っている、クライアント情報プロパティーが組み込まれています。 DatabaseMetaData.getClientInfoProperties メソッドは、 IBM Data Server Driver for JDBC and SQLJ がサポートするクライアント情報プロパティーのリストを戻します。

DatabaseMetaData.getClientInfoProperties を呼び出すと、以下の列を含む結果セットが戻されます。

- NAME
- MAX_LEN
- DEFAULT_VALUE
- DESCRIPTION

以下の表では、 IBM Data Server Driver for JDBC and SQLJ が DB2 Database for Linux, UNIX, and Windowsおよび DB2 for i に対して戻すクライアント情報プロパティー値をリストしています。

表 15. DB2 Database for Linux, UNIX, and Windowsおよび DB2 for i のクライアント情報プロパティー値

NAME	MAX_LEN (バイト)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	255	空ストリング	現在接続を使用しているアプリケーションの名前。この値は DB2 特殊レジスター CURRENT CLIENT_APPLNAME に保管されます。
ClientAccountingInformation	255	空ストリング	接続に指定されているクライアント情報からのアカウンティング・ストリングの値。この値は DB2 特殊レジスター CURRENT CLIENT_ACCTNG に保管されます。
ClientHostname	255	DB2Connection.setDB2ClientWorkstation により設定される値。この値が設定されない場合、デフォルトはローカル・ホストのホスト名になります。	接続を使用しているアプリケーションが実行しているコンピューターのホスト名。この値は DB2 特殊レジスター CURRENT CLIENT_WRKSTNNAME に保管されます。
ClientUser	255	空ストリング	接続を使用しているアプリケーションを実行しているユーザーの名前。この値は DB2 特殊レジスター CURRENT CLIENT_USERID に保管されます。

次の表は、接続でタイプ 4 接続が使用されている場合に IBM Data Server Driver for JDBC and SQLJ が DB2 for z/OS について戻すクライアント情報プロパティー値をリストしています。

表 16. DB2 for z/OS へのタイプ 4 接続の場合のクライアント情報プロパティー値

NAME	MAX_LEN (バイト)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	32	設定されている場合、clientProgramName プロパティー値。そうでない場合、「db2jcc_application」。	現在接続を使用しているアプリケーションの名前。この値は DB2 特殊レジスター CURRENT CLIENT_APPLNAME に保管されます。
ClientAccountingInformation	200	以下の値の連結であるストリング。 • 「JCC*nnnnn*」。ここで *nnnnn* はドライバー・レベルです (例えば 04000 など)。 • DB2Connection.setDB2ClientWorkstation により設定される値。この値が設定されない場合、デフォルトはローカル・ホストのホスト名になります。 • 設定されている場合、applicationName プロパティー値。そうでない場合、20 のブランク。 • 設定されている場合、clientUser プロパティー値。そうでない場合、8 つのブランク。	接続に指定されているクライアント情報からのアカウンティング・ストリングの値。この値は DB2 特殊レジスター CURRENT CLIENT_ACCTNG に保管されます。

表 16. DB2 for z/OS へのタイプ 4 接続の場合のクライアント情報プロパティー値 (続き)

NAME	MAX_LEN (バイト)	DEFAULT_VALUE	DESCRIPTION
ClientHostname	18	DB2Connection.setDB2ClientWorkstation により設定される値。この値が設定されない場合、デフォルトはローカル・ホストのホスト名になります。	接続を使用しているアプリケーションが実行しているコンピューターのホスト名。この値は DB2 特殊レジスター CURRENT CLIENT_WRKSTNNAME に保管されます。
ClientUser	16	DB2Connection.setDB2ClientUser により設定される値。この値が設定されない場合、デフォルトはデータベースの接続に使用される現行のユーザー ID になります。	接続を使用しているアプリケーションを実行しているユーザーの名前。この値は DB2 特殊レジスター CURRENT CLIENT_USERID に保管されます。

次の表は、接続でタイプ 2 接続が使用されている場合に IBM Data Server Driver for JDBC and SQLJ が DB2 for z/OS について戻すクライアント情報プロパティー値をリストしています。

表 17. DB2 for z/OS へのタイプ 2 接続の場合のクライアント情報プロパティー値

NAME	MAX_LEN (バイト)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	32	空ストリング	現在接続を使用しているアプリケーションの名前。この値は DB2 特殊レジスター CURRENT CLIENT_APPLNAME に保管されます。
ClientAccountingInformation	200	空ストリング	接続に指定されているクライアント情報からのアカウンティング・ストリングの値。この値は DB2 特殊レジスター CURRENT CLIENT_ACCTNG に保管されます。
ClientHostname	18	空ストリング	接続を使用しているアプリケーションが実行しているコンピューターのホスト名。この値は DB2 特殊レジスター CURRENT CLIENT_WRKSTNNAME に保管されます。
ClientUser	16	空ストリング	接続を使用しているアプリケーションを実行しているユーザーの名前。この値は DB2 特殊レジスター CURRENT CLIENT_USERID に保管されます。

以下の表では、 IBM Data Server Driver for JDBC and SQLJ が IBM Informix Dynamic Server に対して戻すクライアント情報プロパティー値をリストしています。

表 18. IBM Informix Dynamic Server のクライアント情報プロパティー値

NAME	MAX_LEN (バイト)	DEFAULT_VALUE	DESCRIPTION
ApplicationName	20	空ストリング	現在接続を使用しているアプリケーションの名前。
ClientAccountingInformation	199	空ストリング	接続に指定されているクライアント情報からのアカウンティング・ストリングの値。

表 18. IBM Informix Dynamic Server のクライアント情報プロパティー値 (続き)

NAME	MAX_LEN (バイト)	DEFAULT_VALUE	DESCRIPTION
ClientHostname	20	DB2Connection.setDB2ClientWorkstation により設定される値。この値が設定されない場合、デフォルトはローカル・ホストのホスト名になります。	接続を使用しているアプリケーションが実行しているコンピューターのホスト名。
ClientUser	1024	空ストリング	接続を使用しているアプリケーションを実行しているユーザーの名前。

JDBC アプリケーションでのオプティミスティック・ロック

データ・ソースのオプティミスティック・ロックを利用する JDBC アプリケーションを作成することができます。

オプティミスティック・ロック とは、アプリケーションで SELECT 操作と UPDATE または DELETE 操作との間のロック解除に使用できる手法のことです。選択した行がそのアプリケーションによって更新または削除される前に変更された場合は、UPDATE または DELETE 操作は失敗します。オプティミスティック・ロックによって、指定されたリソースが他のトランザクションによって使用できない時間が最小限に抑えられます。

DB2 for i データ・ソースへの接続でオプティミスティック・ロックを使用するには、DB2 for i V6R1 以降が必要です。

通常、アプリケーションでは以下のステップが実行されて、オプティミスティック・ロックが使用されます。

1. 行を表から選択する。
2. 表に対してロック解除する。
3. 選択した行が変更されていない場合は、更新する。

行が変更されたかどうかを検査するために、アプリケーションにより行の変更トークンが照会されます。行の変更トークンは、必ずしも行が変更されたかどうかの完全に正確な指標であるわけではありません。行の変更トークンは、行の変更タイム・スタンプ列を持つ表を作成した場合は完全に正確です。行の変更タイム・スタンプ列なしで表を作成するか、または表を変更して行の変更タイム・スタンプ列を追加した場合は、行の変更トークンに、行に対する更新が正確に反映されていないおそれがあります。つまり、行の変更トークンにより、行が変更されていないにもかかわらず行が変更されたと示されることがあります。この状態を *false negative* 状態と呼びます。

オプティミスティック・ロックを実行するための JDBC アプリケーションを作成する場合は、以下の同様の手順に従います。

1. 照会を準備して実行する。

   オプティミスティック・ロック情報が必要かどうか、およびその情報に false negative を含めることができるかどうかを指示します。

2. ResultSet にオプティミスティック・ロック情報が含まれるかどうか、およびその情報で false negative の生成が可能かどうかを判別する。

   オプティミスティック・ロック情報のタイプに基づいて、オプティミスティック・ロックを続行するかどうかを判定することができます。
3. 表に対してロック解除する。
4. 選択した行が行の変更トークンによって変更されていないと示される場合は、更新する。

以下のコードでは、JDBC アプリケーションでのオプティミスティック・ロックの実行方法を示しています。例内の数字は、上記の各手順に対応しています。

```
com.ibm.db2.jcc.DB2Statement s1 =
  (com.ibm.db2.jcc.DB2Statement)conn.createStatement();
ResultSet rs =
 ((com.ibm.db2.jcc.DB2Statement)s1).executeDB2OptimisticLockingQuery
 ("SELECT EMPNO, SALARY FROM EMP WHERE EMP.LASTNAME = 'HAAS'",
  com.ibm.db2.jcc.DB2Statement.RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES); 1
                                  // Indicate that you plan to do
                                  // optimistic locking, and that you
                                  // want optimistic locking information
                                  // that does not generate
                                  // false negatives
ResultSetMetaData rsmd = rs.getMetaData();
int optColumns =                                                          2
  ((com.ibm.db2.jcc.DB2ResultSetMetaData)rsmd).getDB2OptimisticLockingColumns();
                                  // Retrieve the optimistic locking
                                  // information.
boolean optColumnsReturned = false;

if (optColumns == 0);             // If optimistic locking information is not
                                  // returned, do not attempt to do
                                  // optimistic locking.
else if (optColumns == 1);        // A value of 1 is never returned if
                                  // RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES
                                  // is specified, because 1 indicates
                                  // that there could be false negatives.
else if (optColumns == 2)         // If optimistic locking information is
   optColumnsReturned = true;     // returned, and false negatives will not
                                  // occur, try optimistic locking.

rs.next();                        // Retrieve the contents of the ResultSet
int emp_id = rs.getInt(1);
double salary = rs.getDouble(2);

long rowChangeToken = 0;
Object rid = null;
int type = -1;

if (optColumnsReturned) {
  rowChangeToken =                // Get the row change token.
    ((com.ibm.db2.jcc.DB2ResultSet)rs).getDB2RowChangeToken();
  rid = ((com.ibm.db2.jcc.DB2ResultSet)rs).getDB2RID();
                                  // Get the RID, which uniquely identifies
                                  // the row.
  int type = ((com.ibm.db2.jcc.DB2ResultSet)rs).getDB2RIDType ();
                                  // Get the data type of the RID.
}
// ************************************************
// Release the locks or disconnect from the database.
// Perform some work on the retrieved data.
// Reconnect to the data source.
// ************************************************
...
PreparedStatement s2 =
  conn.prepareStatement ("UPDATE EMP SET SALARY = ? " +
  "WHERE EMPNO = ? AND ROW CHANGE TOKEN FOR EMP = ? and " +
  "RID_BIT(EMP) = ?");
                                  // Statement for updating the
                                  // previously selected rows that
                                  // have not changed.
s2.setDouble(1, salary+10000);
s2.setInt(2, emp_id);
```

```
                                    // Set the new row values.
s2.setLong(3, rowChangeToken);
                                    // Set the row change token of the
                                    // previously retrieved row.
if (type == java.sql.Types.BIGINT)
  s2.setLong (4, ((Long)rid).longValue());
else if (type == java.sql.Types.VARBINARY)
  s2.setBytes (4, (byte[])rid);
                                    // Set the RID of the previously
                                    // retrieved row.
                                    // Use the correct setXXX method
                                    // for the data type of the RID.
int updateCount = s2.executeUpdate();                                   3
                                    // Perform the update.
if (updateCount == 1);              // Update is successful.
else                                // Update failed.
...
```

JDBC アプリケーションでの XML データ

JDBC アプリケーションでは、XML 列へのデータの保管、および XML 列からのデータを取り出しが可能です。

データベース表では、XML 組み込みデータ・タイプを使用して XML データをノードの構造化セットとしてツリー形式で列に保管します。

アプリケーションでは、XML データは直列化されたストリング形式になります。

DB2 Database for Linux, UNIX, and Windows に接続する JDBC アプリケーションでは、XML データはテキスト XML 形式になります。 DB2 for z/OS に接続する JDBC アプリケーションでは、XML データはテキスト XML 形式またはバイナリー XML 形式が可能です。

JDBC アプリケーションでは以下を行うことができます。

- setXXX メソッドを使用して XML 文書全体を XML 列に保管します。
- getXXX メソッドを使用して XML 文書全体を XML 列から取り出します。
- SQL XMLQUERY 関数を使用してシーケンスをデータベース内のシリアライズされたシーケンスに取り出してから、getXXX メソッドを使用してデータをアプリケーション変数に取り出すことによって、XML 列の文書からシーケンスを取り出します。
- ストリング 'XQUERY' が前に付加されている XQuery 式を使用して、シーケンスのエレメントをデータベース内の結果表に取り出すことによって、XML 列の文書からシーケンスを取り出します。結果表の各行はシーケンス内のアイテムを表します。続いて、getXXX メソッドを使用してデータをアプリケーション変数に取り出します。
- SQL XMLTABLE 関数を使用して結果表を定義し、その表を取り出すことによって、XML 列の文書からユーザー定義表としてシーケンスを取り出します。続いて、getXXX メソッドを使用して、結果表からのデータをアプリケーション変数に取り出します。

JDBC 4.0 の `java.sql.SQLXML` オブジェクトを使用して、XML 列内のデータの取り出しおよび更新を行うことができます。 ResultSetMetaData.getColumnTypeName などのメタデータ・メソッドを呼び出すと、XML 列タイプの整数値 `java.sql.Types.SQLXML` が戻されます。

JDBC アプリケーションでの XML 列の更新

データベース表の XML 列についてデータを更新または挿入する場合は、JDBC アプリケーション内の入力データは直列化されたストリング形式である必要があります。

次の表には、データの XML 列への書き込みに使用できるメソッドと対応する入力データ・タイプがリストされています。

表 19. XML 列の更新用のメソッドとデータ・タイプ

メソッド	入力データ・タイプ
PreparedStatement.setAsciiStream	InputStream
PreparedStatement.setBinaryStream	InputStream
PreparedStatement.setBlob	Blob
PreparedStatement.setBytes	byte[]
PreparedStatement.setCharacterStream	Reader
PreparedStatement.setClob	Clob
PreparedStatement.setObject	byte[]、Blob、Clob、SQLXML、DB2Xml (非推奨)、InputStream、Reader、String
PreparedStatement.setSQLXML	SQLXML
PreparedStatement.setString	String

XML データのエンコード方式は、データ自体から導出する (*内部的にエンコードされた データという*) か、または外部ソースから導き出される (*外部的にエンコードされた データという*) ことができます。データベース・サーバーにバイナリー・データとして送信される XML データは、内部的にエンコードされたデータとして処理されます。データ・ソースに文字データとして送信される XML データは、外部的にエンコードされたデータとして処理されます。

Java アプリケーションの外部エンコード方式は、常に Unicode エンコード方式です。

外部的にエンコードされたデータに、内部エンコード方式を含めることができます。つまり、このデータはデータ・ソースに文字データとして送信されますが、このデータにエンコード方式の情報を含めることができます。データ・ソースにより、以下のように内部エンコード方式と外部エンコード方式の間の非互換性が処理されます。

- データ・ソースが DB2 Database for Linux, UNIX, and Windows の場合、外部エンコード方式と内部エンコード方式の間に互換性がなく、外部エンコード方式と内部エンコード方式が Unicode でない場合は、データベース・ソースによりエラーが生成されます。外部エンコード方式と内部エンコード方式が Unicode の場合は、データベース・ソースでは内部エンコード方式が無視されます。
- データベース・ソースが DB2 for z/OS の場合、データベース・ソースでは内部エンコード方式が無視されます。

XML 列のデータは UTF-8 エンコード方式で保管されます。データベース・ソースにより、内部エンコード方式または外部エンコード方式から UTF-8 へのデータの変換が処理されます。

例: 以下の例は、データを SQLXML オブジェクトから XML 列へ挿入する方法を示しています。データがストリング・データであるため、データベース・ソースにより外部的にエンコードされるデータとして処理されます。

```
public void insertSQLXML()
{
 Connection con = DriverManager.getConnection(url);
  SQLXML info = con.createSQLXML();
                            // Create an SQLXML object
  PreparedStatement insertStmt = null;
  String infoData =
    "<customerinfo xmlns=""http://posample.org"" " +
    "Cid=""1000"">...</customerinfo>";
  info.setString(infoData);
                            // Populate the SQLXML object
  int cid = 1000;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = con.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    insertStmt.setSQLXML(2, info);
                            // Assign the SQLXML object value
               // to an input parameter
    if (insertStmt.executeUpdate() != 1) {
      System.out.println("insertSQLXML: No record inserted.");
    }
  }
  catch (IOException ioe) {
   ioe.printStackTrace();
  }
  catch (SQLException sqle) {
    System.out.println("insertSQLXML: SQL Exception: " +
      sqle.getMessage());
    System.out.println("insertSQLXML: SQL State: " +
      sqle.getSQLState());
    System.out.println("insertSQLXML: SQL Error Code: " +
      sqle.getErrorCode());

  }
}
```

例: 以下の例は、データをファイルから XML 列へ挿入する方法を示しています。データはバイナリー・データとして挿入されるため、データベース・サーバーにより内部エンコード方式が使用されます。

```
public void insertBinStream(Connection conn)
{
  PreparedStatement insertStmt = null;
  String sqls = null;
  int cid = 0;
  Statement stmt=null;
  try {
    sqls = "INSERT INTO CUSTOMER (CID, INFO) VALUES (?, ?)";
    insertStmt = conn.prepareStatement(sqls);
    insertStmt.setInt(1, cid);
    File file = new File(fn);
    insertStmt.setBinaryStream(2,
      new FileInputStream(file), (int)file.length());
    if (insertStmt.executeUpdate() != 1) {
      System.out.println("insertBinStream: No record inserted.");
    }
  }
  catch (IOException ioe) {
   ioe.printStackTrace();
  }
  catch (SQLException sqle) {
```

```
      System.out.println("insertBinStream: SQL Exception: " +
        sqle.getMessage());
      System.out.println("insertBinStream: SQL State: " +
        sqle.getSQLState());
      System.out.println("insertBinStream: SQL Error Code: " +
        sqle.getErrorCode());

    }
  }
```

JDBC アプリケーションにおける XML データの取り出し

JDBC アプリケーションでは、ResultSet.getXXX または ResultSet.getObject メソッドを使用して XML 列からデータを取り出すことができます。

DB2 表の XML 列からデータを取り出す場合、出力データは直列化されたストリング・フォーマットになります。これは、XML 列の内容全体を取り出す場合、または列からシーケンスを取り出す場合に当てはまります。

XML データを取り出すには、以下の手法のいずれかを使用することができます。

- ResultSet.getSQLXML メソッドを使用してデータを取り出します。次に、SQLXML.getXXX メソッドを使用して、データを互換性のある出力データ・タイプに取り出します。この手法には、JDBC 4.0 以降が必要です。
- ResultSet.getObject 以外の ResultSet.getXXX メソッドを使用して、互換性のあるデータ・タイプにデータを取り出します。
- ResultSet.getObject メソッドを使用してデータを取り出してから、それを DB2Xml タイプにキャストし、DB2Xml オブジェクトに割り当てます。次に、DB2Xml.getDB2XXX または DB2Xml.getDB2XmlXXX メソッドを使用して、互換性のある出力データ・タイプにデータを取り出します。

  この手法では、推奨されない DB2Xml オブジェクトが使用されます。前で説明した手法の使用をお勧めします。

次の表には、XML データを取り出すための ResultSet メソッドおよび対応する出力データ・タイプがリストされています。

表 20. XML データを取り出すための ResultSet メソッドおよびデータ・タイプ

メソッド	出力データ・タイプ
ResultSet.getAsciiStream	InputStream
ResultSet.getBinaryStream	InputStream
ResultSet.getBytes	byte[]
ResultSet.getCharacterStream	Reader
ResultSet.getObject	Object
ResultSet.getSQLXML	SQLXML
ResultSet.getString	String

次の表には、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml オブジェクトからのデータの取り出しに呼び出すことができるメソッド、および対応する出力データ・タイプと XML 宣言でのエンコード方式のタイプがリストされています。

表 21. SQLXML および DB2Xml メソッド、データ・タイプ、および追加されるエンコード仕様

メソッド	出力データ・タイプ	追加される XML 内部エンコード宣言のタイプ
SQLXML.getBinaryStream	InputStream	なし
SQLXML.getCharacterStream	Reader	なし
SQLXML.getSource	Source[1]	なし
SQLXML.getString	String	なし
DB2Xml.getDB2AsciiStream	InputStream	なし
DB2Xml.getDB2BinaryStream	InputStream	なし
DB2Xml.getDB2Bytes	byte[]	なし
DB2Xml.getDB2CharacterStream	Reader	なし
DB2Xml.getDB2String	String	なし
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	getDB2XmlBinaryStream *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlBytes	byte[]	DB2Xml.getDB2XmlBytes *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

注:

1. 戻されるクラスは getSource の呼び出し側によって指定されますが、このクラスは javax.xml.transform.Source を拡張するものです。

戻されるデータに対してアプリケーションが XMLSERIALIZE 関数を実行すると、関数の実行後に、そのデータには XML データ・タイプではなく XMLSERIALIZE 関数で指定されたデータ・タイプが含まれます。そのため、ドライバーは指定されたタイプとしてそのデータを処理し、内部エンコード宣言をすべて無視します。

例: 以下の例は、データを XML 列から SQLXML オブジェクトに取り出してから、SQLXML.getString メソッドを使用して、ストリングにデータを取り出す方法を示しています。

```
public void fetchToSQLXML(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToSQLXML: Get XML data as an SQLXML object " +
    "using getSQLXML");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToSQLXML: Column type = " + colType);
      while (rs.next()) {
        // Retrieve the XML data with getSQLXML.
        // Then write it to a string with
```

```
        // explicit internal ISO-10646-UCS-2 encoding.
        java.sql.SQLXML xml = rs.getSQLXML(1);
        System.out.println (xml.getString());
      }
      rs.close();
    }
    catch (SQLException sqle) {
      System.out.println("fetchToSQLXML: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToSQLXML: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToSQLXML: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

例: 以下の例は、データを XML 列からストリング変数に取り出す方法を示しています。

```
public void fetchToString(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToString: Get XML data " +
    "using getString");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;

    try{
      sqls = "SELECT info FROM customer WHERE cid = " + cid;
      selectStmt = conn.prepareStatement(sqls);
      rs = selectStmt.executeQuery();

      // Get metadata
      // Column type for XML column is the integer java.sql.Types.OTHER
      ResultSetMetaData meta = rs.getMetaData();
      int colType = meta.getColumnType(1);
      System.out.println("fetchToString: Column type = " + colType);

      while (rs.next()) {
        stringDoc = rs.getString(1);
        System.out.println("Document contents:");
        System.out.println(stringDoc);
 }
    catch (SQLException sqle) {
      System.out.println("fetchToString: SQL Exception: " +
        sqle.getMessage());
      System.out.println("fetchToString: SQL State: " +
        sqle.getSQLState());
      System.out.println("fetchToString: SQL Error Code: " +
        sqle.getErrorCode());
    }
  }
```

例: 以下の例は、データを XML 列から DB2Xml オブジェクトに取り出してから、DB2Xml.getDB2XmlString メソッドを使用して、ISO-10646-UCS-2 エンコード仕様で追加された XML 宣言を含むストリングにデータを取り出す方法を示しています。

```
public void fetchToDB2Xml(long cid, java.sql.Connection conn)
  {
    System.out.println(">> fetchToDB2Xml: Get XML data as a DB2XML object " +
    "using getObject");
    PreparedStatement selectStmt = null;
    String sqls = null, stringDoc = null;
    ResultSet rs = null;
```

```
  try{
    sqls = "SELECT info FROM customer WHERE cid = " + cid;
    selectStmt = conn.prepareStatement(sqls);
    rs = selectStmt.executeQuery();

    // Get metadata
    // Column type for XML column is the integer java.sql.Types.OTHER
    ResultSetMetaData meta = rs.getMetaData();
    int colType = meta.getColumnType(1);
    System.out.println("fetchToDB2Xml: Column type = " + colType);
    while (rs.next()) {
      // Retrieve the XML data with getObject, and cast the object
      // as a DB2Xml object. Then write it to a string with
      // explicit internal ISO-10646-UCS-2 encoding.
      com.ibm.db2.jcc.DB2Xml xml =
        (com.ibm.db2.jcc.DB2Xml) rs.getObject(1);
      System.out.println (xml.getDB2XmlString());
    }
    rs.close();
  }
  catch (SQLException sqle) {
    System.out.println("fetchToDB2Xml: SQL Exception: " +
      sqle.getMessage());
    System.out.println("fetchToDB2Xml: SQL State: " +
      sqle.getSQLState());
    System.out.println("fetchToDB2Xml: SQL Error Code: " +
      sqle.getErrorCode());
  }
}
```

Java アプリケーションでの XML パラメーターを使用したルーチンの呼び出し

Java アプリケーションは、 DB2 Database for Linux, UNIX, and Windows データ・ソースにある、XML パラメーターを持つストアード・プロシージャーを呼び出すことができます。

ネイティブ SQL プロシージャーの場合、ストアード・プロシージャー定義内の XML パラメーターは XML タイプです。 DB2 Database for Linux, UNIX, and Windows データ・ソース上の外部ストアード・プロシージャーおよびユーザー定義関数の場合は、ルーチン定義内の XML パラメーターは XML AS CLOB タイプです。 XML パラメーターのあるストアード・プロシージャーまたはユーザー定義関数を呼び出す場合は、呼び出しステートメントで互換データ・タイプを使用する必要があります。

JDBC プログラムから XML 入力パラメーターのあるルーチンを呼び出すには、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml タイプのパラメーターを使用します。 XML 出力パラメーターを登録するには、パラメーターを java.sql.Types.SQLXML または com.ibm.db2.jcc.DB2Types.XML タイプで登録します。 (com.ibm.db2.jcc.DB2Xml および com.ibm.db2.jcc.DB2Types.XML タイプは推奨されません。)

例: 3 つの XML パラメーター (IN パラメーター、OUT パラメーター、および INOUT パラメーター) を使用するストアード・プロシージャーを呼び出す JDBC プログラム。この例では JDBC 4.0 が必要です。

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
```

```
                                    // Declare an input, output, and
                                    // INOUT XML parameter
Connection con;
CallableStatement cstmt;
ResultSet rs;
...
cstmt = con.prepareCall("CALL SP_xml(?,?,?)");
                                    // Create a CallableStatement object
cstmt.setObject (1, in_xml);        // Set input parameter
cstmt.setObject (3, inout_xml);     // Set inout parameter
cstmt.registerOutParameter (2, java.sql.Types.SQLXML);
                                    // Register out and input parameters
cstmt.registerOutParameter (3, java.sql.Types.SQLXML);
cstmt.executeUpdate();              // Call the stored procedure
out_xml = cstmt.getSQLXML(2);       // Get the OUT parameter value
inout_xml = cstmt.getSQLXML(3);     // Get the INOUT parameter value
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                           // Use the SQLXML.getString
                           // method to convert the out_xml
                           // value to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                           // Use the SQLXML.getString
                           // method to convert the inout_xml
                           // value to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
```

SQLJ プログラムから XML パラメーターのあるルーチンを呼び出すには、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml タイプのパラメーターを使用します。

例: 3 つの XML パラメーター (IN パラメーター、OUT パラメーター、および INOUT パラメーター) を使用するストアード・プロシージャーを呼び出す SQLJ プログラム。この例では JDBC 4.0 が必要です。

```
java.sql.SQLXML in_xml = xmlvar;
java.sql.SQLXML out_xml = null;
java.sql.SQLXML inout_xml = xmlvar;
                                    // Declare an input, output, and
                                    // INOUT XML parameter
...
#sql [myConnCtx] {CALL SP_xml(:IN in_xml,
                              :OUT out_xml,
                              :INOUT inout_xml)};
                                    // Call the stored procedure
System.out.println("Parameter values from SP_xml call: ");
System.out.println("Output parameter value ");
MyUtilities.printString(out_xml.getString());
                           // Use the SQLXML.getString
                           // method toconvert the out_xml value
                           // to a string for printing.
                           // Call a user-defined method called
                           // printString (not shown) to print
                           // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printString(inout_xml.getString());
                           // Use the SQLXML.getString
                           // method to convert the inout_xml
```

```
// value to a string for printing.
// Call a user-defined method called
// printString (not shown) to print
// the value.
```

XML スキーマの登録および除去の Java サポート

IBM Data Server Driver for JDBC and SQLJ により、XML スキーマとそのコンポーネントを登録および除去するための Java アプリケーション・プログラムを作成できるメソッドが提供されます。

メソッドは、以下のとおりです。

DB2Connection.registerDB2XMLSchema
XML スキーマを 1 つ以上の XML スキーマ文書を使用して DB2 に登録します。このメソッドには 2 つの形式があります。1 つは InputStream オブジェクトから入力される XML スキーマ文書用の形式で、もう 1 つは String 内の XML スキーマ文書用の形式です。

DB2Connection.deregisterDB2XMLObject
XML スキーマ定義を DB2 から除去します。

DB2Connection.updateDB2XmlSchema
登録済みの XML スキーマ内の XML スキーマ文書を登録済みの別の XML スキーマからの XML スキーマ文書で置き換えます。オプションで、内容がコピーされた XML スキーマをドロップします。このメソッドは、 DB2 Database for Linux, UNIX, and Windows への接続の場合のみ使用できます。

これらのメソッドを呼び出す前に、これらのメソッドをサポートしているストアード・プロシージャーを DB2 データベース・サーバーにインストールする必要があります。

例: XML スキーマの登録: 以下の例では、入力ストリームから読み取られた単一の XML スキーマ文書 (customer.xsd) を使用して、XML スキーマを DB2 に登録するために registerDB2XmlSchema を使用する方法が示されています。登録済みのスキーマの SQL スキーマ名は SYSXSR です。 *xmlSchemaLocations* 値が NULL のため、 DB2 では、この XML スキーマは NULL 以外の XML スキーマ・ロケーション値を提供する DSN_XMLVALIDATE の呼び出し時には検出されません。追加のプロパティーは登録されません。

図 20. 入力ストリームからの XML 文書を使用した XML スキーマの DB2 への登録の例

```
public static void registerSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Define the registerDB2XmlSchema parameters
  String[] xmlSchemaNameQualifiers = new String[1];
  String[] xmlSchemaNames = new String[1];
  String[] xmlSchemaLocations = new String[1];
  InputStream[] xmlSchemaDocuments = new InputStream[1];
  int[] xmlSchemaDocumentsLengths = new int[1];
  java.io.InputStream[] xmlSchemaDocumentsProperties = new InputStream[1];
  int[] xmlSchemaDocumentsPropertiesLengths = new int[1];
  InputStream xmlSchemaProperties;
  int xmlSchemaPropertiesLength;
```

```
  //Set the parameter values
  xmlSchemaLocations[0] = "";
  FileInputStream fi = null;
  xmlSchemaNameQualifiers[0] = "SYSXSR";
  xmlSchemaNames[0] = schemaName;
  try {
    fi = new FileInputStream("customer.xsd");
    xmlSchemaDocuments[0] = new BufferedInputStream(fi);
  } catch (FileNotFoundException e) {
         e.printStackTrace();
  }
  try {
     xmlSchemaDocumentsLengths[0] = (int) fi.getChannel().size();
     System.out.println(xmlSchemaDocumentsLengths[0]);
  } catch (IOException e1) {
         e1.printStackTrace();
  }
  xmlSchemaDocumentsProperties[0] = null;
  xmlSchemaDocumentsPropertiesLengths[0] = 0;
  xmlSchemaProperties = null;
  xmlSchemaPropertiesLength = 0;
  DB2Connection ds = (DB2Connection) con;
  // Invoke registerDB2XmlSchema
  ds.registerDB2XmlSchema(
    xmlSchemaNameQualifiers,
    xmlSchemaNames,
    xmlSchemaLocations,
    xmlSchemaDocuments,
    xmlSchemaDocumentsLengths,
    xmlSchemaDocumentsProperties,
    xmlSchemaDocumentsPropertiesLengths,
    xmlSchemaProperties,
    xmlSchemaPropertiesLength,
    false);
}
```

例: XML スキーマの除去: 以下の例では、XML スキーマを DB2 から除去するために deregisterDB2XmlObject を使用する方法が示されています。登録済みのスキーマの SQL スキーマ名は SYSXSR です。

図 21. XML スキーマの DB2 からの除去の例

```
public static void deregisterSchema(
  Connection con,
  String schemaName)
  throws SQLException {
  // Define and assign values to the deregisterDB2XmlObject parameters
  String xmlSchemaNameQualifier = "SYSXSR";
  String xmlSchemaName = schemaName;
  DB2Connection ds = (DB2Connection) con;
  // Invoke deregisterDB2XmlObject
  ds.deregisterDB2XmlObject(
    xmlSchemaNameQualifier,
    xmlSchemaName);
}
```

例: XML スキーマの更新: 以下の例は、 DB2 Database for Linux, UNIX, and Windows への接続のみに適用されます。 XML スキーマの内容を別の XML スキーマの内容で更新するために updateDB2XmlSchema を使用する方法が示されています。コピーされたスキーマは、リポジトリー内に維持されます。登録済みの両方のスキーマの SQL スキーマ名は SYSXSR です。

図 22. XML スキーマの更新の例

```
public static void updateSchema(
  Connection con,
  String schemaNameTarget,
  String schemaNameSource)
  throws SQLException {
  // Define and assign values to the updateDB2XmlSchema parameters
  String xmlSchemaNameQualifierTarget = "SYSXSR";
  String xmlSchemaNameQualifierSource = "SYSXSR";
  String xmlSchemaNameTarget = schemaNameTarget;
  String xmlSchemaNameSource = schemaNameSource;
  boolean dropSourceSchema = false;
  DB2Connection ds = (DB2Connection) con;
  // Invoke updateDB2XmlSchema
  ds.updateDB2XmlSchema(
    xmlSchemaNameQualifierTarget,
    xmlSchemaNameTarget,
  xmlSchemaNameQualifierSource,
  xmlSchemaNameSource,
    dropSourceSchema);
}
```

JDBC アプリケーションにおけるトランザクション制御

JDBC アプリケーションでは、他のタイプの SQL アプリケーションと同様に、トランザクション制御にトランザクションの明示的または暗黙的なコミットおよびロールバック、およびトランザクションの分離レベルの設定が関係します。

IBM Data Server Driver for JDBC and SQLJ の分離レベル

IBM Data Server Driver for JDBC and SQLJ は多数の分離レベルをサポートしています。これらは、データベース・サーバーの分離レベルに対応しています。

JDBC 分離レベルは、Connection.setTransactionIsolation メソッドを使用して、JDBC プログラム内の作業単位に対して設定できます。デフォルトの分離レベルは defaultIsolationLevel プロパティーで設定できます。

次の表は、Connection.setTransactionIsolation メソッドで指定できる *level* の値と、DB2 データベース・サーバーでそれに相当するレベルを示しています。

表 22. 同等の JDBC および DB2 分離レベル

JDBC 値	DB2 独立性レベル
java.sql.Connection.TRANSACTION_SERIALIZABLE	反復可能読み取り
java.sql.Connection.TRANSACTION_REPEATABLE_READ	読み取り固定
java.sql.Connection.TRANSACTION_READ_COMMITTED	カーソル固定
java.sql.Connection.TRANSACTION_READ_UNCOMMITTED	非コミット読み取り

次の表は、Connection.setTransactionIsolation メソッドで指定できる *level* の値と、IBM Informix Dynamic Server (IDS) でそれに相当するレベルを示しています。

表 23. 同等の JDBC および IDS 分離レベル

JDBC 値	IDS 分離レベル
java.sql.Connection.TRANSACTION_SERIALIZABLE	反復可能読み取り

表 23. 同等の JDBC および IDS 分離レベル (続き)

JDBC 値	IDS 分離レベル
java.sql.Connection.TRANSACTION_REPEATABLE_READ	反復可能読み取り
java.sql.Connection.TRANSACTION_READ_COMMITTED	コミット読み取り
java.sql.Connection.TRANSACTION_READ_UNCOMMITTED	ダーティー読み取り
com.ibm.db2.jcc.DB2Connection.TRANSACTION_IDS_CURSOR_STABILITY	IDS カーソル固定
com.ibm.db2.jcc.DB2Connection.TRANSACTION_IDS_LAST_COMMITTED	コミット読み取り、最終コミット

JDBC トランザクションのコミットまたはロールバック

JDBC では、トランザクションの明示的なコミットまたはロールバックのために、commit メソッドまたは rollback メソッドを使用します。

以下に例を示します。

```
 Connection con;
 ...
con.commit();
```

自動コミット・モードがオンになっていると、データベース・マネージャーは、各 SQL ステートメントの完了後にコミット操作を実行します。自動コミット・モードをオンに設定するには、Connection.setAutoCommit(true) メソッドを呼び出します。自動コミット・モードをオフに設定するには、Connection.setAutoCommit(false) メソッドを呼び出します。自動コミット・モードがオンになっているかどうかを確認するには、Connection.getAutoCommit メソッドを呼び出します。

分散トランザクションに加わっている接続では、setAutoCommit(true) メソッドを呼び出せません。

自動コミットの状態を変更すると、データベース・マネージャーは、アプリケーションがトランザクションの境界に達していなくてもコミット操作を実行します。

接続が分散トランザクションに加わっていると、関連するアプリケーションは、commit メソッドまたは rollback メソッドを呼び出せません。

デフォルトの JDBC 自動コミット・モード

デフォルトの自動コミット・モードは、JDBC アプリケーションが接続するデータ・ソースによって決まります。

DB2 データ・ソースの自動コミットのデフォルト

DB2 データ・ソースへの接続の場合、デフォルトの自動コミット・モードは `true` です。

IDS データ・ソースの自動コミットのデフォルト

IDS データ・ソースに対する接続の場合、デフォルトの自動コミット・モードはデータ・ソースのタイプによって決まります。次の表はデフォルトを示しています。

表 24. IDS データ・ソースのデフォルトの自動コミット・モード

データ・ソースのタイプ	デフォルトの自動コミット・モード (ローカル・トランザクション)	デフォルトの自動コミット・モード (グローバル・トランザクション)
ANSI 準拠データベース	true	false
ロギングを行わない非 ANSI 準拠データベース	false	適用外
ロギングを行う非 ANSI 準拠データベース	true	false

IBM Data Server Driver for JDBC and SQLJ での例外と警告

JDBC アプリケーションでは、SQL エラーによって例外がスローされ、ユーザーは try/catch ブロックを使用してこの例外を処理します。SQL 警告では例外はスローされないので、 SQL ステートメントの実行後に警告が発生したかどうかを確認するためのメソッドを呼び出す必要があります。

IBM Data Server Driver for JDBC and SQLJ では次のようなクラスおよびインターフェースが提供され、これによってエラーや警告の情報が得られます。

SQLException

エラー処理のための SQLException クラス。すべての JDBC メソッドは、実行中にエラーが発生すると SQLException のインスタンスをスローします。JDBC 仕様に従って、 SQLException オブジェクトには以下の情報が含まれます。

- エラー・コードを含む int 値。SQLException.getErrorCode はこの値を取得します。
- SQLSTATE を含む String オブジェクト、または NULL。SQLException.getSQLState はこの値を取得します。
- エラーについての記述を含む String オブジェクト、または NULL。SQLException.getMessage はこの値を取得します。
- 次の SQLException へのポインター、または NULL。SQLException.getNextException はこの値を取得します。

JDBC メソッドがスローする SQLException が 1 つだけの場合、その SQLException は、 IBM Data Server Driver for JDBC and SQLJ がメソッドを処理したときに発生した基本の Java 例外によって引き起こされた可能性があります。この場合は SQLException がこの基礎となる例外をラップするので、SQLException.getCause メソッドによってエラーに関する情報を取り出すことができます。

DB2Diagnosable

IBM Data Server Driver for JDBC and SQLJ 専用のインターフェース com.ibm.db2.jcc.DB2Diagnosable は、SQLException クラスを拡張します。DB2Diagnosable インターフェースにより、データ・ソースへのアクセス時に発生するエラーについてより多くの情報が得られます。JDBC ドライバーがエラーを検出した場合は、DB2Diagnosable によって得られる情報は、標準の SQLException クラ

スによる情報と同じです。しかしデータベース・サーバーがエラーを検出したときには、DB2Diagnosable は以下のメソッドを追加して、エラーに関する追加情報を提供します。

getSqlca
以下の情報を含む DB2Sqlca オブジェクトを返します。
- SQL エラー・コード
- SQLERRMC 値
- SQLERRP 値
- SQLERRD 値
- SQLWARN 値
- SQLSTATE

getThrowable
SQLException の原因となった java.lang.Throwable オブジェクトを返します。そういうオブジェクトが存在しない場合は NULL を返します。

printTrace
診断情報を印刷します。

SQLException サブクラス

JDBC 4.0 以降では、以下の例外クラスをキャッチすることで、SQLException よりさらに詳しい情報を取得できます。

- SQLNonTransientException

  SQLNonTransientException は、以前に失敗した SQL 操作を再試行しても正常に実行できない場合に、何らかの修正処置をとらない限りスローされます。
  SQLNonTransientException クラスには以下のサブクラスがあります。
  - SQLFeatureNotSupportedException
  - SQLNonTransientConnectionException
  - SQLDataException
  - SQLIntegrityConstraintViolationException
  - SQLInvalidAuthorizationSpecException
  - SQLSyntaxException
- SQLTransientException

  SQLTransientException は、以前に失敗した SQL 操作を再試行すれば成功する可能性がある場合に、アプリケーションからの介入を受けることなくスローされます。SQLTransientException がスローされたあとでも接続は引き続き有効です。
  SQLTransientException クラスには以下のサブクラスがあります。
  - SQLTransientConnectionException
  - SQLTransientRollbackException
  - SQLTimeoutException
- SQLRecoverableException

  SQLRecoverableException がスローされるのは、以前に失敗した操作が、アプリケーションがリカバリー・ステップを実行してトランザクションを再試行すれば成功する可能性がある場合です。SQLRecoverableException がスローされると、接続は無効になります。

- SQLClientInfoException

  SQLClientInfoException は、設定できないクライアント・プロパティーが 1 つ以上ある場合に Connection.setClientInfo メソッドによってスローされます。SQLClientInfoException は、設定できないプロパティーを示します。

BatchUpdateException

BatchUpdateException オブジェクトには、SQL ステートメントのバッチの実行中に発生するエラーに関する以下の項目が含まれています。
- エラーについての記述を含む String オブジェクト、または NULL
- 失敗した SQL ステートメントの SQLSTATE を含む String オブジェクト、または NULL
- エラー・コードを含む整数値、またはゼロ
- バッチ内の SQL ステートメントの更新カウントの整数配列、または null
- SQLException オブジェクトへのポインター、または NULL

バッチ全体に対して、1 つの BatchUpdateException がスローされます。少なくとも 1 つの SQLException オブジェクトが、BatchUpdateException オブジェクトにチェーニングされます。 SQLException オブジェクトは、それに対応するステートメントがバッチに追加される順序と同じ順序でチェーニングされます。SQLException オブジェクトをバッチ内のステートメントにマッチングできるように、個々の SQLException オブジェクトのエラー記述フィールドは次のストリングで始まっています。

```
Error for batch element #n:
```

n はバッチ内のステートメントの番号です。

バッチ実行中の SQL 警告は BatchUpdateException をスローしません。警告に関する情報を取得するには、executeBatch メソッドを実行したオブジェクトに対して Statement.getWarnings メソッドを使用します。これで、各 SQLWarning オブジェクトのエラーに関する記述、SQLSTATE、およびエラー・コードを取得できます。

SQLWarning

SQL ステートメントが正の数の SQLCODE を戻す場合、および SQL ステートメントがゼロ以外の SQLSTATE を持つ 0 の SQLCODE を戻す場合、 IBM Data Server Driver for JDBC and SQLJ は警告を累積します。

getWarnings を呼び出すことによって、SQLWarning オブジェクトを取得できます。

重要: Statement.executeUpdate または PreparedStatement.executeUpdate への呼び出しがどの行にも影響を与えない場合、 IBM Data Server Driver for JDBC and SQLJ はエラー・コード +100 の SQLWarning を生成します。

ResultSet.next への呼び出しで行が戻されない場合、 IBM Data Server Driver for JDBC and SQLJ は SQLWarning を生成しません。

汎用の SQLWarning オブジェクトには、以下の情報が含まれています。
- 警告についての記述を含む String オブジェクト、または NULL
- SQLSTATE を含む String オブジェクト、または NULL

- エラー・コードを含む int 値
- 次の SQLWarning へのポインター、または NULL

IBM Data Server Driver for JDBC and SQLJ の下では、SQLException オブジェクトと同様に、SQLWarning オブジェクトにも DB2 固有の情報を含めることができます。 SQLWarning オブジェクトの DB2 固有の情報は、SQLException オブジェクトの DB2 固有の情報と同じです。

IBM Data Server Driver for JDBC and SQLJ 使用時の SQLException の処理

すべての Java プログラムに共通する点ですが、JDBC アプリケーションの場合も、エラー処理は、try/catch ブロックで実行します。エラーの発生時にメソッドが例外をスローすると、catch ブロックのコードがその例外を処理します。

IBM Data Server Driver for JDBC and SQLJ で実行する JDBC プログラムで SQLException を処理するための基本手順は、以下のとおりです。

1. com.ibm.db2.jcc.DB2Diagnosable インターフェースと com.ibm.db2.jcc.DB2Sqlca クラスに対するアクセス権をプログラムに与えます。 そのインターフェースとクラスへのすべての参照を完全に修飾するか、そのインターフェースとクラスをインポートできます。

   ```
   import com.ibm.db2.jcc.DB2Diagnosable;
   import com.ibm.db2.jcc.DB2Sqlca;
   ```

2. オプション: SQLException.getMessage 呼び出しから完全なメッセージ・テキストを取得する場合は、 DB2 for z/OS または IBM Informix Dynamic Server (IDS) のデータ・ソースに接続するときに、retrieveMessagesFromServerOnGetMessage プロパティーを true に設定します。
3. オプション: SQLException.getMessage 呼び出しから SQL GET DIAGNOSTICS ステートメントによって提供される情報と同様の詳細な診断情報を入手する場合は、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続を DB2 for z/OS データ・ソースに接続するときに、extendedDiagnosticLevel プロパティーを EXTENDED_DIAG_MESSAGE_TEXT (241) に設定します。
4. SQLException を生成できるコードを try ブロックに配置します。
5. catch ブロックのループの中で以下の手順を実行します。
   a. 最後の SQLException を取得したかどうかをテストします。そうでない場合は、次の手順に進みます。
   b. オプション: IDS データ・ソースで実行する SQL ステートメントの場合は、com.ibm.db2.jcc.DB2Statement.getIDSSQLStatementOffSet メソッドを実行して、構文エラーがある列を判別します。

      DB2Statement.getIDSSQLStatementOffSet は、SQL ステートメントに含まれている最初の構文エラーのオフセットを返します。
   c. オプション: IDS データ・ソースに対して実行される SQL ステートメントの場合、SQLException.getCause メソッドを実行して、ISAM エラー・メッセージを検索します。
      1) SQLException.getCause によって戻される Throwable が NULL でない場合、以下の手順のいずれか 1 つを実行します。

- SQLException.printStackTrace を発行することにより、ISAM エラー・メッセージ・テキストを含むエラー・メッセージを出力します。 ISAM エラー・メッセージ・テキストは ″Caused by:″ というストリングの後にあります。
- 次のようにして、ISAM メッセージのエラー・コードとメッセージ・テキストを検索します。
  a) Throwable が SQLException のインスタンスであるかどうかを検査します。そうであれば、その SQLException から SQL エラー・コードを検索します。
  b) Throwable.getMessage メソッドを実行して、ISAM メッセージのテキストを検索します。

d. SQLException が DB2Diagnosable のインスタンスであるかどうかをテストして、 IBM Data Server Driver for JDBC and SQLJ のみの情報が存在するかどうかを確認します。 存在する場合は、以下のようにします。
   1) そのオブジェクトを DB2Diagnosable オブジェクトにキャストします。
   2) 省略可能: DB2Diagnosable.printTrace メソッドを呼び出して、すべての SQLException 情報を java.io.PrintWriter オブジェクトに書き出します。
   3) DB2Diagnosable.getThrowable メソッドを呼び出して、基礎になっている java.lang.Throwable が SQLException の原因かどうかを確認します。
   4) DB2Diagnosable.getSqlca メソッドを呼び出して、DB2Sqlca オブジェクトを取得します。
   5) DB2Sqlca.getSqlCode メソッドを呼び出して、SQL エラー・コード値を取得します。
   6) DB2Sqlca.getSqlErrmc メソッドを呼び出して、すべての SQLERRMC 値が含まれているストリングを取得するか、DB2Sqlca.getSqlErrmcTokens メソッドを呼び出して、SQLERRMC 値を配列として取得します。
   7) DB2Sqlca.getSqlErrp メソッドを呼び出して、SQLERRP 値を取得します。
   8) DB2Sqlca.getSqlErrd メソッドを呼び出して、SQLERRD 値を配列として取得します。
   9) DB2Sqlca.getSqlWarn メソッドを呼び出して、SQLWARN 値を配列として取得します。
   10) DB2Sqlca.getSqlState メソッドを呼び出して、SQLSTATE 値を取得します。
   11) DB2Sqlca.getMessage メソッドを呼び出して、データ・ソースからエラー・メッセージ・テキストを取得します。

e. SQLException.getNextException メソッドを呼び出して、次の SQLException を取得します。

IBM Data Server Driver for JDBC and SQLJ に用意されている SQLException から IBM Data Server Driver for JDBC and SQLJ 固有の情報を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

図 23. IBM Data Server Driver for JDBC and SQLJ 使用時の SQLException の処理

```
import java.sql.*;                      // Import JDBC API package
import com.ibm.db2.jcc.DB2Diagnosable;  // Import packages for DB2       1
import com.ibm.db2.jcc.DB2Sqlca;        // SQLException support
java.io.PrintWriter printWriter;        // For dumping all SQLException
                                        // information
String url = "jdbc:db2://myhost:9999/myDB:" +                          2
  "retrieveMessagesFromServerOnGetMessage=true;";
                                  // Set properties to retrieve full message
                                  // text
String user = "db2adm";
String password = "db2adm";
java.sql.Connection con =
  java.sql.DriverManager.getConnection (url, user, password)
                                  // Connect to a DB2 for z/OS data source


...
try {                                                                  4
  // Code that could generate SQLExceptions
 ...
} catch(SQLException sqle) {
    while(sqle != null) {             // Check whether there are more    5a
                                      // SQLExceptions to process
 //=====> Optional IBM Data Server Driver for JDBC and SQLJ-only
 // error processing
      if (sqle instanceof DB2Diagnosable) {                            5d
      // Check if IBM Data Server Driver for JDBC and SQLJ-only
      // information exists
          com.ibm.db2.jcc.DB2Diagnosable diagnosable =
            (com.ibm.db2.jcc.DB2Diagnosable)sqle;                      5d1
          diagnosable.printTrace (printWriter, "");                    5d2
          java.lang.Throwable throwable =
            diagnosable.getThrowable();                                5d3
          if (throwable != null) {
            // Extract java.lang.Throwable information
            // such as message or stack trace.
            ...
          }
          DB2Sqlca sqlca = diagnosable.getSqlca();                     5d4
                                          // Get DB2Sqlca object
          if (sqlca != null) {            // Check that DB2Sqlca is not null
            int sqlCode = sqlca.getSqlCode(); // Get the SQL error code 5d5
            String sqlErrmc = sqlca.getSqlErrmc();                     5d6
                                          // Get the entire SQLERRMC
            String[] sqlErrmcTokens = sqlca.getSqlErrmcTokens();
                                          // You can also retrieve the
                                          // individual SQLERRMC tokens
            String sqlErrp = sqlca.getSqlErrp();                       5d7
                                          // Get the SQLERRP
            int[] sqlErrd = sqlca.getSqlErrd();                        5d8
                                          // Get SQLERRD fields
            char[] sqlWarn = sqlca.getSqlWarn();                       5d9
                                          // Get SQLWARN fields
            String sqlState = sqlca.getSqlState();                     5d10
                                          // Get SQLSTATE
            String errMessage = sqlca.getMessage();                    5d11
                                          // Get error message

            System.err.println ("Server error message: " + errMessage);

            System.err.println ("--------------- SQLCA ---------------");
            System.err.println ("Error code: " + sqlCode);
            System.err.println ("SQLERRMC: " + sqlErrmc);
            If (sqlErrmcTokens != null) {
```

```
            for (int i=0; i< sqlErrmcTokens.length; i++) {
              System.err.println ("  token " + i + ": " + sqlErrmcTokens[i]);
            }
          }
          System.err.println ( "SQLERRP: " + sqlErrp );
          System.err.println (
            "SQLERRD(1): " + sqlErrd[0] + "¥n" +
            "SQLERRD(2): " + sqlErrd[1] + "¥n" +
            "SQLERRD(3): " + sqlErrd[2] + "¥n" +
            "SQLERRD(4): " + sqlErrd[3] + "¥n" +
            "SQLERRD(5): " + sqlErrd[4] + "¥n" +
            "SQLERRD(6): " + sqlErrd[5] );
          System.err.println (
            "SQLWARN1: " + sqlWarn[0] + "¥n" +
            "SQLWARN2: " + sqlWarn[1] + "¥n" +
            "SQLWARN3: " + sqlWarn[2] + "¥n" +
            "SQLWARN4: " + sqlWarn[3] + "¥n" +
            "SQLWARN5: " + sqlWarn[4] + "¥n" +
            "SQLWARN6: " + sqlWarn[5] + "¥n" +
            "SQLWARN7: " + sqlWarn[6] + "¥n" +
            "SQLWARN8: " + sqlWarn[7] + "¥n" +
            "SQLWARN9: " + sqlWarn[8] + "¥n" +
            "SQLWARNA: " + sqlWarn[9] );
          System.err.println ("SQLSTATE: " + sqlState);
                                          // portion of SQLException
        }
      sqle=sqle.getNextException();     // Retrieve next SQLException      5e
    }
}
```

IBM Data Server Driver for JDBC and SQLJ 使用時の SQLWarning の処理

SQL エラーとは異なり、SQL 警告の場合は、JDBC メソッドから例外がスローされません。その代わり、
Connection、Statement、PreparedStatement、CallableStatement、ResultSet の各クラスには、getWarnings メソッドが含まれています。SQL ステートメントの実行後にそのメソッドを呼び出して、SQL 警告が生成されているかどうかを確認する必要があります。

SQL 警告の情報を取得するための基本手順は、以下のとおりです。

1. オプション: データベース・サーバーに接続するときに、SQLWarning オブジェクトに影響するプロパティーを設定します。

   SQLWarning.getMessage の呼び出しを実行するときに、 DB2 for z/OS または IBM Informix Dynamic Server (IDS) のデータ・ソースから完全なメッセージ・テキストを取得する場合は、retrieveMessagesFromServerOnGetMessage プロパティーを true に設定します。

   DB2 for z/OS データ・ソースへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用しており、SQLWarning.getMessage 呼び出しを実行すると SQL GET DIAGNOSTICS ステートメントによって提供される情報と同様の詳細な診断情報を入手するようにしたい場合は、extendedDiagnosticLevel プロパティーを EXTENDED_DIAG_MESSAGE_TEXT (241) に設定します。
2. データベース・サーバーに接続するか SQL ステートメントを実行するメソッドを呼び出した直後に、getWarnings メソッドを呼び出して、SQLWarning オブジェクトを取得します。

3. ループの中で以下の手順を実行します。
   a. SQLWarning オブジェクトが NULL かどうかをテストします。そうでない場合は、次の手順に進みます。
   b. SQLWarning.getMessage メソッドを呼び出して、警告に関する記述を取得します。
   c. SQLWarning.getSQLState メソッドを呼び出して、SQLSTATE 値を取得します。
   d. SQLWarning.getErrorCode メソッドを呼び出して、エラー・コード値を取得します。
   e. DB2 固有の警告情報を取得する場合は、SQLException に関する DB2 固有の情報を取得する場合と同じ手順を実行します。
   f. SQLWarning.getNextWarning メソッドを呼び出して、次の SQLWarning を取得します。

汎用の SQLWarning 情報を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
String url = "jdbc:db2://myhost:9999/myDB:" +                          1
  "retrieveMessagesFromServerOnGetMessage=true;";
                                  // Set properties to retrieve full message
                                  // text
String user = "db2adm";
String password = "db2adm";
java.sql.Connection con =
  java.sql.DriverManager.getConnection (url, user, password)
                                  // Connect to a DB2 for z/OS data source
Statement stmt;
ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();     // Create a Statement object
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Get the result table from the query
sqlwarn = stmt.getWarnings();     // Get any warnings generated           2
while (sqlwarn != null) {         // While there are warnings, get and    3a
                                  // print warning information
  System.out.println ("Warning description: " + sqlwarn.getMessage());   3b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());             3c
  System.out.println ("Error code: " + sqlwarn.getErrorCode());         3d
  sqlwarn=sqlwarn.getNextWarning();     // Get next SQLWarning           3f
}
```

図 24. SQLWarning を処理する例

BatchUpdateException からの情報の取り出し

バッチに含まれている 1 つのステートメントの実行中にエラーが発生しても、処理は続行します。ただし、executeBatch から BatchUpdateException がスローされます。

BatchUpdateException から情報を取り出すには、以下の手順を実行します。

1. BatchUpdateException.getUpdateCounts メソッドを使用して、例外がスローされる前に、バッチに含まれている各 SQL ステートメントによって更新された行の数を確認します。

   getUpdateCount からは、バッチに含まれている各ステートメントに対応するエレメントが配列として返されます。各エレメントには、以下のいずれかの値が入ります。

   n　　ステートメントによって更新された行の数。

   Statement.SUCCESS_NO_INFO
   この値は、更新された行の数を確認できない場合に返されます。

   Statement.EXECUTE_FAILED
   この値は、ステートメントが正常に実行されなかった場合に返されます。

2. バッチ・ステートメントから自動生成キーを返す可能性があるなら、
   a. BatchUpdateException を com.ibm.db2.jcc.DBBatchUpdateException にキャストします。
   b. DBBatchUpdateException.getDBGeneratedKeys メソッドを呼び出すことにより、バッチ SQL ステートメントの各実行に対応する自動生成キーを含む ResultSet オブジェクトの配列を取得します。
   c. その配列の各 ResultSet がヌルかどうかをテストします。

      各 ResultSet の内容は以下のとおりです。

      - ResultSet がヌルでない場合、そこには、バッチ SQL ステートメントの実行の自動生成キーが含まれています。
      - ResultSet がヌルの場合、バッチ・ステートメントの実行は失敗します。

3. SQLException のメソッド getMessage、getSQLState、getErrorCode を使用して、最初のエラーの記述、SQLSTATE、エラー・コードを取得します。
4. BatchUpdateException.getNextException メソッドを使用して、チェーンになっている SQLException を取得します。
5. ループの中で、getMessage、getSQLState、getErrorCode、getNextException の各メソッド呼び出しを実行して、SQLException に関する情報を取得し、次の SQLException を取得します。

自動生成キーを返すバッチ・ステートメントについて、BatchUpdateException の各フィールドと、連鎖的につながっている SQLException オブジェクトを取得するコード断片を以下に示します。この例では、自動生成キーには 1 つしか列が含まれていないこと、およびデータ・タイプが数値であるキー値が必ず 1 つあるということを想定しています。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
try {
  // Batch updates
} catch(BatchUpdateException buex) {
    System.err.println("Contents of BatchUpdateException:");
    System.err.println(" Update counts: ");
    int [] updateCounts = buex.getUpdateCounts();            1
    for (int i = 0; i < updateCounts.length; i++) {
      System.err.println("  Statement " + i + ":" + updateCounts[i]);
    }
```

```
    ResultSet[] resultList =
      ((DBBatchUpdateException)buex).getDBGeneratedKeys();    2
    for (i = 0; i < resultList.length; i++)
    {
      if (resultList[i] == null)
         continue; // Skip the ResultSet for which there was a failure
      else {
        rs.next();
        java.math.BigDecimal idColVar = rs.getBigDecimal(1);
                                        // Get automatically generated key
                                        // value
        System.out.println("Automatically generated key value = " + idColVar);
      }
    }
    System.err.println(" Message: " + buex.getMessage());     3
    System.err.println(" SQLSTATE: " + buex.getSQLState());
    System.err.println(" Error code: " + buex.getErrorCode());
    SQLException ex = buex.getNextException();                4
    while (ex != null) {                                      5
      System.err.println("SQL exception:");
      System.err.println(" Message: " + ex.getMessage());
      System.err.println(" SQLSTATE: " + ex.getSQLState());
      System.err.println(" Error code: " + ex.getErrorCode());
      ex = ex.getNextException();
    }
}
```

DB2 JDBC Type 2 ドライバー (非推奨) 使用時の SQLException の処理

すべての Java プログラムに共通する点ですが、Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーの使用時も、エラー処理は、try/catch ブロックで実行します。エラーの発生時にメソッドが例外をスローすると、catch ブロックのコードがその例外を処理します。

Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーは、エラー処理のために SQLException クラスを使用します。すべての JDBC メソッドは、実行中にエラーが発生すると SQLException のインスタンスをスローします。JDBC 仕様に従って、 SQLException オブジェクトには以下の情報が含まれます。

- エラーについての記述を含む String オブジェクト、または NULL
- SQLSTATE を含む String オブジェクト、または NULL
- エラー・コードを含む int 値
- 次の SQLException へのポインター、または NULL

Linux、UNIX、Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) で実行する JDBC プログラムで SQLException を処理するための基本手順は、以下のとおりです。

1. SQLException を生成できるコードを try ブロックに配置します。
2. catch ブロックのループの中で以下の手順を実行します。
   a. 最後の SQLException を取得したかどうかをテストします。そうでない場合は、次の手順に進みます。
   b. SQLException からエラー情報を取得します。
   c. SQLException.getNextException メソッドを呼び出して、次の SQLException を取得します。

DB2 JDBC Type 2 ドライバーに用意されている DB2 バージョンの SQLException を使用した catch ブロックのコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
import java.sql.*;                        // Import JDBC API package
...
try {
  // Code that could generate SQLExceptions                              1
 ...
} catch(SQLException sqle) {
    while(sqle != null) {              // Check whether there are more    2a
      System.out.println("Message:  " + sqle.getMessage());               2b
      System.out.println("SQLSTATE: " + sqle.getSQLState());
      System.out.println("SQL error code: " + sqle.getErrorCode());
      sqle=sqle.getNextException();     // Retrieve next SQLException     2c
    }
}
```

図 25. IBM Data Server Driver for JDBC and SQLJ 使用時の SQLException の処理

DB2 JDBC Type 2 ドライバー使用時の SQLWarning の処理

SQL エラーとは異なり、SQL 警告の場合は、JDBC メソッドから例外がスローされません。その代わり、
Connection、Statement、PreparedStatement、CallableStatement、ResultSet の各クラスには、getWarnings メソッドが含まれています。SQL ステートメントの実行後にそのメソッドを呼び出して、SQL 警告が生成されているかどうかを確認する必要があります。

getWarnings を呼び出すことによって、SQLWarning オブジェクトを取得できます。

Linux、UNIX、Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) は、汎用の SQLWarning オブジェクトを生成します。汎用の SQLWarning オブジェクトには、以下の情報が含まれています。

- 警告についての記述を含む String オブジェクト、または NULL
- SQLSTATE を含む String オブジェクト、または NULL
- エラー・コードを含む int 値
- 次の SQLWarning へのポインター、または NULL

SQL 警告の情報を取得するための基本手順は、以下のとおりです。

1. SQL ステートメントを実行するメソッドを呼び出した直後に、getWarnings メソッドを呼び出して、SQLWarning オブジェクトを取得します。
2. ループの中で以下の手順を実行します。
   a. SQLWarning オブジェクトが NULL かどうかをテストします。そうでない場合は、次の手順に進みます。
   b. SQLWarning.getMessage メソッドを呼び出して、警告に関する記述を取得します。
   c. SQLWarning.getSQLState メソッドを呼び出して、SQLSTATE 値を取得します。
   d. SQLWarning.getErrorCode メソッドを呼び出して、エラー・コード値を取得します。

e. SQLWarning.getNextWarning メソッドを呼び出して、次の SQLWarning を取得します。

汎用の SQLWarning 情報を取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
Connection con;
Statement stmt;
ResultSet rs;
SQLWarning sqlwarn;
...
stmt = con.createStatement();     // Create a Statement object
rs = stmt.executeQuery("SELECT * FROM EMPLOYEE");
                                  // Get the result table from the query
sqlwarn = stmt.getWarnings();     // Get any warnings generated          1
while (sqlwarn != null) {         // While there are warnings, get and   2a
                                  // print warning information
  System.out.println ("Warning description: " + sqlwarn.getMessage());   2b
  System.out.println ("SQLSTATE: " + sqlwarn.getSQLState());             2c
  System.out.println ("Error code: " + sqlwarn.getErrorCode());         2d
  sqlwarn=sqlwarn.getNextWarning();     // Get next SQLWarning           2e
}
```

図 26. SQLWarning の処理

DB2 Database for Linux, UNIX, and Windows および IDS に対する JDBC および SQLJ クライアント・リルート・サポート

DB2 Database for Linux, UNIX, and Windows および IBM Informix Dynamic Server の自動クライアント・リルート機能を使用することにより、クライアント・アプリケーションがサーバーとの通信の消失からリカバリーして、最小限の中断で処理を続けることができます。 JDBC および SQLJ クライアント・アプリケーションはそのサポートを利用できます。

サーバー・クラッシュが発生すると、そのサーバーに接続している各クライアントは通信エラーを受け取り、接続が終了してアプリケーション・エラーが発生します。可用性が重要である場合には、予備セットアップまたはフェイルオーバー・サポートが必要です。フェイルオーバーは、別のサーバーが失敗するときに、操作を引き継ぐサーバーの機能です。どちらの場合も、 IBM Data Server Driver for JDBC and SQLJ クライアントは元のサーバーまたは新規サーバーへの接続を再確立しようとします。接続が再確立されると、アプリケーションは次のトランザクションを続行できます。

DB2 Database for Linux, UNIX, and Windows および IBM Informix Dynamic Server に対する IBM Data Server Driver for JDBC and SQLJ クライアント・リルート・サポートは、
javax.sql.DataSource、javax.sql.ConnectionPoolDataSource、javax.sql.XADataSource、または java.sql.DriverManager インターフェースを使用して得られる接続に有効です。

制約事項: IBM Data Server Driver for JDBC and SQLJ は代替 DB2 Connect ゲートウェイのクライアント・リルートをサポートしていません。ただし、以下の例外があります。

クライアント・サイドで代替 DB2 Connect ゲートウェイを以下のいずれかの方法で設定した場合、ドライバーは初回接続時のみ、代替 DB2 Connect ゲートウェイにフェイルオーバーします。

- DB2ClientRerouteServerList インスタンスを経由する方法
- clientRerouteAlternateServerName および clientRerouteAlternatePortNumber プロパティーの設定による方法

制約事項: IBM Data Server Driver for JDBC and SQLJ は、トラステッド接続に関連したユーザーを認証なしで切り替える場合、クライアント・リルートをサポートしません。認証なしでトラステッド・ユーザーの切り替えをプログラムが実行した後で 1 次サーバーがダウンすると、代替サーバーへのクライアント・リルートは失敗し、SQLException がスローされます。

制約事項: DriverManager インターフェースで確立される接続用のクライアント・リルート・サポートには、以下の制限があります。

- 同じ URL およびプロパティーを指定して接続を作成する場合にのみ、代替サーバー情報が DriverManager 接続間で共有されます。
- DriverManager 接続には clientRerouteServerListJNDIName プロパティーまたは clientRerouteServerListJNDIContext プロパティーを設定できません。
- クライアント・リルートはデフォルト接続 (jdbc:default:connection) には使用可能にされていません。

DB2 Database for Linux, UNIX, and Windows サーバーとの接続のための IBM Data Server Driver for JDBC and SQLJ のクライアント・リルートを有効にする

DB2 Database for Linux, UNIX, and Windows サーバーに接続する IBM Data Server Driver for JDBC and SQLJ アプリケーションのためにクライアント・リルートを有効にするには、どのサーバーが 1 次サーバーで、どのサーバーが代替サーバーかの情報をドライバーに提供するためのプロパティーを設定する必要があります。

DB2 Database for Linux, UNIX, and Windows との接続のための IBM Data Server Driver for JDBC and SQLJ クライアント・リルートを有効にするには、その前に、インストール・システムに 1 次サーバーと代替サーバーがセットアップされていなければなりません。

1 次サーバーと代替サーバーがセットアップされたなら、以下のステップを実行します。

1. 1 次および代替の各サーバーのアドレスを指定するため、該当するプロパティーを設定します。
   - アプリケーションが接続のために DriverManager インターフェースを使用している場合、
     a. 接続 URL で使用する 1 次サーバーのサーバー名とポート番号を指定します。

b. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber の各プロパティーに、使用する代替サーバーのサーバー名とポート番号をそれぞれ設定します。

- アプリケーションで接続のために DataSource インターフェースを使用している場合は、以下のいずれかの方法を使用します。
  - DataSource のプロパティーにサーバー名とポート番号を設定します。
    a. serverName および portNumber の各プロパティーに、使用する 1 次サーバーのサーバー名とポート番号をそれぞれ設定します。
    b. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber の各プロパティーに、使用する代替サーバーのサーバー名とポート番号をそれぞれ設定します。
  - DB2ClientRerouteServerList のインスタンスを使用して 1 次サーバーと代替サーバーを指定することにより、クライアント・リルートのための JNDI を構成します。
    a. DB2ClientRerouteServerList のインスタンスを作成します。
    b. com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName および com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber の各プロパティーに、使用する 1 次サーバーのサーバー名とポート番号をそれぞれ設定します。
    c. com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName および com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber の各プロパティーに、使用する代替サーバーのサーバー名とポート番号をそれぞれ設定します。
    d. DB2ClientRerouteServerList を持続的にします。
       1) DB2ClientRerouteServerList のインスタンスを JNDI レジストリーにバインドします。
       2) DB2ClientRerouteServerList オブジェクトの JNDI 名を IBM Data Server Driver for JDBC and SQLJの clientRerouteServerListJNDIName プロパティーに代入します。

2. 1 次サーバーを表すデータ・ソースに接続します。

例

1 次サーバーと代替サーバーのサーバー名とポート番号が、それぞれ次のようになっているとします。

サーバー名	ポート番号
srv1.sj.ibm.com	50000
srv3.sj.ibm.com	50002

以下のコードは、アプリケーションの中で DataSource のプロパティーをセットアップし、アプリケーションが 1 次サーバーとして srv1.sj.ibm.com に、また代替サーバーとして srv3.sj.ibm.com に接続するようにします。つまり、srv1.sj.ibm.com が初期接続中にダウンしているなら、ドライバーは srv3.sj.ibm.com に接続することになります。

```
ds.setDriverType(4);
ds.setServerName("srv1.sj.ibm.com");
ds.setPortNumber("50000");
ds.setClientRerouteAlternateServerName("srv3.sj.ibm.com");
ds.setClientRerouteAlternatePortNumber("50002");
```

以下のコードでは、クライアント・リルートのための JNDI を構成しています。DB2ClientRerouteServerList のインスタンスを作成し、そのインスタンスを JNDI レジストリーにバインドしてから、DB2ClientRerouteServerList オブジェクトの JNDI 名を clientRerouteServerListJNDIName プロパティーに代入します。

```
// Create a starting context for naming operations
InitialContext registry = new InitialContext();
// Create a DB2ClientRerouteServerList object
DB2ClientRerouteServerList address = new DB2ClientRerouteServerList();

// Set the port number and server name for the primary server
address.setPrimaryPortNumber(50000);
address.setPrimaryServerName("srv1.sj.ibm.com");

// Set the  port number and server name for the alternate server
int[] port = {50002};
String[] server = {"srv3.sj.ibm.com"};
address.setAlternatePortNumber(port);
address.setAlternateServerName(server);

registry.rebind("serverList", address);
// Assign the JNDI name of the DB2ClientRerouteServerList object to the
// clientRerouteServerListJNDIName property
datasource.setClientRerouteServerListJNDIName("serverList");
```

IDS サーバーとの接続のための IBM Data Server Driver for JDBC and SQLJのクライアント・リルートを有効にする

IBM Informix Dynamic Server (IDS) サーバーに接続する IBM Data Server Driver for JDBC and SQLJ アプリケーションのためのクライアント・リルートを有効にするには、接続マネージャーに接続する必要があります。

IBM Informix Dynamic Server に接続するための IBM Data Server Driver for JDBC and SQLJ クライアント・リルートを有効にするには、その前に、インストール・システムに 1 つ以上の接続マネージャー、1 つの 1 次サーバー、そして 1 つ以上の代替サーバーがセットアップされていなければなりません。

接続マネージャー、1 次サーバー、および代替サーバーがセットアップされたなら、以下のステップを実行します。

1. 接続マネージャーのアドレスを指定するため、該当するプロパティーを設定します。
   - クライアント・リルートのために単一の接続マネージャーを使用している場合、アプリケーションで接続のために DataSource インターフェースを使用しているなら、serverName および portNumber の各プロパティーに、接続マネージャーのサーバー名およびポート番号をそれぞれ設定します。
   - クライアント・リルートのために単一の接続マネージャーを使用している場合、アプリケーションで接続のために DriverManager インターフェースを使用しているなら、接続 URL 中に接続マネージャーのサーバー名とポート番号を指定します。

- クライアント・リルートのために複数の接続マネージャーを使用している場合、アプリケーションで接続のために DriverManager インターフェースを使用しているなら、
  a. 接続 URL に、使用するメイン接続マネージャーのサーバー名とポート番号を指定します。
  b. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber の各プロパティーに、使用する代替接続マネージャーのサーバー名およびポート番号をそれぞれ設定します。
- クライアント・リルートのために複数の接続マネージャーを使用している場合、アプリケーションで接続のために DataSource インターフェースを使用しているなら、以下のいずれかの手法を使用してください。
  – DataSource のプロパティーにサーバー名とポート番号を設定します。
    a. serverName および portNumber の各プロパティーに、使用するメイン接続マネージャーのサーバー名およびポート番号をそれぞれ設定します。
    b. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber の各プロパティーに、使用する代替接続マネージャーのサーバー名およびポート番号をそれぞれ設定します。
  – DB2ClientRerouteServerList のインスタンスを使用してメイン接続マネージャーと代替接続マネージャーを指定することにより、クライアント・リルートのための JNDI を構成します。
    a. DB2ClientRerouteServerList のインスタンスを作成します。
    b. com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName および com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber の各プロパティーに、使用するメイン接続マネージャーのサーバー名およびポート番号をそれぞれ設定します。
    c. com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName および com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber の各プロパティーに、使用する代替接続マネージャーのサーバー名およびポート番号をそれぞれ設定します。
    d. DB2ClientRerouteServerList を持続的にします。
      1) DB2ClientRerouteServerList のインスタンスを JNDI レジストリーにバインドします。
      2) DB2ClientRerouteServerList オブジェクトの JNDI 名を IBM Data Server Driver for JDBC and SQLJの clientRerouteServerListJNDIName プロパティーに代入します。

2. メイン接続マネージャーを表すデータ・ソースに接続します。

シームレス・フェイルオーバーが有効でない場合、代替接続マネージャーは、初期接続のためにのみ使用されます。初期接続において、メイン接続マネージャーがダウンしている場合、 IBM Data Server Driver for JDBC and SQLJ は、clientRerouteAlternateServerName と clientRerouteAlternatePortNumber、または com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName と com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber の最初の値セットで指定されている接続マネージャーとの接続を試みます。

例

2 つの接続マネージャーがあり、そのサーバー名とポート番号がそれぞれ次のようになっているとします。

サーバー名	ポート番号
cm1.sf.ibm.com	50000
cm2.sf.ibm.com	50002

以下のコードは、アプリケーションの中で DataSource のプロパティーをセットアップし、アプリケーションがメイン接続マネージャーとして cm2.sf.ibm.com に、また代替接続マネージャーとして cm1.sf.ibm.com として接続するようにします。つまり、cm2.sf.ibm.com が初期接続中にダウンしているなら、ドライバーは cm1.sf.ibm.com に接続することになります。

```
ds.setDriverType(4);
ds.setServerName("cm2.sf.ibm.com");
ds.setPortNumber("50002");
ds.setClientRerouteAlternateServerName("cm1.sf.ibm.com");
ds.setClientRerouteAlternatePortNumber("50000");
```

以下のコードでは、クライアント・リルートのための JNDI を構成しています。DB2ClientRerouteServerList のインスタンスを作成し、そのインスタンスを JNDI レジストリーにバインドしてから、DB2ClientRerouteServerList オブジェクトの JNDI 名を clientRerouteServerListJNDIName プロパティーに代入します。

```
// Create a starting context for naming operations
InitialContext registry = new InitialContext();
// Create a DB2ClientRerouteServerList object
DB2ClientRerouteServerList address = new DB2ClientRerouteServerList();

// Set the port number and server name for the main Configuration Manager
address.setPrimaryPortNumber(50002);
address.setPrimaryServerName("cm2.sf.ibm.com");

// Set the  port number and server name for the alternative Configuration Manager
int[] port = {50000};
String[] server = {"cm1.sf.ibm.com"};
address.setAlternatePortNumber(port);
address.setAlternateServerName(server);

registry.rebind("serverList", address);
// Assign the JNDI name of the DB2ClientRerouteServerList object to the
// clientRerouteServerListJNDIName property
datasource.setClientRerouteServerListJNDIName("serverList");
```

DB2 Database for Linux, UNIX, and Windows および IDS に対する JNDI による JDBC および SQLJ クライアント・リルート・サポート

javax.sql.DataSource、javax.sql.ConnectionPoolDataSource、または javax.sql.XADataSource インターフェースで接続する場合、clientRerouteServerListJNDIName プロパティーを設定することによりクライアント・リルート用に JNDI が構成されることを IBM Data Server Driver for JDBC and SQLJ に示します。

また、clientRerouteServerListJNDIContext プロパティーを指定することもできます。このプロパティーは、DB2ClientRerouteServerList インスタンスのバインディングおよび検索に使用される JNDI コンテキストを提供します。

JNDI によるクライアント・リルートの場合、メモリー内の 1 次サーバーおよび代替サーバーの情報が、javax.naming.Referenceable インターフェースをインプリメントする DB2ClientRerouteServerList クラスのインスタンスです。

DB2ClientRerouteServerList は、以下のプロパティーを備えた、シリアライズ可能な Java Bean です。

プロパティー名	データ・タイプ
com.ibm.db2.jcc.DB2ClientRerouteServerList.alternateServerName	String[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.alternatePortNumber	int[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryServerName	String[]
com.ibm.db2.jcc.DB2ClientRerouteServerList.primaryPortNumber	int[]

getXXX および setXXX メソッドは、プロパティーごとに定義されます。

DataSource がクライアント・リルート代替情報を保管するために JNDI を使用するように構成されている場合、DataSource の標準サーバーおよびポート・プロパティーは getConnection 要求には使用されません。その代わりに、1 次サーバー・アドレスが一時 clientRerouteServerList 情報から取得されます。 JNDI のバインドまたは検索の失敗のために JNDI ストアが使用できない場合、 IBM Data Server Driver for JDBC and SQLJ は DataSource の標準サーバーおよびポート・プロパティーを使用して接続を試行します。 JNDI のバインドまたは検索の失敗が発生したことを示すために、警告が累積されます。

フェイルオーバーの後:

- IBM Data Server Driver for JDBC and SQLJ は、更新されたサーバー情報を JNDI ストアに伝搬しようとします。
- DB2ClientRerouteServerList に指定された primaryServerName および primaryPortNumber の値が接続に使用されます。primaryServerName が指定されていない場合、DataSource インスタンスの serverName 値が使用されます。

DB2 Database for Linux, UNIX, and Windows および IDS に対する JDBC および SQLJ クライアント・リルート操作

IBM Data Server Driver for JDBC and SQLJ クライアント・リルート・サポートが使用可能になっている場合、 DB2 Database for Linux, UNIX, and Windows サーバーに接続している Java アプリケーションは、1 次サーバーに障害が起こった場合でも引き続き実行できます。

DB2 サーバーに接続している Java アプリケーションのクライアント・リルートは、カスケード・フェイルオーバーのためのクライアント・アフィニティーのサポートが使用不可になっている場合、以下のように機能します。

1. 各データ・ソースへの接続時に、 IBM Data Server Driver for JDBC and SQLJ は 1 次サーバーおよび代替サーバーの情報を取得します。

- DB2 Database for Linux, UNIX, and Windows サーバーへの最初の接続時には、以下が行われます。
  a. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber のプロパティーが設定されると、 IBM Data Server Driver for JDBC and SQLJ は、1 次サーバー値 serverName および portNumber と共に、これらの値を代替サーバー値としてメモリーにロードします。
  b. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber のプロパティーが設定されておらず、JNDI ストアが DB2BaseDataSource にプロパティー clientRerouteServerListJNDIName を設定して構成されている場合、 IBM Data Server Driver for JDBC and SQLJ は 1 次サーバーおよび代替サーバーの情報を JNDI ストアからメモリーにロードします。
  c. DataSource プロパティーが代替サーバーに設定されておらず、JNDI が構成されていない場合、 IBM Data Server Driver for JDBC and SQLJ は DNS 表を調べて、1 次サーバーおよび代替サーバーの情報を確認します。DNS 情報がある場合、 IBM Data Server Driver for JDBC and SQLJ はそれらの値をメモリーにロードします。
  d. 1 次サーバーおよび代替サーバーの情報が使用可能でない場合、接続を確立できません。また、 IBM Data Server Driver for JDBC and SQLJ は例外をスローします。
- IBM Informix Dynamic Server への最初の接続時には、以下が行われます。
  a. アプリケーションは、初期接続のサーバーとポートを指定します。それらの値は、接続マネージャーを識別します。
  b. IBM Data Server Driver for JDBC and SQLJ は、接続マネージャーからの情報を使用して、1 次サーバーと代替サーバーに関する情報を取得します。 IBM Data Server Driver for JDBC and SQLJ は、それらの値をメモリーにロードします。
  c. 接続マネージャーへの初期接続に失敗した場合は、以下が行われます。
     – clientRerouteAlternateServerName および clientRerouteAlternatePortNumber プロパティーが設定されていると、 IBM Data Server Driver for JDBC and SQLJ は、clientRerouteAlternateServerName および clientRerouteAlternatePortNumber によって識別される接続マネージャーに接続し、1 次サーバーと代替サーバーに関する情報を接続マネージャーから取得します。 IBM Data Server Driver for JDBC and SQLJ は、それらの値を 1 次サーバーと代替サーバーの値としてメモリーにロードします。
     – clientRerouteAlternateServerName および clientRerouteAlternatePortNumber プロパティーが設定されておらず、JNDI ストアが DB2BaseDataSource にプロパティー clientRerouteServerListJNDIName を設定して構成されている場合、 IBM Data Server Driver for JDBC and SQLJ は、DB2ClientRerouteServerList.alternateServerName および DB2ClientRerouteServerList.alternatePortNumber によって識別される接続マネージャーに接続し、1 次サーバーおよび代替サーバーに関する情報をその接続マネージャーから取得します。 IBM Data Server Driver for JDBC and SQLJ は、接続マネージャーからの 1 次サーバーと代替サーバーの情報をメモリーにロードします。

d. clientRerouteAlternateServerName および clientRerouteAlternatePortNumber が設定されておらず、JNDI が構成されていない場合、 IBM Data Server Driver for JDBC and SQLJ は DNS 表を調べて、接続マネージャーのサーバーとポートの情報を確認します。DNS 情報がある場合、 IBM Data Server Driver for JDBC and SQLJ は接続マネージャーに接続し、1 次サーバーと代替サーバーに関する情報を取得して、それらの値をメモリーにロードします。

e. 1 次サーバーおよび代替サーバーの情報が使用可能でない場合、接続を確立できません。また、 IBM Data Server Driver for JDBC and SQLJ は例外をスローします。

- それ以降の接続時には、 IBM Data Server Driver for JDBC and SQLJは 1 次サーバーと代替サーバーの値をドライバー・メモリーから取得します。

2. IBM Data Server Driver for JDBC and SQLJ は 1 次サーバー名とポート番号を使用してデータ・ソースへの接続を試みます。

   接続が DriverManager インターフェースを介している場合、 IBM Data Server Driver for JDBC and SQLJ は、クライアント・リルート処理用に DataSource オブジェクトを作成します。

3. 1 次サーバーへの接続が失敗する場合:

   a. これが最初の接続である場合、 IBM Data Server Driver for JDBC and SQLJ は元の 1 次サーバーへの再接続を試みます。

   b. これが最初の接続でない場合、 IBM Data Server Driver for JDBC and SQLJ は新しい 1 次サーバーへの再接続を試みます (サーバー名とポート番号はサーバーによって提供されます)。

   c. 1 次サーバーへの再接続が失敗すると、 IBM Data Server Driver for JDBC and SQLJ は代替サーバーに接続しようとします。

      これが最初の接続でない場合、最新の代替サーバー・リストを使って次の代替サーバーを検出します。

   1 次サーバーへの再接続のことを*フェイルバック* と呼びます。代替サーバーへの接続は*フェイルオーバー* と呼ばれます。

   IBM Data Server Driver for JDBC and SQLJ は maxRetriesForClientReroute および retryIntervalForClientReroute プロパティーを使用して、接続が再試行された回数および再試行の間の待機時間を判別します。 1 次サーバーおよび代替サーバーへの接続の試行は、1 回の再試行としてカウントされます。

4. 接続が確立されず、maxRetriesForClientReroute および retryIntervalForClientReroute が設定されておらず、DataSource で定義されている元の serverName と portNumber の値が元の接続で使用された serverName と portNumber の値と異なっている場合、DataSource で定義されている serverName と portNumber の値を使用して接続を再試行します。

5. 初期接続中にフェイルオーバーが成功すると、ドライバーは SQLWarning を生成します。最初の接続の後、フェイルオーバーに成功した場合には:

   - シームレス・フェイルオーバーが可能な場合、ドライバーはアプリケーションに通知せずに、新しいサーバー上でトランザクションを再試行します。

シームレス・フェイルオーバーが行われるには、次の条件が満たされる必要があります。

- enableSeamlessFailover プロパティーが DB2BaseDataSource.YES (1) に設定されている。

  Sysplex ワークロード・バランシングが有効になっている場合 (enableSysplexWLB の値が `true`)、enableSeamlessFailover の設定に関係なく、シームレス・フェイルオーバーが試行されます。
- 接続はトランザクションではありません。つまり、失敗はトランザクション内の最初の SQL ステートメントが実行されたときに発生します。
- サーバー上で使用中のグローバル一時表がない。
- 開いた状態の保留カーソルが存在しない。

• シームレス・フェイルオーバーが有効でない場合、接続が自動的に再確立されてトランザクションが暗黙的にロールバックされたことをアプリケーションに通知するために、ドライバーはエラー・コード -4498 の SQLException をアプリケーションにスローします。その後、アプリケーションは、明示的ロールバックをあらかじめ行う必要なく、トランザクションを再試行できます。

  エラー・コード -4498 と共に戻される理由コードは、元の接続で変更されたすべてのデータベース・サーバー特殊レジスターが、フェイルオーバー接続で再確立されるかどうかを示します。

初期接続の確立に代替サーバー情報が使用されたかどうかを判別するには、DB2Connection.alternateWasUsedOnConnect メソッドを呼び出します。

6. フェイルオーバーの後、新しい 1 次サーバーから得られた新しい 1 次サーバーおよび代替サーバーに関する情報を使ってドライバー・メモリーが更新されます。

例

例: DB2 Database for Linux, UNIX, and Windows サーバーへのクライアント・リルート (maxRetriesForClientReroute および retryIntervalForClientReroute が設定されていない場合): データベースへの接続に、以下のプロパティーが設定されているものと想定します。

プロパティー	値
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
clientRerouteAlternateServerName	host2
clientRerouteAlternatePortNumber	port2

以下のステップは、 DB2 Database for Linux, UNIX, and Windows サーバーへの接続の場合のクライアント・リルート・シナリオを示しています。

1. IBM Data Server Driver for JDBC and SQLJ は、1 次サーバー・アドレスとして host1:port1 をメモリーにロードし、代替サーバー・アドレスとして host2:port2 をメモリーにロードします。

2. 初期接続で、ドライバーは host1:port1 への接続を試みます。
3. host1:port1 への接続が失敗します。そのため、ドライバーは host1:port1 への接続をもう一度試みます。
4. host1:port1 への再接続が失敗します。そのため、ドライバーは host2:port2 への接続を試みます。
5. host2:port2 への接続が成功します。
6. ドライバーは、サーバー host2:port2 から受信した代替サーバー情報を取得して、メモリーをその情報で更新します。

   ドライバーは contains host2a:port2a、host2:port2 を含むサーバー・リストを受け取るものと想定します。host2a:port2a は新しい 1 次サーバーとして保管されており、host2:port2 は新しい代替サーバーとして保管されています。この同じ接続、または同じ DataSource から作成された別の接続で別の失敗が検出された場合、ドライバーは新しい 1 次サーバーとして host2a:port2a への接続を試みます。その接続が失敗すると、ドライバーは新しい代替サーバー host2:port2 への接続を試みます。
7. host2:port2 への接続中に通信障害が発生します。
8. ドライバーは host2a:port2a への接続を試みます。
9. host2a:port2a への接続が成功します。
10. ドライバーは、サーバー host2a:port2a から受信した代替サーバー情報を取得して、メモリーをその情報で更新します。

以下のステップは、 IBM Informix Dynamic Server への接続の場合のクライアント・リルート・シナリオを示しています。

1. IBM Data Server Driver for JDBC and SQLJ は、host1:port1 によって識別される接続マネージャーへの接続を試みます。
2. host1:port1 への接続が失敗します。そのため、ドライバーは host2:port2 によって識別される接続マネージャーへの接続を試みます。
3. host2:port2 への接続が成功します。
4. ドライバーは、サーバー host2:port2 から受信した代替サーバー情報を取得して、メモリーをその情報で更新します。

   ドライバーは host1a:port1a、host2a:port2a を含むサーバー・リストを受け取るものと想定します。host1a:port1a は新しい 1 次サーバーとして保管されており、host2a:port2a は新しい代替サーバーとして保管されています。この同じ接続、または同じ DataSource から作成された別の接続で別の失敗が検出された場合、ドライバーは新しい 1 次サーバーとして host1a:port1a への接続を試みます。その接続が失敗すると、ドライバーは新しい代替サーバー host2a:port2a への接続を試みます。
5. ドライバーは host1a:port1a に接続します。
6. host1a:port1a への接続中に障害が発生します。
7. ドライバーは host2a:port2a への接続を試みます。
8. host2a:port2a への接続が成功します。
9. ドライバーは、サーバー host2a:port2a から受信した代替サーバー情報を取得して、メモリーをその情報で更新します。

例: DB2 Database for Linux, UNIX, and Windows サーバーへのクライアント・リルート (複数回の再試行のために maxRetriesForClientReroute および retryIntervalForClientReroute が設定されている場合): データベースへの接続に、以下のプロパティーが設定されているものと想定します。

プロパティー	値
enableClientAffinitiesList	DB2BaseDataSource.NO (2)
serverName	host1
portNumber	port1
clientRerouteAlternateServerName	host2
clientRerouteAlternatePortNumber	port2
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2

以下のステップは、 DB2 Database for Linux, UNIX, and Windows サーバーへの接続の場合のクライアント・リルート・シナリオを示しています。

1. IBM Data Server Driver for JDBC and SQLJ は、1 次サーバー・アドレスとして host1:port1 をメモリーにロードし、代替サーバー・アドレスとして host2:port2 をメモリーにロードします。
2. 初期接続で、ドライバーは host1:port1 への接続を試みます。
3. host1:port1 への接続が失敗します。そのため、ドライバーは host1:port1 への接続をもう一度試みます。
4. host1:port1 への接続が再び失敗します。そのため、ドライバーは host2:port2 への接続を試みます。
5. host2:port2 への接続は失敗します。
6. ドライバーは 2 秒間待機します。
7. ドライバーは host1:port1 への接続を試みますが、失敗します。
8. ドライバーは host2:port2 への接続を試みますが、失敗します。
9. ドライバーは 2 秒間待機します。
10. ドライバーは host1:port1 への接続を試みますが、失敗します。
11. ドライバーは host2:port2 への接続を試みますが、失敗します。
12. ドライバーは 2 秒間待機します。
13. ドライバーはエラー・コード -4499 の SQLException をスローします。

以下のステップは、 IBM Informix Dynamic Server への接続の場合のクライアント・リルート・シナリオを示しています。

1. IBM Data Server Driver for JDBC and SQLJ は、host1:port1 によって識別される接続マネージャーへの接続を試みます。
2. host1:port1 への接続が失敗します。そのため、ドライバーは host2:port2 によって識別される接続マネージャーへの接続を試みます。
3. host2:port2 への接続が成功します。
4. ドライバーは、host2:port2 によって識別される接続マネージャーから代替サーバー情報を取得して、メモリーをその情報で更新します。接続マネージャーは

host1a:port1a を新しい 1 次サーバーとして識別し、host2a:port2a を新しい代替サーバーとして識別するものとします。

5. ドライバーは host1a:port1a への接続を試みます。
6. host1a:port1a への接続は失敗します。
7. ドライバーは host2a:port2a への接続を試みます。
8. host2a:port2a への接続は失敗します。
9. ドライバーは 2 秒間待機します。
10. ドライバーは host1a:port1a への接続を試みます。
11. host1a:port1a への接続は失敗します。
12. ドライバーは host2a:port2a への接続を試みます。
13. host2a:port2a への接続は失敗します。
14. ドライバーは 2 秒間待機します。
15. ドライバーは host1a:port1a への接続を試みます。
16. host1a:port1a への接続は失敗します。
17. ドライバーは host2a:port2a への接続を試みます。
18. host2a:port2a への接続は失敗します。
19. ドライバーは 2 秒間待機します。
20. ドライバーはエラー・コード -4499 の SQLException をスローします。

IBM Data Server Driver for JDBC and SQLJ クライアントから DB2 for z/OS サーバーへの接続の高可用性に対する Sysplex サポート

高可用性に対する Sysplex サポートを使用することにより、クライアント・アプリケーションがサーバーとの通信の消失からリカバリーして、最小限の中断で処理を続けることができます。 Java アプリケーションの場合、このサポートを使用するには IBM Data Server Driver for JDBC and SQLJ Type 4 接続が必要です。

IBM Data Server Driver for JDBC and SQLJ クライアントから DB2 for z/OS サーバーへの接続に対する Sysplex サポートには、以下の機能が含まれます。

自動クライアント・リルート

このサポートは、使用可能な Sysplex メンバーを介してデータベースへの再接続を試みることにより、障害からのクライアントのリカバリーを可能にします。クライアントの自動クライアント・リルートを有効にするには、クライアントの Sysplex ワークロード・バランシングを有効にします。デフォルトでは、Sysplex の自動クライアント・リルートはシームレスであり、転送の際のエラーはアプリケーションに戻されません。

接続コンセントレーター

このサポートは、多数のワークステーションおよび Web ユーザーをサポートするために必要な DB2 for z/OS データベース・サーバー上のリソースを軽減します。接続コンセントレーターを使用すると、データベース・サーバーに並行アクセスする多数のアプリケーションをサポートするために必要なアクティブな並行物理接続が、ごく少数で済みます。クライアントの Sysplex ワークロード・バランシングを有効にすると、接続コンセントレーターも自動的に有効になります。

DB2 for z/OS の Sysplex ワークロード・バランシングでは、データ共用グループが使用されます。クライアント・リルートが可能かどうかは、リクエスターがデータ共用グループと通信するためにセットアップされたアクセス方式で決まります。次の表は、アクセス方式をリストし、データ共用が可能かどうかを示したものです。

表 25. クライアント・リルート・アクセス方式とクライアント・リルート機能

データ共用アクセス方式[1]	説明	クライアント・リルートは可能か?
グループ・アクセス	リクエスターは、グループの動的仮想 IP アドレス (DVIPA) を使用して、 DB2 for z/OS ロケーションへの初期接続を行います。少なくとも 1 つのメンバーが開始されていれば、グループ IP アドレスを使用するデータ共用グループへの接続は常に成功します。グループ IP アドレスは、z/OS Sysplex ディストリビューターを使用して構成されます。 Sysplex 外のクライアントには、DB2 ロケーションを表す単一 IP アドレスを Sysplex ディストリビューターが提供します。フォールト・トレランスの提供に加え、接続ロード・バランシングを提供するように Sysplex ディストリビューターを構成できます。	はい
メンバー固有のアクセス	リクエスターはロケーション別名を使用して、メンバーのうち別名で表されるもの 1 つに初期接続を行います。 DB2 for z/OS は、現在アクティブになっていて作業を実行できるメンバーのリストを戻します。アクティブなメンバーごとに、メンバーの現在の能力を示す重みも戻します。リクエスターはこの情報を使用して、ロケーション別名にも関連付けられた最大能力のメンバー (複数可) に接続します。メンバー固有のアクセスは、データ共用グループのメンバーのサブセット内でワークロード・バランシングとクライアント・リルートをリクエスターが利用する必要がある場合に使用します。	はい
単一メンバー・アクセス	単一メンバー・アクセスは、リクエスターがデータ共用グループの 1 つのメンバーのみにアクセスする必要がある場合に使用します。	いいえ

表 25. クライアント・リルート・アクセス方式とクライアント・リルート機能 (続き)

データ共用アクセス方式[1]	説明	クライアント・リルートは可能か?
クライアント・アフィニティー	DB2 サーバーはこのオプションをサポートしません。すべての再ルーティングがドライバーによって制御されます。データ共用グループ・メンバーに障害が起きたためにフェイルオーバーが行われる場合、障害の起きたメンバーは保持ロックを使用している可能性があります。このロックは、フェイルオーバー先のメンバーでのトランザクションに重大な影響を与えるおそれがあります。他の方式のいずれかがご利用の環境で機能しない場合のみ、クライアント・リルートにクライアント・アフィニティーを使用するようにしてください。詳しくは、カスケード・フェイルオーバー用のクライアント・アフィニティーに対する IBM Data Server Driver for JDBC and SQLJ サポートについてのトピックを参照してください。	はい

注:

1. データ共用アクセス方式について詳しくは、http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.dshare/db2z_tcpipaccessmethods.htmを参照してください。

クライアントでの高可用性に対する Sysplex サポートの構成

DB2 for z/OS に直接接続する IBM Data Server Driver for JDBC and SQLJ クライアント・アプリケーションを、 Sysplex サポートを使用するように構成するには、データ共用グループ (グループ・アクセスの場合) またはデータ共用グループのサブセット (メンバー固有のアクセスの場合) を表すアドレスに接続し、ワークロード・バランシングを使用可能にするプロパティーと最大接続数を設定する必要があります。

次の表に、Java アプリケーションの基本設定値を示します。

表 26. *Java* アプリケーションで *Sysplex* サポートを使用可能にするための基本設定値

データ共用アクセス・メソッド	IBM Data Server Driver for JDBC and SQLJ 設定	値
グループ・アクセス	enableSysplexWLB プロパティー	true
	maxTransportObjects プロパティー	リクエスターがデータ共用グループに対して行える接続の最大数
	接続アドレス:	
	server	グループ IP アドレス、またはデータ共用グループのドメイン・ネーム
	port	SQL ポート番号
	database	インストール中に定義される DB2 ロケーション名

表 26. Java アプリケーションで Sysplex サポートを使用可能にするための基本設定値 (続き)

データ共用アクセス・メソッド	IBM Data Server Driver for JDBC and SQLJ 設定	値
メンバー固有のアクセス	enableSysplexWLB プロパティー	true
	maxTransportObjects プロパティー	リクエスターがデータ共用グループのサブセットに対して行える接続の最大数
	接続アドレス:	
	server	データ共用グループのサブセットを表す DB2 ロケーション別名
	port	SQL ポート番号
	database	初期接続のデータ共用メンバーの DB2 ロケーション名

Sysplex サポートの接続コンセントレーター機能を微調整する場合は、追加プロパティーを使用できます。 IBM Data Server Driver for JDBC and SQLJ のプロパティーを次の表に示します。これらのプロパティーは構成プロパティーであり、Connection および DataSource プロパティーではありません。

表 27. IBM Data Server Driver for JDBC and SQLJ から DB2 for z/OS への直接接続に対する Sysplex サポートを微調整するためのプロパティー

IBM Data Server Driver for JDBC and SQLJ 構成プロパティー	説明
db2.jcc.maxTransportObjectIdleTime	アイドル・トランスポートがドロップされるまでの最大経過時間を秒数で指定します。デフォルトは 600 です。最小サポート値は 0 です。
db2.jcc.maxTransportObjectWaitTime	トランスポートが使用可能になるのをクライアントが待機する秒数を指定します。デフォルトは -1 (無制限) です。最小サポート値は 0 です。
db2.jcc.minTransportObjects	グローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクト数の下限を指定します。デフォルト値は 0 です。0 以下のすべての値は、グローバル・トランスポート・オブジェクト・プールは空になってもかまわないことを意味します。

DB2 for z/OS Sysplex サポートの使用可能化の例 (JDBC または SQLJ アプリケーションの場合)

IBM Data Server Driver for JDBC and SQLJ を WebSphere Application Server および DB2 for z/OS サーバーと一緒に使用するアプリケーションで Sysplex サポートを使用するには、まずこれらの機能を構成する必要があります。

サーバー要件:

- WLM for z/OS

ワークロード・バランシングが効率的に機能するには、DB2 上の作業を分類する必要があります。作業の分類基準にする必要がある領域として、以下の項目が挙げられます。
  – 許可 ID
  – クライアント情報プロパティー
  – ストアード・プロシージャー名

    ストアード・プロシージャー名が分類に使用されるのは、クライアントがトランザクションで最初に発行するステートメントが SQL CALL ステートメントの場合のみです。
- DB2 for z/OS バージョン 8 以降 (データ共用セットアップ)

クライアント要件:
- IBM Data Server Driver for JDBC and SQLJ バージョン 2.7 以降
- WebSphere Application Server バージョン 5.1 以降

IBM Data Server Driver for JDBC and SQLJ の接続コンセントレーター機能とワークロード・バランシング機能を WebSphere Application Server と一緒に使用することを可能にするための手順の例を次に示します。ここで指定する値を推奨値として示す意図はありません。使用可能なトランスポート・オブジェクトの数などの要因に基づいて、値を決定する必要があります。トランスポート・オブジェクトの数は、接続オブジェクトの数以上でなければなりません。

1. 以下のステップを実行することにより、Sysplex ワークロード・バランシングをサポートするために IBM Data Server Driver for JDBC and SQLJ が正しいレベルのものであることを確認してください。
   a. コマンド行ウィンドウで以下のコマンドを実行します。

      ```
      java com.ibm.db2.jcc.DB2Jcc -version
      ```

   b. 出力で以下のような行を見つけて、*nnn* が 2.7 以降になっているかどうかを確認します。

      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```

2. IBM Data Server Driver for JDBC and SQLJ のデータ・ソース・プロパティーを設定し、接続コンセントレーターまたはワークロード・バランシングを使用可能にします。

   WebSphere Application Server 管理コンソールで、アプリケーションからデータ・ソースに接続するために使用する DataSource の以下のプロパティーを設定します。
   - enableSysplexWLB
   - maxTransportObjects

   まずは以下のような設定で始めてみてください。

表 28. DB2 for z/OS における IBM Data Server Driver for JDBC and SQLJ の接続コンセントレーターとワークロード・バランシングのためのデータ・ソース・プロパティーの設定の例

プロパティー	設定
enableSysplexWLB	true
maxTransportObjects	80

3. IBM Data Server Driver for JDBC and SQLJ の構成プロパティーを設定し、このドライバーのもとで作成されるすべての DataSource インスタンスまたは Connection インスタンス用に接続コンセントレーターを微調整します。DB2JccConfiguration.properties ファイルの構成プロパティーを設定する手順は、以下のとおりです。
   a. DB2JccConfiguration.properties ファイルを作成するか、既存の DB2JccConfiguration.properties ファイルを編集します。
   b. 以下の構成プロパティーを設定します。
      - db2.jcc.maxTransportObjects

      まずはこのような設定で始めてみてください。

      ```
      db2.jcc.maxTransportObjects=500
      ```
   c. DB2JccConfiguration.properties のディレクトリー・パスを WebSphere Application Server IBM Data Server Driver for JDBC and SQLJ のクラスパスに追加します。
4. WebSphere Application Server を再始動します。

IBM Data Server Driver for JDBC and SQLJ クライアントから DB2 for z/OS への直接接続に対する Sysplex サポートの操作

DB2 for z/OS への直接接続に対する Sysplex サポートには、Sysplex ワークロード・バランシング機能が含まれます。

クライアントが DB2 for z/OS Sysplex に接続され、Sysplex ワークロード・バランシングが有効になっている場合に行われるステップの概要を以下に示します。

1. クライアントがグループ IP アドレスと呼ばれるシスプレックス全体にわたる IP アドレスを使用して最初の接続を確立したとき、または接続が別の接続オブジェクトによって再利用されるとき、サーバーはメンバーのワークロード分散情報を戻します。

   サーバー・リストがクライアントによってキャッシュに入れられます。サーバー・リストの存続期間は 30 秒です。
2. 新規トランザクションの開始時に、クライアントはキャッシュに入れられたサーバー・リストを読み取って、潜在能力のあるメンバーを識別し、十分に利用されていないそのメンバーに結合されたアイドル・トランスポートを、トランスポート・プールで調べます。 (アイドル・トランスポートとは、接続オブジェクトが関連付けられていないトランスポートのことです。)
   - アイドル・トランスポートが使用可能な場合、クライアントは接続オブジェクトをトランスポートと関連付けます。
   - ユーザー構成可能タイムアウトになってもトランスポート・プール内に使用可能なアイドル・トランスポートがなく、トランスポート・プールがその限度に達しているために新規トランスポートを割り振れない場合は、アプリケーションにエラーが戻されます。
3. トランザクションが実行されると、トランザクションはトランスポートに結合されたメンバーにアクセスします。

4. トランザクションが終了すると、クライアントは接続オブジェクトのトランスポート再利用がまだ許可されるかどうかをサーバーで検査します。
5. トランスポート再利用が許可された場合、サーバーは接続オブジェクトの実行環境に適用される特殊レジスターに対する SET ステートメントのリストを戻します。

   クライアントはこれらのステートメントをキャッシュに入れます。キャッシュに入れられたステートメントは、接続オブジェクトが新規トランスポートと関連付けられるときに、実行環境を再構成するためにクライアントによって再生されます。
6. その後、接続オブジェクトはトランスポートから切り離されます。
7. サーバー・リストのクライアント・コピーは、新規接続が行われるとき、または 30 秒ごとにリフレッシュされます。
8. 新規トランザクションにワークロード・バランシングが必要な場合も、クライアントは同じプロセスを使用して接続オブジェクトをトランスポートと関連付けます。

DB2 for z/OS への直接接続に対する Sysplex サポート対応のアプリケーション・プログラミング

DB2 for z/OS への直接接続に対する Sysplex サポートは、シームレスになるように設計されています。つまり、ほとんどの場合、アプリケーションはこのサポートを取り込むためのコードを追加する必要はありません。

Sysplex サポートがシームレスな自動クライアント・リルートを提供するには、以下の条件を満たす必要があります。

- 接続はトランザクションではありません。つまり、失敗はトランザクション内の最初の SQL ステートメントが実行されたときに発生します。
- すべてのグローバル・セッション・データがクローズされているかドロップされている。
- オープン状態の保留カーソルが存在しない。オープン状態の結果セットも存在しない。

また、アプリケーションが以下の状態にある場合は、シームレス自動クライアント・リルートが成功しない可能性があります。

- 自動コミットが有効になっている。自動コミットが有効になっていると、ステートメントの実行とコミットが複数回行われる可能性があります。
- コンパウンド SQL などの特定のタイプの SQL 操作を実行中である、または配列入力を処理中である。

これらの条件のいずれかを満たしていない場合にサーバーとの接続が再確立されると、アプリケーションに SQLCODE -30108 (SQL30108N) が戻され、現行トランザクション内で行われた作業はすべてロールバックされます。このアプリケーションで行う必要のある処理は次のとおりです。

- -30108 エラーと一緒に戻された理由コードを検査し、障害の起きているデータ共用メンバーの特殊レジスターの設定が新しい (フェイルオーバー) データ共用メンバーに引き継がれているかどうかを判別します。現行のものではない特殊レジスター値はすべて再設定します。
- 直前の COMMIT 操作以降に行われたすべての SQL 操作を実行します。

IBM Data Server Driver for JDBC and SQLJ でのシームレス・フェイルオーバー用のクライアント・アフィニティー・サポート

シームレス・フェイルオーバー用のクライアント・アフィニティーが、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続に関してサポートされます。

特定のアプリケーションに関して特定のフェイルオーバー順序を適用したい場合には、シームレス・フェイルオーバーのクライアント・アフィニティーのサポートを使用可能にすることができます。

IBM Informix Dynamic Server 環境では、1 次サーバーとスタンバイ・サーバーは、接続マネージャーによって制御される高可用性クラスターのメンバーに対応します。複数の接続マネージャーが存在する場合、clientRerouteAlternateServerName および clientRerouteAlternatePortNumber プロパティーが設定されていれば、 IBM Data Server Driver for JDBC and SQLJ はそれら複数の接続マネージャーを使用して 1 次サーバーおよび代替サーバーの情報を確認することができます。シームレス・フェイルオーバーのクライアント・アフィニティーのサポートを使用可能にしていない場合、 IBM Data Server Driver for JDBC and SQLJは、最初の接続についてのみ代替接続マネージャーを使用します。

シームレス・フェイルオーバーのクライアント・アフィニティーのサポートは、enableClientAffinitiesList プロパティーを設定することによって使用可能にします。そのプロパティーは、クライアント・アフィニティー・リストを指定します。このリストは、障害後の接続の試行順序を指定します。

シームレス・フェイルオーバーのクライアント・アフィニティーが使用可能になっている場合、 IBM Data Server Driver for JDBC and SQLJ は、clientRerouteAlternateServerName および clientRerouteAlternatePortNumber プロパティーによって識別されるサーバーに対してのみ接続を再試行します。clientRerouteAlternateServerName リストには 1 次サーバーも含まれます。

例

データベースへの接続に、以下のプロパティーが設定されているものと想定します。

プロパティー	値
enableClientAffinitiesList	DB2BaseDataSource.YES (1)
clientRerouteAlternateServername	host1,host2,host3
clientRerouteAlternatePortNumber	port1,port2,port3
maxRetriesForClientReroute	3
retryIntervalForClientReroute	2

host1:port1 によって識別されるサーバーへの接続中に、通信障害が発生したとします。以下のステップは、シームレス・フェイルオーバーが有効な場合のクライアント・リルート・シナリオを示しています。

1. ドライバーは host1:port1 への接続を試みます。
2. host1:port1 への接続は失敗します。
3. ドライバーは 2 秒間待機します。
4. ドライバーは host1:port1 への接続を試みます。
5. host1:port1 への接続は失敗します。
6. ドライバーは 2 秒間待機します。
7. ドライバーは host1:port1 への接続を試みます。
8. host1:port1 への接続は失敗します。
9. ドライバーは 2 秒間待機します。
10. ドライバーは host2:port2 への接続を試みます。
11. host2:port2 への接続は失敗します。
12. ドライバーは 2 秒間待機します。
13. ドライバーは host2:port2 への接続を試みます。
14. host2:port2 への接続は失敗します。
15. ドライバーは 2 秒間待機します。
16. ドライバーは host2:port2 への接続を試みます。
17. host2:port2 への接続は失敗します。
18. ドライバーは 2 秒間待機します。
19. ドライバーは host3:port3 への接続を試みます。
20. host3:port3 への接続は失敗します。
21. ドライバーは 2 秒間待機します。
22. ドライバーは host3:port3 への接続を試みます。
23. host3:port3 への接続は失敗します。
24. ドライバーは 2 秒間待機します。
25. ドライバーは host3:port3 への接続を試みます。
26. host3:port3 への接続は失敗します。
27. ドライバーは 2 秒間待機します。
28. ドライバーはエラー・コード -4499 の SQLException をスローします。

IDS サーバーの JDBC 接続コンセントレーターおよびワークロード・バランシング

IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用して IBM Informix Dynamic Server にアクセスする Java アプリケーションでは、接続コンセントレーターとワークロード・バランシング機能を利用することができます。

IBM Data Server Driver for JDBC and SQLJ の接続コンセントレーター機能により、 IBM Informix Dynamic Server データ・ソースによる多数のクライアント・アプリケーションのサポートに必要なリソースを削減することができます。 IBM Data

Server Driver for JDBC and SQLJ の接続コンセントレーター機能によって、多数の接続オブジェクトが同じ物理接続を使用することができ、これによりデータ・ソースへの物理接続の総数が減少します。

IBM Informix Dynamic Serverのワークロード・バランシングの利点: IBM Data Server Driver for JDBC and SQLJのワークロード・バランシングにより、高可用性クラスターの可用性を高めることができます。ワークロード・バランシングが使用可能になっている場合は、ドライバーによりクラスター内のサーバーについての状況情報が頻繁に取得されます。ドライバーによりこの情報が使用され、次のトランザクションがどのサーバーに転送されるべきかが判断されます。ワークロード・バランシングを使用すれば、接続マネージャーにより、作業がクラスターの各サーバー間で効率よく分散され、またあるサーバーに障害が発生した場合は、その作業が別のサーバーに転送されます。

IBM Data Server Driver for JDBC and SQLJ はトランスポート・オブジェクト およびグローバル・トランスポート・オブジェクト・プール を使用して、接続コンセントレーターおよびワークロード・バランシングをサポートします。データ・ソースへの各物理接続ごとに、トランスポート・オブジェクトが 1 つあります。接続コンセントレーターとワークロード・バランシングを使用可能にした場合は、トランスポート・オブジェクトの最大数を設定することによって、任意の時点でデータ・ソースへの物理接続の最大数を設定します。

ドライバー・レベルでは、 IBM Data Server Driver for JDBC and SQLJ 構成プロパティーを使用して、トランスポート・オブジェクトの数に対する限度を設定します。

接続レベルでは、DataSource プロパティーを使用して、 IBM Data Server Driver for JDBC and SQLJ の接続コンセントレーターとワークロード・バランシングを使用可能および使用不可にし、トランスポート・オブジェクトの数に対する限度を設定します。これらのプロパティーは、DataSource インターフェースまたは DriverManager インターフェースを使用して接続を取得する際に設定することができます。

IDS サーバー用の IBM Data Server Driver for JDBC and SQLJ ワークロード・バランシングを有効にする例

IBM Data Server Driver for JDBC and SQLJ のワークロード・バランシング機能を WebSphere Application Server および IDS サーバーと一緒に使用するには、まずそれらの機能を構成する必要があります。

サーバー要件:

- 少なくとも 1 つの接続マネージャー構成

  接続マネージャーの 1 つの構成ごとに、以下のものが必要です。
  - IBM Informix Dynamic Server バージョン 11.50 以降
  - IBM Informix Connect または IBM Informix Client Software Development Kit (Client SDK)、バージョン 3.50 以降
  - 1 つの 1 次サーバーと 1 つ以上の 2 次サーバー (タイプは HDR 2 次、SD 2 次、RS 2 次のいずれか)

クライアント要件:
- IBM Data Server Driver for JDBC and SQLJ バージョン 3.52 以降
- WebSphere Application Server バージョン 5.1 以降

IBM Data Server Driver for JDBC and SQLJ のワークロード・バランシング機能を WebSphere Application Server と一緒に使用することを可能にするための手順の例を次に示します。ここで指定する値を推奨値として示す意図はありません。以下のような要因に基づいて値を判別する必要があります。
- システム・リソースの可用性
- 使用可能な物理接続の数
- 接続オブジェクトと転送オブジェクトの望ましい比率

1. 以下のステップを実行することにより、ワークロード・バランシングをサポートするために IBM Data Server Driver for JDBC and SQLJ が正しいレベルのものであることを確認してください。
   a. コマンド行ウィンドウで以下のコマンドを実行します。
      ```
      java com.ibm.db2.jcc.DB2Jcc -version
      ```
   b. 出力で以下のような行を見つけて、*nnn* が 3.52 以降になっているかどうかを確認します。
      ```
      [jcc] Driver: IBM Data Server Driver for JDBC and SQLJ Architecture nnn xxx
      ```
2. ワークロード・バランシングが有効になるよう、 IBM Data Server Driver for JDBC and SQLJ のプロパティーを設定します。以下のようにプロパティーを設定します。
   - このドライバーの下で作成される DataSource または Connection のすべてのインスタンスで、ワークロード・バランシングの同じ動作が有効になるように、 IBM Data Server Driver for JDBC and SQLJ の構成プロパティーを設定します。そのためには、次のようにします。
     a. DB2JccConfiguration.properties ファイルを作成するか、既存の DB2JccConfiguration.properties ファイルを編集します。
     b. 以下の構成プロパティーを設定します。
        - db2.jcc.minTransportObjects
        - db2.jcc.maxTransportObjects
        - db2.jcc.maxTransportObjectWaitTime
        - db2.jcc.dumpPool
        - db2.jcc.dumpPoolStatisticsOnSchedule
        - db2.jcc.dumpPoolStatisticsOnScheduleFile

        まずは以下のような設定で始めてみてください。
        ```
        db2.jcc.minTransportObjects=0
        db2.jcc.maxTransportObjects=1500
        db2.jcc.maxTransportObjectWaitTime=-1
        db2.jcc.dumpPool=0
        db2.jcc.dumpPoolStatisticsOnSchedule=60
        db2.jcc.dumpPoolStatisticsOnScheduleFile=/home/WAS/logs/srv1/poolstats
        ```
     c. DB2JccConfiguration.properties のディレクトリー・パスを WebSphere Application Server IBM Data Server Driver for JDBC and SQLJ のクラスパスに追加します。
   - DataSource または Connection のすべてのインスタンスでワークロード・バランシングの動作が同じになるようにはしない場合は、Connection または

DataSource の各インスタンスについて、個別にワークロード・バランシングのプロパティーを設定することができます。

3. IBM Data Server Driver for JDBC and SQLJ のデータ・ソースのプロパティーを設定し、ワークロード・バランシングが有効になるようにします。

   WebSphere Application Server 管理コンソールで、アプリケーションからデータ・ソースに接続するために使用する DataSource の以下のプロパティーを設定します。

表 29. IBM Data Server Driver for JDBC and SQLJ における IBM Informix Dynamic Server の接続コンセントレーターとワークロード・バランシングのためのデータ・ソース・プロパティーの設定の例

プロパティー	設定
enableSysplexWLB	true
maxTransportObjects	100

4. WebSphere Application Server を再始動します。

JDBC アプリケーションでのデータ・ソースからの切断

データ・ソースへの接続を使用した作業が完了したら、データ・ソースへの接続を閉じることが必要 です。そうすると、Connection オブジェクトのデータベースと JDBC リソースがすぐに解放されます。

データ・ソースへの接続を閉じるには、close メソッドを使用します。以下に例を示します。

```
 Connection con;
 ...
con.close();
```

DB2 データ・ソースへの接続で、自動コミット・モードがオンになっていない場合に接続を閉じるには、接続が作業単位の境界に達している必要があります。

IBM Informix Dynamic Server データベースへの接続で、そのデータベースがロギングをサポートしており、自動コミット・モードがオンになっていない場合に接続を閉じるには、接続が作業単位の境界に達している必要があります。

第 4 章 SQLJ アプリケーション・プログラミング

SQLJ アプリケーションの作成は、他の言語での SQL アプリケーションの作成と多くの点で共通しています。

一般に、以下のことを行う必要があります。
- SQLJ と JDBC のメソッドが含まれている Java パッケージをインポートする。
- DB2 表へのデータの送信用またはこの表からのデータの取得用の変数を宣言する。
- データ・ソースに接続する。
- SQL ステートメントを実行する。
- SQL エラーおよび警告を処理する。
- データ・ソースへの接続を切断する。

実行する必要のあるタスクは他の言語のものと同様ですが、これらのタスクを実行する方法およびその順序は多少異なります。

簡単な SQLJ アプリケーションの例

この簡単な SQLJ アプリケーションでは、JDBC アプリケーションに含める必要がある基本要素を示しています。

図 27. 簡単な SQLJ アプリケーション

```
import sqlj.runtime.*;                                          1
import java.sql.*;

#sql context EzSqljCtx;                                         3a
#sql iterator EzSqljNameIter (String LASTNAME);                 4a

public class EzSqlj {
  public static void main(String args[])
    throws SQLException
  {
    EzSqljCtx ctx = null;
    String URLprefix = "jdbc:db2:";
    String url;
    url = new String(URLprefix + args[0]);
                                          // Location name is an input parameter
    String hvmgr="000010";                                      2
    String hvdeptno="A00";
    try {                                                       3b
      Class.forName("com.ibm.db2.jcc.DB2Driver");
    } catch (Exception e)
    {
       throw new SQLException("Error in EzSqlj: Could not load the driver");
    }
    try
    {
       System.out.println("About to connect using url: " + url);
       Connection con0 = DriverManager.getConnection(url);      3c
                                          // Create a JDBC Connection
       con0.setAutoCommit(false);         // set autocommit OFF
       ctx = new EzSqljCtx(con0);                               3d
```

```
      try
      {
        EzSqljNameIter iter;
        int count=0;

        #sql [ctx] iter =
          {SELECT LASTNAME FROM EMPLOYEE};                          4b
                                          // Create result table of the SELECT
        while (iter.next()) {                                       4c
           System.out.println(iter.LASTNAME());
                                          // Retrieve rows from result table
           count++;
        }
        System.out.println("Retrieved " + count + " rows of data");
        iter.close();                     // Close the iterator
      }
      catch( SQLException e )                                       5
      {
         System.out.println ("**** SELECT SQLException...");
         while(e!=null) {
           System.out.println ("Error msg: " + e.getMessage());
           System.out.println ("SQLSTATE: " + e.getSQLState());
           System.out.println ("Error code: " + e.getErrorCode());
           e = e.getNextException(); // Check for chained exceptions
         }
      }
      catch( Exception e )
      {
         System.out.println("**** NON-SQL exception   = " + e);
         e.printStackTrace();
      }
      try
      {
        #sql [ctx]                                                  4d
          {UPDATE DEPARTMENT SET MGRNO=:hvmgr
             WHERE DEPTNO=:hvdeptno};   // Update data for one department
                                                                    6
        #sql [ctx] {COMMIT};            // Commit the update
      }
      catch( SQLException e )
      {
         System.out.println ("**** UPDATE SQLException...");
         System.out.println ("Error msg: " + e.getMessage() + ". SQLSTATE=" +
           e.getSQLState() + " Error code=" + e.getErrorCode());
         e.printStackTrace();
      }
      catch( Exception e )
      {
         System.out.println("**** NON-SQL exception   = " + e);
         e.printStackTrace();
      }
      ctx.close();                                                  7
}
catch(SQLException e)
{
   System.out.println ("**** SQLException ...");
   System.out.println ("Error msg: " + e.getMessage() + ". SQLSTATE=" +
     e.getSQLState() + " Error code=" + e.getErrorCode());
   e.printStackTrace();
}
catch(Exception e)
{
  System.out.println ("**** NON-SQL exception = " + e);
```

```
    e.printStackTrace();
  }

}
```

149 ページの図 27 の注:

注	説明
1	これらのステートメントにより、JDBC コア API が含まれた java.sql パッケージ、および SQLJ API が含まれた sqlj.runtime パッケージがインポートされます。アクセスが必要になる可能性のある他のパッケージまたはクラスについては、『SQLJ サポート用の Java パッケージ』を参照してください。
2	String 変数の hvmgr と hvdeptno は、ホスト *ID* です。これらは、 DB2 のホスト変数に相当します。詳細については、『SQLJ アプリケーションでの変数』を参照してください。
3a、3b、3c、および 3d	これらのステートメントでは、使用可能な 3 つの技法のいずれかを使用して、データ・ソースに接続する方法が示されています。詳細については、『SQLJ を使用したデータ・ソースへの接続』を参照してください。 手順 3b (JDBC ドライバーのロード) は、JDBC 4.0 を使用する場合は不要です。
4a、4b、4c、および 4d	これらのステートメントでは、SQLJ で SQL ステートメントを実行する方法が示されています。ステートメント 4a では、SQLJ での SQL カーソルの宣言に相当するものが示されています。ステートメント 4b と 4c では、SQL OPEN CURSOR および SQL FETCH の実行に相当することを SQLJ で行う 1 つの方法が示されています。ステートメント 4d では、SQL UPDATE の実行に相当することを SQLJ で行う方法が示されています。詳細については、『SQLJ アプリケーションにおける SQL ステートメント』を参照してください。
5	この try/catch ブロックでは、SQL エラー処理用の SQLException クラスの使用が示されています。 SQL エラーの処理の詳細については、『SQLJ アプリケーションでの SQL エラーの処理』を参照してください。 SQL 警告の処理の詳細については、『SQLJ アプリケーションでの SQL 警告の処理』を参照してください。
6	これはコメントの例です。SQLJ プログラムへのコメントの組み込みに関する規則については、『SQLJ アプリケーションでのコメント』を参照してください。
7	このステートメントにより、データ・ソースへの接続がクローズされます。『SQLJ アプリケーションでのデータ・ソースへの接続のクローズ』を参照してください。

SQLJ を使用したデータ・ソースへの接続

SQLJ アプリケーションの場合も、他の DB2 アプリケーションの場合と同じく、SQL ステートメントを実行する前にデータ・ソースに接続する必要があります。

SQLJ プログラムでデータ・ソースに接続するには、6 つの技法のうちのいずれか 1 つを使用します。そのうち、JDBC の DriverManager インターフェースを使用する技法が 2 つ、JDBC の DataSource インターフェースを使用する技法が 2 つ、既に作成されている接続コンテキストを使用する技法が 1 つ、デフォルトの接続を使用する技法が 1 つあります。

SQLJ の接続技法 1: JDBC DriverManager インターフェース

SQLJ の接続技法 1 では、接続を作成するための基礎的な手段として JDBC の DriverManager インターフェースを使用します。

SQLJ の接続技法 1 を使用するには、以下の手順を実行します。

1. SQLJ の*接続宣言節* を実行します。

   そうすると、*接続*コンテキスト・クラス が生成されます。最も簡単な形式の接続宣言節は、以下のとおりです。

   ```
   #sql context context-class-name;
   ```

   生成される接続コンテキスト・クラスの名前は、*context-class-name* になります。
2. Class.forName メソッドを呼び出して、JDBC ドライバーをロードします。
   - IBM Data Server Driver for JDBC and SQLJ の場合は、以下のようにして Class.forName を呼び出します。

     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```

     JDBC 4.0 ドライバーを使用している場合は、この手順は不要です。
   - Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーの場合 (非推奨) は、以下のようにして Class.forName を呼び出します。

     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. 手順 1 で作成した接続コンテキスト・クラスのコンストラクターを呼び出します。

   そうすると、関連するデータ・ソースで実行する各 SQL ステートメントに指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下のいずれかの形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class(String url, boolean autocommit);

   connection-context-class connection-context-object=
     new connection-context-class(String url, String user,
       String password, boolean autocommit);
   connection-context-class connection-context-object=
     new connection-context-class(String url, Properties info,
       boolean autocommit);
   ```

   各パラメーターの意味は、以下のとおりです。

   url データ・ソースに関連したロケーション名を指定するストリング。その引数は、『DriverManager インターフェースと IBM Data Server Driver for JDBC and SQLJ を使用したデータ・ソースへの接続』で指定されているいずれかの形式になります。形式は、使用する JDBC ドライバーによって異なります。

   user と *password*
   接続先のデータ・ソースで接続用のユーザー ID とパスワードが必要であれば、そのユーザー ID とパスワードを指定します。

   info
   接続用の一連のドライバー・プロパティーを含んだタイプ java.util.Properties のオブジェクトを指定します。Linux、UNIX、Windows 用の DB2 JDBC

Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の場合は、user プロパティーと password プロパティーだけを指定します。 IBM Data Server Driver for JDBC and SQLJ の場合は、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』でリストされているいずれかのプロパティーを指定できます。

autocommit
各ステートメントの後でデータベース・マネージャーから COMMIT を実行するかどうかを指定します。可能な値は true または false です。 false を指定した場合は、明示的なコミット操作を実行する必要があります。

接続技法 1 を使用してロケーション NEWYORK への接続を作成するためのコードを以下に示します。この接続ではユーザー ID とパスワードが必要ですが、自動コミットは必要ありません。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql context Ctx;             // Create connection context class Ctx        1
String userid="dbadm";        // Declare variables for user ID and password
String password="dbadm";
String empname;               // Declare a host variable
...
try {                         // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                              2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Ctx myConnCtx=                                                             3
  new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
  userid,password,false);     // Create connection context object myConnCtx
                              // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                              // Use myConnCtx for executing an SQL statement
```

図 28. 接続技法 1 によるデータ・ソース接続

SQLJ の接続技法 2: JDBC DriverManager インターフェース

SQLJ の接続技法 2 では、接続を作成するための基礎的な手段として JDBC の DriverManager インターフェースを使用します。

SQLJ の接続技法 2 を使用するには、以下の手順を実行します。

1. SQLJ の*接続宣言節* を実行します。

   そうすると、*接続コンテキスト・クラス* が生成されます。最も簡単な形式の接続宣言節は、以下のとおりです。

   ```
   #sql context context-class-name;
   ```

   生成される接続コンテキスト・クラスの名前は、*context-class-name* になります。
2. Class.forName メソッドを呼び出して、JDBC ドライバーをロードします。
   - IBM Data Server Driver for JDBC and SQLJ の場合は、以下のようにして Class.forName を呼び出します。

```
Class.forName("com.ibm.db2.jcc.DB2Driver");
```

JDBC 4.0 ドライバーを使用している場合は、この手順は不要です。

- Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーの場合 (非推奨) は、以下のようにして Class.forName を呼び出します。

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
```

3. JDBC の DriverManager.getConnection メソッドを呼び出します。

   そうすると、データ・ソースに接続するための JDBC 接続オブジェクトが作成されます。『DriverManager インターフェースと IBM Data Server Driver for JDBC and SQLJ を使用したデータ・ソースへの接続』で指定されているいずれかの形式の getConnection を使用できます。

   パラメーター *url*、*user*、*password* の意味は、以下のとおりです。

   url データ・ソースに関連したロケーション名を指定するストリング。その引数は、『DriverManager インターフェースと IBM Data Server Driver for JDBC and SQLJ を使用したデータ・ソースへの接続』で指定されているいずれかの形式になります。形式は、使用する JDBC ドライバーによって異なります。

   user と *password*
   接続先のデータ・ソースで接続用のユーザー ID とパスワードが必要であれば、そのユーザー ID とパスワードを指定します。

4. 手順 1 (153 ページ) で作成した接続コンテキスト・クラスのコンストラクターを呼び出します。

   そうすると、関連するデータ・ソースで実行する各 SQL ステートメントに指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下の形式にする必要があります。

```
connection-context-class connection-context-object=
  new connection-context-class(Connection JDBC-connection-object);
```

   JDBC-connection-object パラメーターは、手順 3 で作成した Connection オブジェクトです。

接続技法 2 を使用してロケーション NEWYORK への接続を作成するためのコードを以下に示します。この接続ではユーザー ID とパスワードが必要ですが、自動コミットは必要ありません。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql context Ctx;           // Create connection context class Ctx       1
String userid="dbadm";      // Declare variables for user ID and password
String password="dbadm";
String empname;             // Declare a host variable
...
try {                       // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                           2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=                                                     3
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                            // Create JDBC connection object jdbccon
jdbccon.setAutoCommit(false); // Do not autocommit
Ctx myConnCtx=new Ctx(jdbccon);                                         4
                            // Create connection context object myConnCtx
                            // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

図 29. 接続技法 2 によるデータ・ソース接続

SQLJ の接続技法 3: JDBC DataSource インターフェース

SQLJ の接続技法 3 では、接続を作成するための基礎的な手段として JDBC の DataSource を使用します。

SQLJ の接続技法 3 を使用するには、以下の手順を実行します。

1. SQLJ の*接続宣言節* を実行します。

   そうすると、*接続コンテキスト・クラス* が生成されます。最も簡単な形式の接続宣言節は、以下のとおりです。

   ```
   #sql context context-class-name;
   ```

   生成される接続コンテキスト・クラスの名前は、*context-class-name* になります。
2. システム管理者が DataSource オブジェクトを別のプログラムで作成した場合は、以下の手順を実行します。そうでない場合は、DataSource オブジェクトを作成して、そのオブジェクトにプロパティーを割り当てます。
   a. 接続する必要があるデータ・ソースの論理名を取得します。
   b. 次の手順で使用するコンテキストを作成します。
   c. アプリケーション・プログラムで Java Naming and Directory Interface (JNDI) を使用して、論理データ・ソース名に関連した DataSource オブジェクトを取得します。
3. JDBC の DataSource.getConnection メソッドを呼び出します。

   そうすると、データ・ソースに接続するための JDBC 接続オブジェクトが作成されます。以下のいずれかの形式の getConnection を使用できます。

   ```
   getConnection();
   getConnection(user, password);
   ```

   パラメーター *user*、*password* の意味は、以下のとおりです。

user と *password*
接続先のデータ・ソースで接続用のユーザー ID とパスワードが必要であれば、そのユーザー ID とパスワードを指定します。

4. デフォルトの自動コミット・モードが適切でない場合は、JDBC の Connection.setAutoCommit メソッドを呼び出します。

   そうすることによって、各ステートメントの後でデータベース・マネージャーから COMMIT を実行するかどうかを指定できます。そのメソッドの形式は、以下のとおりです。

   ```
   setAutoCommit(boolean autocommit);
   ```

5. 手順 1 (155 ページ) で作成した接続コンテキスト・クラスのコンストラクターを呼び出します。

   そうすると、関連するデータ・ソースで実行する各 SQL ステートメントに指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下の形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class(Connection JDBC-connection-object);
   ```

   JDBC-connection-object パラメーターは、手順 3 (155 ページ) で作成した Connection オブジェクトです。

接続技法 3 を使用して論理名 jdbc/sampledb のロケーションへの接続を作成するコードを以下に示します。この例では、システム管理者が JNDI 検索で利用できる DataSource オブジェクトを作成してデプロイしたことを想定しています。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
#sql context CtxSqlj;       // Create connection context class CtxSqlj   1
Context ctx=new InitialContext();                                         2b
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");                    2c
Connection con=ds.getConnection();                                        3
String empname;             // Declare a host variable
...
con.setAutoCommit(false);   // Do not autocommit                          4
CtxSqlj myConnCtx=new CtxSqlj(con);                                       5
                            // Create connection context object myConnCtx
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

図 30. 接続技法 3 によるデータ・ソース接続

SQLJ の接続技法 4: JDBC DataSource インターフェース

SQLJ の接続技法 4 では、接続を作成するための基礎的な手段として JDBC の DataSource を使用します。この技法では、DataSource を JNDI に登録することが**必要**です。

SQLJ の接続技法 4 を使用するには、以下の手順を実行します。

1. システム管理者から、接続する必要があるデータ・ソースの論理名を取得します。
2. SQLJ の接続宣言節を実行します。

   このタイプの接続では、接続宣言節を以下の形式にする必要があります。

   ```
   #sql public static context context-class-name
    with (dataSource="logical-name");
   ```

   接続コンテキストは、public、static として宣言する必要があります。
   logical-name は、手順 1 で取得したデータ・ソース名です。
3. 手順 2 で作成した接続コンテキスト・クラスのコンストラクターを呼び出します。

   そうすると、関連するデータ・ソースで実行する各 SQL ステートメントに指定する接続コンテキスト・オブジェクトが作成されます。コンストラクター呼び出しステートメントは、以下のいずれかの形式にする必要があります。

   ```
   connection-context-class connection-context-object=
     new connection-context-class();

   connection-context-class connection-context-object=
     new connection-context-class (String user,
       String password);
   ```

   パラメーター *user*、*password* の意味は、以下のとおりです。

   user と *password*
   : 接続先のデータ・ソースで接続用のユーザー ID とパスワードが必要であれば、そのユーザー ID とパスワードを指定します。

接続技法 4 を使用して論理名 jdbc/sampledb のロケーションへの接続を作成するコードを以下に示します。この接続では、ユーザー ID とパスワードが必要です。

```
#sql public static context Ctx
  with (dataSource="jdbc/sampledb");                                    2
                              // Create connection context class Ctx
String userid="dbadm";        // Declare variables for user ID and password
String password="dbadm";

String empname;               // Declare a host variable
...
Ctx myConnCtx=new Ctx(userid, password);                                3
                              // Create connection context object myConnCtx
                              // for the connection to jdbc/sampledb
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                              // Use myConnCtx for executing an SQL statement
```

図 31. 接続技法 4 によるデータ・ソース接続

SQLJ の接続技法 5: 既に作成されている接続コンテキストの使用

SQLJ 接続技法 5 では、既に作成されている接続コンテキストを使用してデータ・ソースに接続します。

通常は、1 つのプログラムで接続コンテキスト・クラスを宣言し、接続コンテキストを作成し、その接続コンテキストをパラメーターとして他のプログラムに渡すのが一般的です。接続コンテキストを使用するプログラムは、渡された接続コンテキスト・オブジェクトを引数としてコンストラクターを呼び出します。

プログラム CtxGen.sqlj は、接続コンテキスト Ctx を宣言し、インスタンス oldCtx を作成します。

```
#sql context Ctx;
...
// Create connection context object oldCtx
```

プログラム test.sqlj は、oldCtx をパラメーターとして受け取り、その oldCtx を接続コンテキスト・コンストラクターの引数として使用します。

```
void useContext(sqlj.runtime.ConnectionContext oldCtx)
                                    // oldCtx was created in CtxGen.sqlj
{
  Ctx myConnCtx=
    new Ctx(oldCtx);                // Create connection context object myConnCtx
                                    // from oldCtx
  #sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
     WHERE EMPNO='000010'};
                                    // Use myConnCtx for executing an SQL statement
...
}
```

SQLJ の接続技法 6: デフォルトの接続の使用

SQLJ の接続技法 6 では、デフォルトの接続を使用してデータ・ソースに接続します。この技法を使用するのは、データベース・スレッドを別のリソース・マネージャー (Java ストアード・プロシージャー環境など) によって制御する場合に限ってください。

デフォルトの接続を使用する場合は、接続コンテキスト・オブジェクトなしで SQL ステートメントを指定します。この技法を使用すると、プログラムで JDBC インターフェースを明示的に使用する場合以外は、JDBC ドライバーをロードする必要がありません。

デフォルトの接続コンテキストは、以下のいずれかになります。

- 論理名 jdbc/defaultDataSource にバインドされているデータ・ソースに関連した接続コンテキスト。
- ConnectionContext.setDefaultContext メソッドでデフォルトの接続コンテキストとして設定されている、明示的に作成された接続コンテキスト。この方法でデフォルトの接続コンテキストを作成することは、お勧めできません。

以下の SQLJ 実行節には接続コンテキストがないので、デフォルトの接続コンテキストが使用されます。

```
#sql {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};  // Use default connection for
                           // executing an SQL statement
```

SQLJ サポート用の Java パッケージ

SQLJ ステートメントを実行するか、または SQLJ プログラムで JDBC メソッドを呼び出す場合は、事前にそれらのステートメントをサポートする各種 Java パッケージのすべてまたは一部にアクセスできるようにしておく必要があります。

これは、パッケージまたは特定のクラスをインポートするか、または完全修飾クラス名を使用して実行できます。 SQLJ プログラムには、以下のパッケージまたはクラスが必要になる場合があります。

sqlj.runtime
SQLJ ランタイム API が含まれています。

java.sql
コア JDBC API が含まれます。

com.ibm.db2.jcc
JDBC および SQLJ のドライバー固有のインプリメンテーションが含まれています。

javax.naming
Java Naming and Directory Interface (JNDI) 検索の実行用のメソッドが含まれています。

javax.sql
DataSource オブジェクトの作成用のメソッドが含まれています。

SQLJ アプリケーションでの変数

他の言語の DB2 プログラムでは、ホスト変数を使用してアプリケーション・プログラムと DB2 の間のデータの受け渡しを行います。 SQLJ プログラムでは、ホスト変数またはホスト式 を使用できます。

ホスト式はコロン (:) で始まります。コロンの後にオプション・パラメーター・モード ID (IN、OUT、または INOUT) が続き、その後に括弧で囲んだ式の節が続きます。

ホスト変数およびホスト式には大文字と小文字の区別があります。

複合式とは、単一値に評価する配列エレメントまたは Java 式です。 SQLJ 節での複合式は括弧で囲む必要があります。

次の例は、ホスト式の使用法を表したものです。

例: Java ID の宣言および SELECT ステートメントでの Java ID の使用

この例では、`#sql` で始まるステートメントには他の言語の SELECT ステートメントと同じ機能があります。このステートメントは、従業員番号 000010 の従業員の姓を Java ID `empname` に割り当てます。

```
String empname;
...
#sql [ctxt]
  {SELECT LASTNAME INTO :empname FROM EMPLOYEE WHERE EMPNO='000010'};
```

例: Java ID の宣言およびストアード・プロシージャー呼び出しでの Java ID の使用

この例では、#sql で始まるステートメントには他の言語の SQL CALL ステートメントと同じ機能があります。このステートメントは Java ID empno をストアード・プロシージャー A への入力パラメーターとして使用します。empno の前のキーワード IN は、empno が入力パラメーターであることを示します。 CALL ステートメントでのパラメーターの場合、IN がデフォルトです。パラメーターが使用される方法を示す明示的またはデフォルトの修飾子 (IN、OUT、または INOUT) は、ストアード・プロシージャーの CREATE PROCEDURE ステートメントで指定したパラメーター定義の対応する値と一致しなければなりません。

```
String empno = "0000010";
...
#sql [ctxt] {CALL A (:IN empno)};
```

例: ホスト ID としての複合式の使用

この例は、複合式 (((int)yearsEmployed++/5)*500) をホスト式として使用します。

```
#sql [ctxt] {UPDATE EMPLOYEE
        SET BONUS=:(((int)yearsEmployed++/5)*500) WHERE EMPNO=:empID};
```

SQLJ は複合ホスト式を処理する際に以下のアクションを実行します。

- ステートメント内の各ホスト式を、それぞれの値をデータベースに割り当てる前に左から右に評価します。
- 副次作用 (postfix 演算子を指定した演算など) を、通常の Java 規則に従って評価します。すべてのホスト式の値が完全に評価されてから、それらの値が DB2 に渡されます。
- 丸めおよび切り捨てに Java 規則を使用します。

従って、UPDATE ステートメントが実行される前に yearsEmployed の値が 6 である場合、UPDATE ステートメントによって列 BONUS に割り当てられる値は ((int)6/5)*500、または 500 です。 500 が BONUS に割り当てられた後、yearsEmployed の値が増やされます。

変数名に関する制限: SQLJ プログラムでは 2 つのストリングに特殊な意味があります。それらのストリングを SQLJ プログラムで使用する場合は以下の制限を守ってください。

- ストリング __sJT_ は SQLJ によって生成される変数名のために予約された接頭部です。以下のタイプの名前を __sJT_ で始めないでください。
  - ホスト式の名前
  - 実行可能 SQL ステートメントを含むブロックで宣言されている Java 変数名
  - 実行可能 SQL ステートメントを含むメソッドのパラメーターの名前
  - 実行可能 SQL ステートメントを含むクラス内のフィールドの名前、あるいは実行可能 SQL ステートメントを含むサブクラスまたは囲みクラスを持つクラス内のフィールドの名前
- ストリング _SJ は SQLJ によって生成されるリソース・ファイルおよびクラスのために予約された接尾部です。クラス名および入力ソース・ファイル名ではストリング _SJ を使用しないでください。

SQLJ アプリケーションでのコメント

SQLJ プログラムで説明を記述するには、コメントを組み込む必要があります。これを行うには、Java のコメントを使用します。Java でのコメントは、/* */ または // で示します。

Java のコメントは、SQLJ 節の外側に組み込みますが、Java 言語で許可される場所であればどこにでも組み込むことができます。 SQLJ 節内で、Java のコメントを使用できるのは、以下の場所です。

- ホスト式 (/* */ または //) 内。
- データ・ソースで SQL ステートメント内のコメント (/* */ または --) がサポートされる場合は、実行可能節中の SQL ステートメント内。

  SQL ステートメント内の /* および */ の組みは、ネストすることができます。

SQLJ アプリケーションでの SQL ステートメントの実行

従来の SQL プログラムでは SQL ステートメントを実行して、表の作成、表のデータの更新、表からのデータの取得、ストアード・プロシージャーの呼び出し、トランザクションのコミットまたはロールバックを行います。 SQLJ プログラムでは、さらにこれらのステートメントを SQLJ *実行可能節* 内で実行します。

実行可能節は、以下のいずれかの一般形式にすることができます。

```
#sql [connection-context] {sql-statement};
#sql [connection-context,execution-context] {sql-statement};
#sql [execution-context] {sql-statement};
```

execution-context の指定

実行可能節では、**常に**明示接続コンテキストを指定する必要があります。ただし、1 つの例外として、FETCH ステートメントには明示接続コンテキストを指定しません。特定の場合に限り、実行コンテキストを組み込みます。実行コンテキストが必要となる場合については、『SQLJ での SQL ステートメントの実行の制御』を参照してください。

connection-context の指定

実行可能節では、接続コンテキストを明示的に指定しない場合、実行可能節はデフォルト接続コンテキストを使用します。

SQLJ アプリケーションでの DB2 オブジェクトの作成および変更

SQLJ 実行可能節を使用して、データ定義ステートメント (CREATE、ALTER、DROP、GRANT、REVOKE)、INSERT ステートメント、検索または位置指定の UPDATE ステートメント、検索または位置指定の DELETE ステートメントを実行できます。

INSERT、検索の UPDATE、検索の DELETE の実行可能ステートメントを以下に示します。

```
#sql [myConnCtx] {INSERT INTO DEPARTMENT VALUES
  ("X00","Operations 2","000030","E01",NULL)};
#sql [myConnCtx] {UPDATE DEPARTMENT
  SET MGRNO="000090" WHERE MGRNO="000030"};
#sql [myConnCtx] {DELETE FROM DEPARTMENT
  WHERE DEPTNO="X00"};
```

SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行

SQLJ で位置指定の UPDATE と DELETE を実行する操作は、他の言語の DB2 アプリケーションの場合と同じく、結果表から行を取得する操作を拡張した操作になります。

基本手順は、以下のとおりです。

1. イテレーターを宣言します。

   イテレーターには、位置指定と名前指定があります。位置指定の UPDATE 操作または DELETE 操作では、以下の節のいずれかまたは両方を使用して、イテレーターを更新可能として宣言します。

   implements sqlj.runtime.ForUpdate
   この節を指定すると、生成されたイテレーター・クラスに更新可能なイテレーターを使用するためのメソッドが組み込まれます。位置指定の UPDATE 操作または DELETE 操作を組み込んだプログラムでは、この節が必須になります。

   with (updateColumns=*"column-list"***)**
   この節では、イテレーターが更新する結果表の各列のコンマ区切りリストを指定します。この節はオプションです。

   イテレーターは `public` として宣言する必要があるので、`public` イテレーターの宣言と使用を同じファイルに記述する場合も別々のファイルに記述する場合も、以下の規則に従わなければなりません。

   ファイルの中でイテレーターを単独で宣言する場合は、そのイテレーターに対するアドレス可能度があり、生成されたクラスをインポートする SQLJ ソース・ファイルから、そのイテレーターを使用してデータを取得し、位置指定の UPDATE ステートメントまたは DELETE ステートメントを実行できます。

   位置指定の UPDATE ステートメントまたは DELETE ステートメントを実行するための許可 ID は、ステートメントを静的に実行するか動的に実行するかによって異なります。ステートメントを静的に実行する場合の許可 ID は、そのステートメントが含まれているプランまたはパッケージの所有者になります。ステートメントを動的に実行する場合の許可 ID は、有効な DYNAMICRULES の動作によって決まります。 IBM Data Server Driver for JDBC and SQLJ の場合、その動作は常に DYNAMICRULES BIND になります。
2. 接続の自動コミット・モードを使用不可にします。

   自動コミット・モードが使用可能になっていると、位置指定の UPDATE ステートメントが実行されるたびに COMMIT 操作が実行され、イテレーターに `with (holdability=true)` 属性が設定されている場合以外はイテレーターが破棄されることになります。したがって、イテレーターを使用する作業が完了するまで COMMIT 操作が実行されないようにするには、自動コミットをオフにしなければなりません。各更新操作の後に COMMIT を実行する場合は、各 COMMIT 操作の後にイテレーターが破棄されないようにするために、イテレーターを `with (holdability=true)` で宣言するという代替方法を使用してください。
3. イテレーター・クラスのインスタンスを作成します。

更新可能でないイテレーターの場合と同じ手順です。

4. SELECT の結果表をイテレーターのインスタンスに割り当てます。

   更新可能でないイテレーターの場合と同じ手順です。FOR UPDATE 節が含まれている SELECT ステートメントは、使用できません。

5. 行を取得して更新します。

   位置指定イテレーターの場合は、そのために以下のアクションをループの中で実行します。

   a. 実行可能節で FETCH ステートメントを実行して現在行を取得します。
   b. イテレーターが結果表の行を参照しているかどうかをテストするために、PositionedIterator.endFetch メソッドを呼び出します。
   c. イテレーターが結果表の行を参照していれば、実行可能節で SQL の UPDATE... WHERE CURRENT OF *:iterator-object* ステートメントを実行して、現在行の各列を更新します。現在行を削除する場合は、実行可能節で SQL の DELETE ... WHERE CURRENT OF *:iterator-object* ステートメントを実行します。

   名前指定イテレーターの場合は、そのために以下のアクションをループの中で実行します。

   a. next メソッドを呼び出して、イテレーターを前方へ移動します。
   b. イテレーターが結果表の行を参照しているかどうかをテストするために、next から `true` が返されるかどうかをチェックします。
   c. 実行可能節で SQL の UPDATE... WHERE CURRENT OF *:iterator-object* ステートメントを実行して、現在行の各列を更新します。現在行を削除する場合は、実行可能節で SQL の DELETE ... WHERE CURRENT OF *iterator-object* ステートメントを実行します。

6. イテレーターを閉じます。

   そのためには、close メソッドを使用します。

位置指定イテレーターを宣言し、そのイテレーターを位置指定の UPDATE で使用するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

まず、1 つのファイルで位置指定イテレーター `UpdByPos` を宣言し、そのイテレーターで列 `SALARY` を更新することを指定します。

```
import java.math.*;    // Import this class for BigDecimal data type
#sql public iterator UpdByPos implements sqlj.runtime.ForUpdate       1
  with(updateColumns="SALARY") (String, BigDecimal);
```

図 32. 位置指定の UPDATE で使用する位置指定イテレーターを宣言する例

次に、別のファイルで、位置指定の UPDATE のために `UpdByPos` を使用します。そのためのコード断片は、以下のとおりです。

```
import sqlj.runtime.*;       // Import files for SQLJ and JDBC APIs
import java.sql.*;
import java.math.*;          // Import this class for BigDecimal data type
import UpdByPos;             // Import the generated iterator class that
                             // was created by the iterator declaration clause
                             // for UpdByName in another file
#sql context HSCtx;          // Create a connnection context class HSCtx
public static void main (String args[])
{
  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
    e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                     // Create a JDBC connection object
  HSjdbccon.setAutoCommit(false);
                     // Set autocommit off so automatic commits       2
                     // do not destroy the cursor between updates
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                     // Create a connection context object
  UpdByPos upditer;  // Declare iterator object of UpdByPos class     3
  String empnum;     // Declares host variable to receive EMPNO
  BigDecimal sal;    // and SALARY column values
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                     4
      WHERE WORKDEPT='D11'};
                     // Assign result table to iterator object
  #sql {FETCH :upditer INTO :empnum,:sal};                            5a
                     // Move cursor to next row
  while (!upditer.endFetch())                                         5b
                     // Check if on a row
  {
    #sql [myConnCtx] {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
      WHERE CURRENT OF :upditer};                                     5c
                     // Perform positioned update
    System.out.println("Updating row for " + empnum);
    #sql {FETCH :upditer INTO :empnum,:sal};
                     // Move cursor to next row
  }
  upditer.close();   // Close the iterator                           6
  #sql [myConnCtx] {COMMIT};
                     // Commit the changes
  myConnCtx.close(); // Close the connection context
}
```

図 33. 位置指定イテレーターで位置指定の UPDATE を実行する例

名前指定イテレーターを宣言し、そのイテレーターを位置指定の UPDATE で使用するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

まず、1 つのファイルで名前指定イテレーター UpdByName を宣言し、そのイテレーターで列 SALARY を更新することを指定します。

```
import java.math.*;          // Import this class for BigDecimal data type
#sql public iterator UpdByName implements sqlj.runtime.ForUpdate     1
  with(updateColumns="SALARY") (String EmpNo, BigDecimal Salary);
```

図 34. 位置指定の UPDATE で使用する名前指定イテレーターを宣言する例

次に、別のファイルで、位置指定の UPDATE のために UpdByName を使用します。そのためのコード断片は、以下のとおりです。

```
import sqlj.runtime.*;       // Import files for SQLJ and JDBC APIs
import java.sql.*;
import java.math.*;          // Import this class for BigDecimal data type
import UpdByName;            // Import the generated iterator class that
                             // was created by the iterator declaration clause
                             // for UpdByName in another file
#sql context HSCtx;          // Create a connnection context class HSCtx
public static void main (String args[])
{
  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch (ClassNotFoundException e) {
    e.printStackTrace();
          }
  Connection HSjdbccon=
  DriverManager.getConnection("jdbc:db2:SANJOSE");
                         // Create a JDBC connection object
  HSjdbccon.setAutoCommit(false);
                         // Set autocommit off so automatic commits  2
                         // do not destroy the cursor between updates
  HSCtx myConnCtx=new HSCtx(HSjdbccon);
                         // Create a connection context object
  UpdByName upditer;                                                  3
                         // Declare iterator object of UpdByName class
  String empnum;               // Declare host variable to receive EmpNo
                             // column values
  #sql [myConnCtx]
    upditer = {SELECT EMPNO, SALARY FROM EMPLOYEE                     4
      WHERE WORKDEPT='D11'};
                             // Assign result table to iterator object
  while (upditer.next())                                              5a,5b
                             // Move cursor to next row and
                             // check ifon a row
  {
    empnum = upditer.EmpNo();  // Get employee number from current row
    #sql [myConnCtx]
      {UPDATE EMPLOYEE SET SALARY=SALARY*1.05
        WHERE CURRENT OF :upditer};                                   5c
                             // Perform positioned update
    System.out.println("Updating row for " + empnum);
  }
  upditer.close();           // Close the iterator                   6
  #sql [myConnCtx] {COMMIT};
                             // Commit the changes
  myConnCtx.close();         // Close the connection context
}
```

図 35. 名前指定イテレーターで位置指定の UPDATE を実行する例

SQLJ アプリケーションでの、位置指定 UPDATE または DELETE 操作の変数としてのイテレーターの受け渡し

SQLJ はイテレーターを変数としてメソッド間で渡すことを許可します。

位置指定 UPDATE または DELETE ステートメントに使用されるイテレーターは、実行時にのみ識別できます。同じ SQLJ 位置指定 UPDATE または DELETE ステートメントを別のイテレーターと共に実行時に使用できます。 SQLJ アプリケーシ

ョンをプログラム準備処理の一部としてカスタマイズする際に -staticpositioned に値 YES を指定する場合、SQLJ カスタマイザーは位置指定 UPDATE または DELETE ステートメントを静的に実行する準備を行います。この場合、カスタマイザーはどのイテレーターがどの位置指定 UPDATE または DELETE ステートメントに属するのかを判別する必要があります。 SQLJ カスタマイザーは、イテレーター・データ・タイプを UPDATE または DELETE ステートメント内のデータ・タイプにマッチングさせることによってこれを行います。ただし、UPDATE または DELETE ステートメント内の表からイテレーター・クラスへの固有のマッピングが存在しない場合、SQLJ カスタマイザーはどのイテレーターと UPDATE または DELETE ステートメントが組になるのかを正確には判別できません。 SQLJ カスタマイザーはイテレーターと UPDATE または DELETE ステートメントを任意のペアにしなければならず、その結果として SQL エラーが生じることもあります。以下のコード・フラグメントはこの点を示しています。

```
#sql iterator GeneralIter implements sqlj.runtime.ForUpdate
  ( String );

  public static void main ( String args[] )
  {
...
    GeneralIter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };

    GeneralIter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };
...

    doUpdate ( iter1 );
  }

  public static void doUpdate ( GeneralIter iter )
  {
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
  }
```

図 36. 失敗する静的な位置指定 UPDATE

この例では、1 つのイテレーターのみ宣言されます。そのイテレーターの 2 つのインスタンスが宣言され、それぞれが、異なる表からデータを検索する異なる SELECT ステートメントに関連付けられます。カスタマイズおよび -staticpositioned YES とのバインドの際に、SQLJ は 2 つの DECLARE CURSOR ステートメント (各 SELECT ステートメントに対して 1 つずつ) を作成し、UPDATE ステートメントをそれぞれのカーソルにバインドしようとします。しかし、バインド処理は、`UPDATE TABLE1 ... WHERE CURRENT OF :iter` が `SELECT CHAR_COL2 FROM TABLE2` のカーソルにバインドされるとき、UPDATE の表がカーソルの表と一致しないために、SQLCODE -509 で失敗します。

図 36 のようなプログラムでバインド時のエラーを避けるには、バインド・オプション SQLERROR(CONTINUE) を指定します。しかし、この手法には、プログラム内に SQL エラーがある場合でも、 DB2 データベース・マネージャーがパッケージを作成してしまうという欠点があります。より優れた手法は、位置指定 UPDATE または DELETE ステートメント内の表とイテレーター・クラスの間に 1 対 1 のマッピングが存在するようにプログラムを作成することです。 167 ページの図 37はこれを行う方法の例を示しています。

```
#sql iterator Table2Iter(String);
#sql iterator Table1Iter(String);
  public static void main ( String args[] )
  {
...
    Table2Iter iter2 = null;
    #sql [ctxt] iter2 = { SELECT CHAR_COL2 FROM TABLE2 };

    Table1Iter iter1 = null;
    #sql [ctxt] iter1 = { SELECT CHAR_COL1 FROM TABLE1 };
...

    doUpdate(iter1);

  }

 public static void doUpdate ( Table1Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE1 ... WHERE CURRENT OF :iter };
    ...
  }
 public static void doUpdate ( Table2Iter iter )
  {
    ...
    #sql [ctxt] { UPDATE TABLE2 ... WHERE CURRENT OF :iter };
    ...
  }
```

図 37. 成功する静的な位置指定 UPDATE

このコーディング方法を使用すると、各イテレーター・クラスは 1 つの表にのみ関連付けられます。そのため、 DB2 バインド処理では、常に位置指定 UPDATE ステートメントを有効なイテレーターに関連付けることができます。

SQLJ アプリケーションでのバッチ更新の実行

IBM Data Server Driver for JDBC and SQLJ は、SQLJ のバッチ更新をサポートしています。表の各行を 1 行ずつ更新する代わりにバッチ更新を使用すれば、SQLJ によって一群の更新を同時に実行できます。

バッチ更新には、以下のタイプのステートメントを組み込めます。

- 検索の INSERT、UPDATE、または DELETE、または MERGE の各ステートメント
- CREATE、ALTER、DROP、GRANT、REVOKE の各ステートメント
- 入力パラメーターだけを指定した CALL ステートメント

JDBC の場合とは異なり、SQLJ では、入力パラメーターまたはホスト式を含んだステートメントを組み込んだ異種バッチを使用できます。したがって、SQLJ バッチでは、以下の項目を自由に組み合わせることができます。

- 同じステートメントの各インスタンス
- 別々のステートメント
- 異なる数の入力パラメーターまたはホスト式を含んだステートメント
- 異なるデータ・タイプの入力パラメーターまたはホスト式を含んだステートメント
- 入力パラメーターまたはホスト式がないステートメント

バッチに含まれている 1 つのステートメントの実行中にエラーが発生しても、残りのステートメントは実行され、バッチ内のすべてのステートメントが実行されてから、BatchUpdateException がスローされます。

警告に関する情報を取得するには、バッチの中に組み込むステートメントをサブミットするのに使用する ExecutionContext で ExecutionContext.getWarnings メソッドを使用します。これで、各 SQLWarning オブジェクトのエラーに関する記述、SQLSTATE、およびエラー・コードを取得できます。

バッチに追加できないステートメントがプログラムに含まれているために、バッチが暗黙的に実行される場合は、バッチが実行されてから、その新しいステートメントが処理されます。バッチの実行中にエラーが発生すると、バッチ実行の原因になったステートメントは実行されません。

ステートメント・バッチの作成、実行、削除の基本手順は、以下のとおりです。

1. 接続の `AutoCommit` を使用不可にします。

   そのようにしておくと、バッチの実行中にエラーが発生した場合に、既に実行されたステートメントの変更内容をコミットするかどうかを自分で制御できます。
2. 実行コンテキストを取得します。

   バッチの中で実行するすべてのステートメントは、その実行コンテキストを使用します。
3. ExecutionContext.setBatching(true) メソッドを呼び出して、バッチを作成します。

   手順 2 で作成した実行コンテキストに関連した後続のバッチ可能ステートメントは、その後の実行時にバッチに追加されます。

   バッチ互換でないステートメント・セットを並列的にバッチに組み込む場合は、バッチ互換ステートメントの各セットごとに実行コンテキストを作成する必要があります。
4. バッチに含める SQL ステートメントの SQLJ 実行可能節を組み込みます。

   それらの節には、手順 2 で作成した実行コンテキストを組み込む必要があります。

   SQLJ 実行可能節に入力パラメーターまたはホスト式が含まれている場合は、バッチの中にそのステートメントを複数回組み込んで、それぞれに入力パラメーターまたはホスト式の別々の値を適用することもできます。

   ステートメントが既存のバッチに追加されたかどうか、ステートメントが新しいバッチの最初のステートメントかどうか、ステートメントが実行されたのはバッチの中か外かを確認するには、ExecutionContext.getUpdateCount メソッドを呼び出します。このメソッドは、以下のいずれかの値を返します。

   `ExecutionContext.ADD_BATCH_COUNT`
   ステートメントが既存のバッチに追加された場合は、この定数が返されます。

ExecutionContext.NEW_BATCH_COUNT
ステートメントが新しいバッチの最初のステートメントの場合は、この定数が返されます。

ExecutionContext.EXEC_BATCH_COUNT
ステートメントがバッチに含まれている状態でバッチが実行された場合は、この定数が返されます。

その他の整数
この値は、ステートメントにより更新された行の数です。ステートメントがバッチに追加されない状態で実行された場合は、この値が返されます。

5. 明示的または暗黙的にバッチを実行します。
   - バッチを明示的に実行するには、ExecutionContext.executeBatch メソッドを呼び出します。

     executeBatch からは、バッチに含まれている各ステートメントによって更新された行の数を含んだ整数配列が返されます。配列内の各エレメントの順序は、各ステートメントをバッチに追加した順序に対応します。
   - バッチは、以下の状況では暗黙的に実行されます。
     - 既にバッチに含まれているステートメントとの互換性のないバッチ可能ステートメントをプログラムに組み込んだ場合。この場合、SQLJ は、既にバッチに含まれているステートメントを実行し、非互換ステートメントを組み込んだ新しいバッチを作成します。
     - バッチ可能でないステートメントをプログラムに組み込んだ場合。この場合、SQLJ は、既にバッチに含まれているステートメントを実行します。SQLJ は、バッチ可能でないステートメントも実行します。
     - `ExecutionContext.setBatchLimit(n)` メソッドを呼び出した後に、バッチにステートメントを追加して、バッチ内のステートメントの数が *n* 以上になる場合。*n* は、以下のいずれかの値になります。

       ExecutionContext.UNLIMITED_BATCH
       この定数を指定した場合は、バッチ可能ではあっても非互換のステートメント、またはバッチ可能でないステートメントを SQLJ が検出した場合にのみ、バッチが暗黙的に実行されます。この値を設定することは、setBatchLimit を呼び出さないことと同じです。

       ExecutionContext.AUTO_BATCH
       この定数を指定した場合は、バッチ内のステートメントの数が SQLJ によって設定されている数に達した場合に、バッチが暗黙的に実行されます。

       正の整数
       この数のステートメントがバッチに追加されると、SQLJ によってバッチが暗黙的に実行されます。ただし、バッチ可能ではあっても非互換のステートメント、またはバッチ可能でないステートメントを SQLJ が検出した場合は、その数のステートメントが追加される前にバッチが実行される可能性があります。

     暗黙的に実行されたバッチで更新された行の数を確認するには、ExecutionContext.getBatchUpdateCounts メソッドを呼び出します。getBatchUpdateCounts からは、バッチに含まれている各ステートメントによっ

て更新された行の数を含んだ整数配列が返されます。配列内の各エレメントの順序は、各ステートメントをバッチに追加した順序に対応します。それぞれの配列エレメントには、以下のいずれかの値が入ります。

-2 この値は、SQL ステートメントは正常に実行されたが、更新された行数は判別できなかったことを示します。

-3 この値は SQL ステートメントが失敗したことを示します。

その他の整数
この値は、ステートメントにより更新された行の数です。

6. オプションとして、すべてのステートメントをバッチに追加した後に、バッチ処理を使用不可にすることもできます。

   そのためには、ExecutionContext.setBatching(false) メソッドを呼び出します。バッチ処理を使用不可にしても、暗黙的または明示的にバッチを実行することは可能ですが、バッチにさらにステートメントを追加することができなくなります。バッチが既に存在する場合に、バッチ互換ステートメントをバッチに追加する代わりにそのまま実行することが望ましい状況では、バッチ処理を使用不可にすると便利です。

   バッチを実行しないでクリアする場合は、ExecutionContext.cancel メソッドを呼び出します。

7. バッチ実行が暗黙的だった場合は、最後に明示的な executeBatch を実行して、すべてのステートメントを確実に実行するようにします。

バッチの中で UPDATE を実行して、すべてのマネージャー (管理職) に昇給を適用するコード断片を以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql iterator GetMgr(String);       // Declare positioned iterator
...
{
  GetMgr deptiter;                  // Declare object of GetMgr class
  String mgrnum = null;             // Declare host variable for manager number
  int raise = 400;                  // Declare raise amount
  int currentSalary;                // Declare current salary
  String url, username, password;   // Declare url, user ID, password
  ...
  TestContext c1 = new TestContext (url, username, password, false); 1
  ExecutionContext ec = new ExecutionContext();                      2
  ec.setBatching(true);                                              3

  #sql [c1] deptiter =
    {SELECT MGRNO FROM DEPARTMENT};
                                    // Assign the result table of the SELECT
                                    // to iterator object deptiter
  #sql {FETCH :deptiter INTO :mgrnum};
                                    // Retrieve the first manager number
  while (!deptiter.endFetch()) {    // Check whether the FETCH returned a row
    #sql [c1]
      {SELECT SALARY INTO :currentSalary FROM EMPLOYEE
        WHERE EMPNO=:mgrnum};
    #sql [c1, ec]                                                    4
      {UPDATE EMPLOYEE SET SALARY=:(currentSalary+raise)
        WHERE EMPNO=:mgrnum};
    #sql {FETCH :deptiter INTO :mgrnum };
                                    // Fetch the next row
  }
  ec.executeBatch();                                                 5
  ec.setBatching(false);                                             6
  #sql [c1] {COMMIT};
  deptiter.close();                 // Close the iterator
  c1.close();                       // Close the connection
}
```

図 38. バッチ更新の実行例

SQLJ アプリケーションでのデータの検索

SQLJ アプリケーションは*結果セット・イテレーター* を使用して、結果セットを検索します。カーソルと同様に、結果セット・イテレーターはスクロール不可にもスクロール可能にもできます。

他の言語の DB2 アプリケーションと同様に、SQLJ アプリケーションで表から単一行を検索する場合、その行のみを含む結果表を定義する、WHERE 節を指定した SELECT INTO ステートメントを作成できます。

```
#sql [myConnCtx] {SELECT DEPTNO INTO :hvdeptno
  FROM DEPARTMENT WHERE DEPTNAME="OPERATIONS"};
```

しかし、使用するほとんどの SELECT ステートメントでは、多数の行を含む結果表が作成されます。他の言語の DB2 アプリケーションでは、カーソルを使用して個々の行を結果表から選択します。そのカーソルはスクロール不可にすることができます。つまり、カーソルを使用して行をフェッチする場合、カーソルを結果表の最初から最後まで連続的に移動させます。あるいは、カーソルをスクロール可能にすることもできます。この場合、カーソルを使用して行をフェッチするときは、カーソルを順方向、逆方向、または結果表の任意の行に移動させることができます。

このトピックでは、スクロール不可イテレーターの使用方法を解説します。スクロール可能イテレーターの使用について詳しくは、『SQLJ アプリケーションでのスクロール可能イテレーターの使用』を参照してください。

結果セット・イテレーターとは、結果表から行を取得するために使用する Java オブジェクトです。カーソルとは異なり、結果セット・イテレーターはパラメーターとしてメソッドに渡すことができます。

結果セット・イテレーターを使用する基本ステップは次のとおりです。
1. イテレーターを宣言します。それがイテレーター・クラスになります。
2. イテレーター・クラスのインスタンスを定義します。
3. SELECT の結果表をイテレーターのインスタンスに割り当てます。
4. 行を取得します。
5. イテレーターを閉じます。

イテレーターには、*位置指定イテレーター* と *名前指定イテレーター* という 2 つのタイプがあります。位置指定イテレーターはインターフェース sqlj.runtime.PositionedIterator を拡張します。位置指定イテレーターは結果表の列を、結果表内での位置で識別します。名前指定イテレーターはインターフェース sqlj.runtime.NamedIterator を拡張します。名前指定イテレーターは結果表の列を、結果表の列名で識別します。

SQLJ アプリケーションでの名前指定イテレーターの使用

名前指定イテレーターを使用して、結果表の各列を名前で参照できます。

名前指定イテレーターを使用するための手順は、以下のとおりです。

1. イテレーターを宣言します。

   結果セット・イテレーターを宣言するには、*イテレーター宣言節* を使用します。そうすると、イテレーターと同じ名前のイテレーター・クラスが作成されます。名前指定イテレーターの場合は、イテレーター宣言節で以下の情報を指定します。

   - イテレーターの名前
   - 列名と Java データ・タイプのリスト
   - Java クラス宣言のための情報 (イテレーターが `public` か `static` か、など)
   - 一連の属性 (イテレーターが保持可能かどうか、各列が更新可能かどうか、など)

   照会の名前指定イテレーターを宣言するときには、イテレーターの各列の名前を指定します。指定する名前は、照会の結果表の各列の名前と一致している必要があります。イテレーターの列名と結果表の列名の大/小文字だけが違っている場合は、一致する名前と見なされます。イテレーター宣言節から生成される名前指定イテレーター・クラスには、*accessor メソッド* が含まれています。イテレーターの各列ごとに 1 つの accessor メソッドが存在します。各 accessor メソッドの名前は、対応するイテレーター列名と同じです。その accessor メソッドを使用して、結果表の各列のデータを取得します。

イテレーターには、対応する DB2 列のデータ・タイプに近い Java データ・タイプを指定する必要があります。Java データ・タイプと DB2 データ・タイプの最適な対応関係のリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

イテレーターを宣言するには、いくつかの方法があります。ただし、各イテレーターは Java クラスが基礎になっているので、イテレーターを宣言するときには、基礎になっているクラスが Java の規則に準拠していることを確認する必要があります。例えば、*with 節* が含まれているイテレーターは、public として宣言しなければなりません。したがって、public にしなければならないイテレーターは、public クラスを使用できる場所でのみ宣言できる、ということになります。イテレーターを宣言するためのいくつかの代替方法を以下のリストにまとめます。

- ソース・ファイルの中で単独で public として宣言する方法

  この方法の場合は、他のコード・モジュールの中でもイテレーター宣言を使用できるので、すべての SQLJ アプリケーションで有効なイテレーターを用意できます。さらに、同じソース・ファイルの中に他の最上位クラスや public クラスがあるかどうかを気にする必要はありません。
- ソース・ファイルの中で他の最上位クラス定義と一緒に最上位クラスとして宣言する方法

  Java では、1 つのコード・モジュールの中に public の最上位クラスを 1 つしか宣言できません。したがって、イテレーターを public として宣言しなければならない場合 (イテレーターに with 節を組み込む場合など) は、コード・モジュール内の他のクラスを public として宣言することはできません。
- 別のクラスの中にネストした static クラスとして宣言する方法

  この代替方法の場合は、イテレーター宣言と他のクラス宣言を同じソース・ファイルの中で組み合わせ、イテレーターと他のクラスを public として宣言し、イテレーター・クラスを他のコード・モジュールやパッケージから参照できるようにすることが可能になります。ただし、ネスト元のクラスの外からイテレーターを参照する場合は、ネスト元のクラスの名前を付けた完全修飾イテレーター名を指定する必要があります。
- 別のクラスの inner クラスとして宣言する方法

  この方法でイテレーターを宣言する場合は、ネスト元のクラスのインスタンスの中でしかイテレーターのインスタンスを生成できません。ただし、ファイルの中でイテレーターと他のクラスを public として宣言することは可能です。

  イテレーターを inner クラスとして宣言した場合は、JDBC の ResultSet をイテレーターにキャストできません。この制限は、static のネスト・クラスとして宣言したイテレーターには当てはまりません。ResultSet をイテレーターにキャストする方法の詳細については、『同じアプリケーションでの SQLJ と JDBC の使用』を参照してください。

2. イテレーター・クラスのインスタンスを作成します。

結果表から行を取得するための名前指定イテレーター・クラスのオブジェクトを宣言します。

3. SELECT の結果表をイテレーターのインスタンスに割り当てます。

   SELECT の結果表をイテレーターに割り当てるには、SQLJ の *代入節* を使用します。名前指定イテレーターの代入節の形式は、以下のとおりです。

   ```
   #sql context-clause iterator-object={select-statement};
   ```

   詳細については、『SQLJ 代入節』と『SQLJ コンテキスト節』を参照してください。

4. 行を取得します。

   そのためには、ループの中で accessor メソッドを呼び出します。accessor メソッドの名前は、対応するイテレーターの列の名前と同じです。accessor メソッドにパラメーターはありません。accessor メソッドからは、結果表の現在行の対応する列の値が返されます。結果表の中でカーソルを前方へ移動するには、NamedIterator.next() メソッドを使用します。

   すべての行を取得できたかどうかをテストするには、next メソッドを呼び出したときに返される値をチェックします。次の行がない場合は、next から `false` の値の `boolean` が返されます。

5. イテレーターを閉じます。

   そのためには、NamedIterator.close メソッドを使用します。

名前指定イテレーターを宣言して使用するためのコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql  iterator ByName(String LastName, Date HireDate);                    1
                                // Declare named iterator ByName
{
  ...
  ByName nameiter;              // Declare object of ByName class          2
  #sql [ctxt]
 nameiter={SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                        3
                                // Assign the result table of the SELECT
                                // to iterator object nameiter
  while (nameiter.next())       // Move the iterator through the result    4
                                // table and test whether all rows retrieved
  {
    System.out.println( nameiter.LastName() + " was hired on "
      + nameiter.HireDate());   // Use accessor methods LastName and
                                // HireDate to retrieve column values
  }
  nameiter.close();             // Close the iterator                      5
}
```

図 39. 名前指定イテレーターの使用例

SQLJ アプリケーションでの位置指定イテレーターの使用

位置指定イテレーターを使用すれば、結果表の各列を結果セット内の位置で参照できます。

位置指定イテレーターを使用するための手順は、以下のとおりです。

1. イテレーターを宣言します。

   結果セット・イテレーターを宣言するには、*イテレーター宣言節* を使用します。そうすると、イテレーターと同じ名前、同じ属性のイテレーター・クラスが作成されます。位置指定イテレーターの場合は、イテレーター宣言節で以下の情報を指定します。

   - イテレーターの名前
   - Java データ・タイプのリスト
   - Java クラス宣言のための情報 (イテレーターが `public` か `static` か、など)
   - 一連の属性 (イテレーターが保持可能かどうか、各列が更新可能かどうか、など)

   データ・タイプ宣言は、結果表の列に対応しており、その宣言のことを結果セット・イテレーターの列といいます。結果セット・イテレーターの列は、左から右の順で結果表の列に対応しています。例えば、イテレーター宣言節に 2 つのデータ・タイプ宣言があれば、最初のデータ・タイプ宣言が結果表の最初の列に対応し、2 番目のデータ・タイプ宣言が結果表の 2 番目の列に対応する、という具合になります。

   イテレーターには、対応する DB2 列のデータ・タイプに近い Java データ・タイプを指定する必要があります。Java データ・タイプと DB2 データ・タイプの最適な対応関係のリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

   イテレーターを宣言するには、いくつかの方法があります。ただし、各イテレーターは Java クラスが基礎になっているので、イテレーターを宣言するときには、基礎になっているクラスが Java の規則に準拠していることを確認する必要があります。例えば、*with 節* が含まれているイテレーターは、`public` として宣言しなければなりません。したがって、`public` にしなければならないイテレーターは、`public` クラスを使用できる場所でのみ宣言できる、ということになります。イテレーターを宣言するためのいくつかの代替方法を以下のリストにまとめます。

   - ソース・ファイルの中で単独で `public` として宣言する方法

     これは、イテレーターを宣言するための最も汎用的な方法です。この方法の場合は、他のコード・モジュールの中でもイテレーター宣言を使用できるので、すべての SQLJ アプリケーションで有効なイテレーターを用意できます。さらに、同じソース・ファイルの中に他の最上位クラスや `public` クラスがあるかどうかを気にする必要はありません。

   - ソース・ファイルの中で他の最上位クラス定義と一緒に最上位クラスとして宣言する方法

     Java では、1 つのコード・モジュールの中に public の最上位クラスを 1 つしか宣言できません。したがって、イテレーターを public として宣言しなければならない場合 (イテレーターに with 節を組み込む場合など) は、コード・モジュール内の他のクラスを public として宣言することはできません。

   - 別のクラスの中にネストした static クラスとして宣言する方法

この代替方法の場合は、イテレーター宣言と他のクラス宣言を同じソース・ファイルの中で組み合わせ、イテレーターと他のクラスを public として宣言し、イテレーター・クラスを他のコード・モジュールやパッケージから参照できるようにすることが可能になります。ただし、ネスト元のクラスの外からイテレーターを参照する場合は、ネスト元のクラスの名前を付けた完全修飾イテレーター名を指定する必要があります。

- 別のクラスの inner クラスとして宣言する方法

  この方法でイテレーターを宣言する場合は、ネスト元のクラスのインスタンスの中でしかイテレーターのインスタンスを生成できません。ただし、ファイルの中でイテレーターと他のクラスを public として宣言することは可能です。

  イテレーターを inner クラスとして宣言した場合は、JDBC の ResultSet をイテレーターにキャストできません。この制限は、static のネスト・クラスとして宣言したイテレーターには当てはまりません。ResultSet をイテレーターにキャストする方法の詳細については、『同じアプリケーションでの SQLJ と JDBC の使用』を参照してください。

2. イテレーター・クラスのインスタンスを作成します。

   結果表から行を取得するための位置指定イテレーター・クラスのオブジェクトを宣言します。

3. SELECT の結果表をイテレーターのインスタンスに割り当てます。

   SELECT の結果表をイテレーターに割り当てるには、SQLJ の *代入節* を使用します。位置指定イテレーターの代入節の形式は、以下のとおりです。

   ```
   #sql context-clause iterator-object={select-statement};
   ```

4. 行を取得します。

   そのためには、ループの実行可能節の中で FETCH ステートメントを実行します。この FETCH ステートメントは、他の言語の FETCH ステートメントと同じ形式になります。

   すべての行を取得できたかどうかをテストするには、各 FETCH の後で PositionedIterator.endFetchメソッドを呼び出します。取得する行がなくて FETCH が失敗した場合は、endFetch から `true` の値の `boolean` が返されます。

5. イテレーターを閉じます。

   そのためには、PositionedIterator.close メソッドを使用します。

位置指定イテレーターを宣言して使用するためのコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql iterator ByPos(String,Date); // Declare positioned iterator ByPos  1
{
  ...
  ByPos positer;                    // Declare object of ByPos class    2
  String name = null;               // Declare host variables
  Date hrdate;
  #sql [ctxt] positer =
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                          3
                                    // Assign the result table of the SELECT
                                    // to iterator object positer
  #sql {FETCH :positer INTO :name, :hrdate };                           4
                                    // Retrieve the first row
  while (!positer.endFetch())       // Check whether the FETCH returned a row
  { System.out.println(name + " was hired in " +
      hrdate);
    #sql {FETCH :positer INTO :name, :hrdate };
                                    // Fetch the next row
  }
  positer.close();                  // Close the iterator                5
}
```

図 40. 位置指定イテレーターの使用例

SQLJ アプリケーションにおける同じ SQL ステートメントの複数のオープン・イテレーター

IBM Data Server Driver for JDBC and SQLJ により、ご使用のアプリケーションは SQLJ アプリケーション内の単一 SQL ステートメントの複数のイテレーターを同時にオープンできます。この機能によって、1 つのイテレーターを使用して表である操作を実行しながら、別のイテレーターを使用して同じ表で別の操作を実行できます。

アプリケーションで同時にオープンしているイテレーターを使用する場合、それが不要になったら Java ヒープでの過剰なストレージ消費を防ぐためにそれらのイテレーターをクローズする必要があります。

以下の例では、単一の SQL ステートメントで同時にオープンしているイテレーターがない場合と、単一の SQL ステートメントで同時にオープンしているイテレーターがある場合に、表に対して同じ操作を実行する方法を示しています。これらの例では、次のイテレーター宣言を使用します。

```
import java.math.*;
#sql public iterator  MultiIter(String EmpNo, BigDecimal Salary);
```

単一の SQL ステートメントで複数のイテレーターを同時にオープンできない場合、特定の従業員番号の従業員および給与の値を選択するには、 178 ページの図 41 に示されるように、従業員番号ごとに異なる SQL ステートメントを定義する必要があります。

```
MultiIter iter1 = null;            // Iterator instance for retrieving
                                   // data for first employee
String EmpNo1 = "000100";          // Employee number for first employee
#sql [ctx] iter1 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo1};
                                   // Assign result table to first iterator
MultiIter iter2 = null;            // Iterator instance for retrieving
                                   // data for second employee
String EmpNo2 = "000200";          // Employee number for second employee
#sql [ctx] iter2 =
  {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo2};
                                   // Assign result table to second iterator
// Process with iter1
// Process with iter2
iter1.close();                     // Close the iterators
iter2.close();
```

図 41. 異なる SQL ステートメントでイテレーターを使用して表に対する操作を同時に実行する例

図 42 では、単一の SQL ステートメントで複数のイテレーターを同時にオープンできる場合に、同じ操作を実行する方法を示しています。

```
...
MultiIter iter1 = openIter("000100"); // Invoke openIter to assign the result table
                                      // (for employee 100) to the first iterator
MultiIter iter2 = openIter("000200"); // Invoke openIter to assign the result
                                      // table to the second iterator
                                      // iter1 stays open when iter2 is opened
// Process with iter1
// Process with iter2
...
iter1.close();                        // Close the iterators
iter2.close();
...
public MultiIter openIter(String EmpNo)
                                      // Method to assign a result table
                                      // to an iterator instance
{
  MultiIter iter;
  #sql [ctxt] iter =
    {SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = :EmpNo};
  return iter;                        // Method returns an iterator instance
}
```

図 42. 同じ SQL ステートメントでイテレーターを使用して表に対する操作を同時に実行する例

SQLJ アプリケーションにおけるイテレーターの複数のオープン・インスタンス

イテレーターの複数インスタンスを単一の SQLJ アプリケーションで同時にオープンできます。この機能の応用の 1 つは、ホスト式を使用するイテレーターの複数インスタンスをオープンすることです。それぞれのインスタンスは様々なホスト式の値のセットを使用できます。

以下の例は、イテレーターの 2 つのインスタンスが同時にオープンしているアプリケーションを示しています。

```
...
ResultSet myFunc(String empid)  // Method to open an iterator and get a resultSet
{
  MyIter iter;
  #sql iter = {SELECT * FROM EMPLOYEE WHERE EMPNO = :empid};
  return iter.getResultSet();
}

// An application can call this method to get a resultSet for each
// employee ID. The application can process each resultSet separately.
...
ResultSet rs1 = myFunc("000100");  // Get employee record for employee ID 000100
...
ResultSet rs2 = myFunc("000200");  // Get employee record for employee ID 000200
```

図 43. 単一アプリケーションでイテレーターの複数インスタンスをオープンする例

その他のイテレーターの場合と同様に、このイテレーターを最後に使用した後は、過剰なストレージ消費を防ぐために必ずクローズする必要があります。

SQLJ アプリケーションでのスクロール可能イテレーターの使用

結果表の中では、1 行ずつ前方へ移動する機能だけでなく、後方へ移動したり、特定の行に直接移動したりする機能があれば便利です。 IBM Data Server Driver for JDBC and SQLJ には、そのための機能が用意されています。

前方への移動、後方への移動、特定の行への移動が可能なイテレーターのことをスクロール可能イテレーター といいます。SQLJ のスクロール可能イテレーターは、SCROLL として宣言したデータベース・カーソルの結果表と等価です。

両方向スクロール・カーソルの場合と同じく、スクロール可能イテレーターにも、インセンシティブ とセンシティブ があります。センシティブのスクロール可能イテレーターには、*静的* と*動的* があります。インセンシティブとは、イテレーターを開いた後の基礎表の変更がイテレーターから不可視になる、という意味です。インセンシティブ・イテレーターは、読み取り専用です。センシティブとは、イテレーターまたは他のプロセスが基礎表に加えた変更がイテレーターから可視になる、という意味です。アセンシティブとは、カーソルが読み取り専用の場合に、インセンシティブ・カーソルとして動作する、という意味です。読み取り専用カーソルでなければ、センシティブ・カーソルとして動作します。

スクロール可能イテレーターが静的であれば、結果表のサイズと各行の順序は、イテレーターを開いた後も変わりません。したがって、結果表に行を挿入することはできませんし、結果表の行を削除すると、削除ホールが発生します。結果表の行を更新して、行が結果表の資格を満たさなくなると、更新ホールが発生します。ホールからフェッチすると、SQLException になります。

スクロール可能イテレーターが動的であれば、結果表のサイズと各行の順序は、イテレーターを開いた後に変わる可能性があります。同じアプリケーション・プロセスで実行される INSERT ステートメントや DELETE ステートメントで挿入されたり削除されたりした行は、すぐに可視になります。他のアプリケーション・プロセスで実行される INSERT ステートメントや DELETE ステートメントで挿入されたり削除されたりした行は、変更のコミット後に可視になります。

重要: DB2 Database for Linux, UNIX, and Windows サーバーは動的両方向スクロール・カーソルをサポートしません。 SQLJ アプリケーションで動的なスクロール

可能イテレーターを使用できるのは、そのアプリケーションがバージョン 9 以降の DB2 for z/OS サーバーのデータにアクセスする場合に限られます。

重要:

スクロール可能イテレーターを作成して使用するには、以下の手順を実行する必要があります。

1. 以下の節を組み込んだイテレーター宣言節を指定します。
   - `implements sqlj.runtime.Scrollable`

     イテレーターをスクロール可能として指定します。
   - `with (sensitivity=INSENSITIVE|SENSITIVE|ASENSITIVE)` または `with (sensitivity=SENSITIVE, dynamic=true|false)`

     `sensitivity=INSENSITIVE|SENSITIVE|ASENSITIVE` では、基礎表の更新操作や削除操作をイテレーターから可視にするかどうかを指定します。デフォルトの sensitivity は `INSENSITIVE` です。

     `dynamic=true|false` では、イテレーターを開いた後に、結果表のサイズと各行の順序を変更可能にするかどうかを指定します。dynamic のデフォルト値は `false` です。

   イテレーターには、位置指定イテレーターと名前指定イテレーターがあります。

   例: 以下のイテレーター宣言節では、位置指定、センシティブ、動的、スクロール可能のイテレーターを指定しています。

   ```
   #sql public iterator ByPos
     implements sqlj.runtime.Scrollable
     with (sensitivity=SENSITIVE, dynamic=true)  (String);
   ```

   例: 以下のイテレーター宣言節では、名前指定、インセンシティブ、スクロール可能のイテレーターを指定しています。

   ```
   #sql public iterator ByName
     implements sqlj.runtime.Scrollable
     with (sensitivity=INSENSITIVE) (String EmpNo);
   ```

   制約事項: DB2 Database for Linux, UNIX, and Windows サーバーでは、スクロール可能イテレーターを使用して、表から以下のデータ・タイプの列を選択することができません。
   - LONG VARCHAR
   - LONG VARGRAPHIC
   - BLOB
   - CLOB
   - XML
   - このリスト内の前述のデータ・タイプのいずれかに基づいた特殊タイプ
   - 構造化タイプ
2. イテレーター・クラスのインスタンスであるイテレーター・オブジェクトを作成します。
3. 初期のフェッチ方向に関するヒントを SQLJ ランタイム環境に与えるには、`setFetchDirection(int direction)` メソッドを使用します。*direction* は、

FETCH_FORWARD と FETCH_REVERSE のいずれかになります。setFetchDirection を呼び出さない場合のフェッチ方向は、FETCH_FORWARD です。

4. アクセスする各行ごとに以下のようにします。

   名前指定イテレーターの場合は、以下の手順を実行します。

   a. 以下の表にまとめられているメソッドのいずれかを使用して、カーソルの位置を指定します。

表 30. 両方向スクロール・カーソルの位置を指定するための sqlj.runtime.Scrollable メソッド

メソッド	カーソルの位置
first[1]	結果表の最初の行
last[1]	結果表の最後の行
previous[1,2]	結果表の前の行
next	結果表の次の行
absolute(int *n*)[1,3]	*n*>0 の場合は、結果表の行 *n*。 *n*<0 の場合は、結果表の行 *m*+*n*+*1* (*m* は結果表の行数)。
relative(int *n*)[1,4]	*n*>0 の場合、現在行の *n* 行後ろにある行。 *n*<0 の場合、現在行の *n* 行前にある行。 *n*=0 の場合は現在行。
afterLast[1]	結果表の最後の行の後
beforeFirst[1]	結果表の最初の行の前

注:

1. この方法は、 IBM Informix Dynamic Server への接続には適用されません。
2. カーソルが結果表の最後の行の後にあれば、このメソッドによってカーソルはその最後の行に配置されます。
3. *n* の絶対値が結果表の行数より大きい場合、*n* が正であれば、このメソッドによってカーソルは最後の行の後に配置され、*n* が負であれば最初の行の前に配置されます。
4. *m* が結果表に含まれる行数、*x* が結果表内の現在行の番号とします。 *n*>0 であり、*x*+*n*>*m* である場合、イテレーターは最後の行の後ろに置かれます。 *n*<0 であり、*x*+*n*<1 である場合、イテレーターは最初の行の前に置かれます。

   b. 現行カーソル位置を知る必要がある場合には、getRow、isFirst、isLast、isBeforeFirst、または isAfterLast メソッドを使用してこの情報を取得します。

      現在のフェッチ方向を確認する必要がある場合は、getFetchDirection メソッドを呼び出します。

   c. accessor メソッドを使用して、結果表の現在行を取得します。

   d. イテレーターまたは他の手段による更新操作または削除操作が結果表の中で可視になっている場合は、getWarnings メソッドを呼び出して、現在行がホールになっているかどうかをチェックします。

   位置指定イテレーターの場合は、以下の手順を実行します。

   a. フェッチ・オリエンテーション節を指定した FETCH ステートメントを使用して、イテレーターの位置を指定し、結果表の現在行を取得します。カーソルの位置を指定するために使用できる節を 182 ページの表 31 にまとめます。

表 31. 両方向スクロール・カーソルの位置を指定するための FETCH 節

メソッド	カーソルの位置
FIRST[1]	結果表の最初の行
LAST[1]	結果表の最後の行
PRIOR[1,2]	結果表の前の行
NEXT	結果表の次の行
ABSOLUTE(*n*)[1,3]	*n*>0 の場合は、結果表の行 *n*。 *n*<0 の場合は、結果表の行 *m+n+1* (*m* は結果表の行数)。
RELATIVE(*n*)[1,4]	*n*>0 の場合、現在行の *n* 行後ろにある行。 *n*<0 の場合、現在行の *n* 行前にある行。 *n*=0 の場合は現在行。
AFTER[1,5]	結果表の最後の行の後
BEFORE[1,5]	結果表の最初の行の前

注:

1. この値は、 IBM Informix Dynamic Server との接続ではサポートされていません。
2. カーソルが結果表の最後の行の後にあれば、このメソッドによってカーソルはその最後の行に配置されます。
3. *n* の絶対値が結果表の行数より大きい場合、*n* が正であれば、このメソッドによってカーソルは最後の行の後に配置され、*n* が負であれば最初の行の前に配置されます。
4. *m* が結果表に含まれる行数、*x* が結果表内の現在行の番号とします。 *n*>0 であり、*x+n*>*m* である場合、イテレーターは最後の行の後ろに置かれます。 *n*<0 であり、*x+n*<1 である場合、イテレーターは最初の行の前に置かれます。
5. ホスト式には値が割り当てられません。

b. イテレーターまたは他の手段による更新操作または削除操作が結果表の中で可視になっている場合は、getWarnings メソッドを呼び出して、現在行がホールになっているかどうかをチェックします。

5. close メソッドを呼び出してイテレーターを閉じます。

名前指定イテレーターを使用して、従業員表のすべての行から従業員番号とラストネーム (姓) を逆順で取得するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql context Ctx;            // Create connection context class Ctx
#sql  iterator ScrollIter implements sqlj.runtime.Scrollable        1
  (String EmpNo, String LastName);
{
  ...
  Ctx ctxt =
   new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
   userid,password,false);   // Create connection context object ctxt
                             // for the connection to NEWYORK
  ScrollIter scrliter;                                              2
  #sql [ctxt]
    scrliter={SELECT EMPNO, LASTNAME FROM EMPLOYEE};
  scrliter.afterLast();
  while (scrliter.previous())                                       4a
  {
    System.out.println(scrliter.EmpNo() + " "                       4c
      + scrliter.LastName());
  }
  scrliter.close();                                                 5
}
```

SQLJ アプリケーションでのストアード・プロシージャーの呼び出し

ストアード・プロシージャーを呼び出すには、SQL の CALL ステートメントが含まれている実行可能節を使用します。

ホスト ID パラメーターを指定した CALL ステートメントを実行することも可能です。リテラル・パラメーターを指定した CALL ステートメントを実行できるのは、その CALL ステートメントを実行する DB2 サーバーが CALL ステートメントの動的実行をサポートしている場合に限られます。

ストアード・プロシージャーを呼び出すための基本手順は、以下のとおりです。

1. 入力 (IN または INOUT) パラメーターに値を割り当てます。
2. ストアード・プロシージャーを呼び出します。
3. 出力 (OUT または INOUT) パラメーターを処理します。
4. ストアード・プロシージャーが複数の結果セットを返す場合は、それらの結果セットを取得します。

3 つの入力パラメーターと 3 つの出力パラメーターがあるストアード・プロシージャーを呼び出すコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
String FirstName="TOM";            // Input parameters                  1
String LastName="NARISINST";
String Address="IBM";
int CustNo;                        // Output parameters
String Mark;
String MarkErrorText;
...
#sql [myConnCtx] {CALL ADD_CUSTOMER(:IN FirstName,                       2
                            :IN LastName,
                            :IN Address,
                            :OUT CustNo,
                            :OUT Mark,
                            :OUT MarkErrorText)};
                                  // Call the stored procedure
System.out.println("Output parameters from ADD_CUSTOMER call: ");
System.out.println("Customer number for " + LastName + ": " + CustNo);  3
System.out.println(Mark);
If (MarkErrorText != null)
  System.out.println(" Error messages:" + MarkErrorText);
```

図 44. SQLJ アプリケーションでストアード・プロシージャーを呼び出す例

SQLJ アプリケーションの CALL ステートメントでの名前付きパラメーターの使用

名前付きパラメーターを使用すると、CALL ステートメント内のホスト変数名をストアード・プロシージャー定義内のパラメーター名にマップすることができます。

名前付きパラメーターを使用する場合、ストアード・プロシージャー定義内と同じ順序で CALL ステートメント内のパラメーターを指定する必要はありません。さら

に、すべてのパラメーターを CALL ステートメントで指定する必要はありません。指定されないパラメーターは、ストアード・プロシージャー定義で指定されたデフォルト値になります。

名前付きパラメーターを CALL ステートメントで使用するには、以下の手順に従います。

1. CALL ステートメントで、IN または INOUT ホスト変数に値を割り当てます。

   名前付きパラメーターはホスト変数を指し示します。名前付きパラメーターの命名および名前付きパラメーターへの割り当てに関する規則は、SQL CALL ステートメントでの名前付きパラメーターの規則に従う必要があります。 DEFAULT キーワードまたは NULL キーワードを指定することで、名前付きパラメーターにデフォルト値や NULL 値を明示的に割り当てることができます。デフォルト値が CREATE PROCEDURE ステートメントで指定されるパラメーターの場合、CALL ステートメントでこれらのパラメーターを省略することで、名前付きパラメーターに暗黙的にデフォルト値を割り当てることができます。省略されるすべてのパラメーターのデフォルト値がストアード・プロシージャー定義で指定される場合に限り、パラメーターを省略できます。

   名前付きパラメーターと無名パラメーターを同じ CALL ステートメント内で混用することはできません。
2. 出力 (OUT または INOUT) パラメーターを処理します。
3. ストアード・プロシージャーが複数の結果セットを返す場合は、それらの結果セットを取得します。

以下のコードでは、次のような定義のストアード・プロシージャーが呼び出されます。

```
CREATE PROCEDURE SALS (
  OUT retcode INTEGER,
  IN lowsal DOUBLE,
  IN medsal DOUBLE,
  IN highsal DOUBLE DEFAULT 100000,
  IN department CHAR(3) DEFAULT '---')
SPECIFIC JDBC_SALS
DYNAMIC RESULT SETS 0
DETERMINISTIC
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO DBINFO
FENCED
THREADSAFE
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'MYJAR:MyClass.sals'
```

CALL ステートメント内の入力パラメーターは名前付きパラメーターで表されます。 3 番目と 4 番目のパラメーターは、ストアード・プロシージャーのデフォルト値を使って呼び出されます。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
double hvLowSal=10000;              // Host variables for input parameters
double hvMedSal=50000;
int hvRetCode;                      // Host variable for output parameter
...
#sql [myConnCtx] {CALL SALS(retcode=>:OUT hvRetCode,            1
                    lowsal=>:IN hvLowSal,
```

```
                    medsal=>:IN hvMedSal,
                    highsal=>DEFAULT)};
                                 // Call the stored procedure.
                                 // Implicitly use the default
                                 // value for the last parameter
                                 // by omitting it.
System.out.println("Return code from SALS call: " + hvRetCode);   2
```

SQLJ アプリケーションでのカーソル出力パラメーターからのデータの取得

DB2 Database for Linux, UNIX, and Windows ストアード・プロシージャーでは、カーソル・タイプの OUT パラメーターを使用できます。 SQLJ アプリケーションでこれらのパラメーターからデータを取得するには、イテレーターまたは ResultSet オブジェクトを使用します。

カーソル変数からデータを取得するには、以下の手順に従います。

1. ストアード・プロシージャー定義の中で CURSOR データ・タイプを持つ OUT パラメーターごとにイテレーターまたは ResultSet オブジェクトを定義します。

   カーソル OUT パラメーターを取得するためのイテレーターは、名前付きまたは位置指定にすることができます。
2. 入力パラメーターに値を割り当てます。
3. ストアード・プロシージャーを呼び出します。
4. カーソル・パラメーターから行を取得します。
   - カーソル・パラメーター用に位置指定イテレーターを宣言した場合、FETCH ステートメントを使ってデータを取得します。
   - カーソル・パラメーター用に名前付きイテレーターを宣言した場合、NamedIterator メソッドを使ってデータを取得します。
   - カーソル・パラメーター用に ResultSet オブジェクトを定義した場合、ResultSet メソッドを使ってカーソルを位置指定し、結果セット行から値を取得します。
5. WITH RETURN として定義されているカーソルをオープンすることにより複数の結果セットがストアード・プロシージャーによって戻される場合は、これらの結果セットを取得してください。

   1 つのストアード・プロシージャーが、CURSOR パラメーターだけでなく複数の結果セットを介してデータを戻すことがあります。

カーソル・データ・タイプおよびストアード・プロシージャーの定義は次のとおりです。

```
CREATE TYPE myRowType AS ROW (name VARCHAR(128))
CREATE TYPE myCursorType AS myRowType CURSOR
CREATE PROCEDURE MYPROC(IN pempNo VARCHAR(6), OUT pcv1 myCursorType)
  RESULT SETS 0
  LANGUAGE SQL
  BEGIN
     SET pcv1 = CURSOR FOR SELECT name FROM employee WHERE empNo = pempNo;
     OPEN pcv1;
  END
```

以下のコードでは、ストアード・プロシージャー MYPROC を呼び出し、位置指定イテレーターを使ってカーソル pcv1 からデータを取得します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
#sql iterator Iter (String);         // Declare a positioned iterator
...
Iter iter = null;                    // Output parameter                1
String hvPempNo="000500";            // Input parameter                 2
#sql [ctx] {CALL MYPROC (:IN hvPempNo, :OUT iter)};                      3
                                     // Call the stored procedure
String hvEmpName = null;
while (true) {                       // Retrieve rows from the result set
  #sql { FETCH :iter into :hvName };                                     4
  if (iter.endFetch()) break;
  System.out.println("Employee name for " + hvPempNo
    + ": " + hvEmpName);
}
```

以下のコードでは、ストアード・プロシージャー MYPROC を呼び出し、ResultSet オブジェクトを使用してカーソル pcv1 からデータを取得します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
...
ResultSet rs = null;                 // Output parameter                1
String hvPempNo="000500";            // Input parameter                 2
#sql [ctx] {CALL MYPROC (:IN hvPempNo, :OUT rs)};                        3
                                     // Call the stored procedure
String hvEmpName = null;
while (rs.next()) {                  // Retrieve result set rows        4
  hvEmpName=rs.getString(1);
  System.out.println("Employee name for " + hvPempNo
    + ": " + hvEmpName);
}
```

SQLJ アプリケーションでのストアード・プロシージャーの複数の結果セットの検索

いくつかのストアード・プロシージャーは、定義に DYNAMIC RESULT SETS *n* 節 (*n*>0) を組み込み、WITH RETURN 節で定義されているカーソルをオープンすることにより、1 つ以上の結果セットを呼び出し側プログラムに戻します。呼び出し側プログラムはそれらの結果セットの内容を検索する必要があります。

それらの結果セットから行を取得するには、以下のステップを実行します。

1. ストアード・プロシージャーの結果セットを取得するために実行コンテキストを取得します。
2. 実行コンテキストをストアード・プロシージャーの CALL ステートメントと関連付けます。

   最後の結果セットを取得して処理するまで、この実行コンテキストを他の目的では使用しないでください。
3. 結果セットごとに、次のことを実行します。
   a. ExecutionContext メソッド getNextResultSet を使用して結果セットを取得します。
   b. 結果セットの内容がわからない場合、ResultSetMetaData メソッドを使用してこの情報を取得します。

c. SQLJ 結果セット・イテレーターまたは JDBC ResultSet を使用して、結果セットから行を取得します。

結果セットは、カーソルがストアード・プロシージャーでオープンされるのと同じ順序で呼び出し側プログラムに戻されます。検索する結果セットがなくなると、getNextResultSet は NULL 値を戻します。

getNextResultSet には次の 2 つの形式があります。

```
getNextResultSet();
getNextResultSet(int current);
```

最初の形式である getNextResultSet を呼び出すと、SQLJ は現在オープンしている結果セットをクローズし、次の結果セットに進みます。 2 番目の形式である getNextResultSet を呼び出すと、SQLJ が次の結果セットに進む前に現在オープンしている結果セットで行う事柄を、*current* の値で指示します。

java.sql.Statement.CLOSE_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現在の ResultSet オブジェクトをクローズすることを指定します。

java.sql.Statement.KEEP_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現在の ResultSet オブジェクトをオープンしたままにすることを指定します。

java.sql.Statement.CLOSE_ALL_RESULTS
次の ResultSet オブジェクトが戻されるときに、オープンしているすべての ResultSet オブジェクトをクローズすることを指定します。

以下のコードは、複数の結果セットを戻すストアード・プロシージャーを呼び出します。この例の場合、戻される結果セットの数またはそれらの結果セットの内容を呼び出し元が認識していないことを想定しています。また、autoCommit が false であることも想定しています。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();      1
#sql [myConnCtx, execCtx] {CALL MULTRSSP()};                   2
            // MULTRSSP returns multiple result sets
ResultSet rs;
while ((rs = execCtx.getNextResultSet()) != null)              3a
{
  ResultSetMetaData rsmeta=rs.getMetaData();                   3b
  int numcols=rsmeta.getColumnCount();
  while (rs.next())                                            3c
  {
    for (int i=1; i<=numcols; i++)
    {
      String colval=rs.getString(i);
      System.out.println("Column " + i + "value is " + colval);
    }
  }
}
```

図 45. ストアード・プロシージャーの結果セットの検索

IBM Data Server Driver for JDBC and SQLJ 使用時の SQLJ アプリケーションでの LOB

IBM Data Server Driver for JDBC and SQLJを使用すると、LOB データの Clob または Blob ホスト式への取り出しや、 Clob または Blob ホスト式で CLOB、BLOB、 DBCLOB 列の更新ができます。また、Clob または Blob データ・タイプのイテレーターを宣言して、 CLOB、BLOB、または DBCLOB 列のデータを検索することができます。

LOB データの検索または更新: BLOB 列のデータを検索するには、データ・タイプ Blob または byte[] を含むイテレーターを宣言します。 CLOB または DBCLOB 列のデータを検索するには、対応する列に Clob データ・タイプがあるイテレーターを宣言します。

BLOB 列のデータを更新するには、データ・タイプが Blob のホスト式を使用します。 CLOB 列または DBCLOB 列のデータを更新するには、データ・タイプが Clob のホスト式を使用します。

連続ストリーミングまたは LOB ロケーター: SQLJ アプリケーションでは、連続ストリーミング (動的データ形式ともいう) または LOB ロケーターを、JDBC アプリケーションで使用するのと同じ方法で使用できます。

SQLJ アプリケーションでの LOB 列データの検索および更新のための Java データ・タイプ

deferPrepares プロパティーが true に設定されており、 IBM Data Server Driver for JDBC and SQLJ がホスト式を含むカスタマイズされていない SQLJ ステートメントを処理する場合、ドライバーはデータ・タイプを判別するための追加の処理を行う必要があるかもしれません。この追加処理は、パフォーマンスに影響を与えるおそれがあります。

JDBC ドライバーで、LOB 列とともに使用されるパラメーターのデータ・タイプがすぐに判別できない場合は、LOB データ・タイプと互換性のあるパラメーターのデータ・タイプを選択する必要があります。

BLOB 列の入力パラメーター

BLOB 列の入力パラメーターの場合、以下の手法のいずれかを使用できます。

- 次のように、BLOB 列と完全に一致する java.sql.Blob 入力変数を使用します。

  ```
  java.sql.Blob blobData;
  #sql {CALL STORPROC(:IN blobData)};
  ```

  java.sql.Blob 入力変数を使用するには、その前に java.sql.Blob オブジェクトを作成し、それからそのオブジェクトにデータを入力する必要があります。
- タイプ sqlj.runtime.BinaryStream の入力パラメーターを使用します。 sqlj.runtime.BinaryStream オブジェクトは BLOB データ・タイプと互換性があります。以下に例を示します。

```
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(byteData);
int numBytes = byteData.length;
sqlj.runtime.BinaryStream binStream =
  new sqlj.runtime.BinaryStream(byteStream, numBytes);
#sql {CALL STORPROC(:IN binStream)};
```

この手法は INOUT パラメーターには使用できません。

BLOB 列の出力パラメーター

BLOB 列の出力パラメーターまたは INOUT パラメーターの場合、以下の手法を使用できます。

- 出力パラメーターまたは INOUT 変数を java.sql.Blob データ・タイプで宣言します。

```
java.sql.Blob blobData = null;
#sql CALL STORPROC (:OUT blobData)};
```

```
java.sql.Blob blobData = null;
#sql CALL STORPROC (:INOUT blobData)};
```

CLOB 列の入力パラメーター

CLOB 列の入力パラメーターの場合、以下の手法のいずれかを使用できます。

- 次のように、CLOB 列と完全に一致する java.sql.Clob 入力変数を使用します。

```
#sql CALL STORPROC(:IN clobData)};
```

  java.sql.Clob 入力変数を使用するには、その前に java.sql.Clob オブジェクトを作成し、それからそのオブジェクトにデータを入力する必要があります。

- 以下のいずれかのタイプのストリーム IN パラメーターを使用します。
  - sqlj.runtime.CharacterStream 入力パラメーター:

```
java.lang.String charData;
java.io.StringReader reader = new java.io.StringReader(charData);
sqlj.runtime.CharacterStream charStream =
  new sqlj.runtime.CharacterStream (reader, charData.length);
#sql {CALL STORPROC(:IN charStream)};
```

  - Unicode UTF-16 データ用の sqlj.runtime.UnicodeStream パラメーター:

```
byte[] charDataBytes = charData.getBytes("UnicodeBigUnmarked");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream(charDataBytes);
sqlj.runtime.UnicodeStream uniStream =
  new sqlj.runtime.UnicodeStream(byteStream, charDataBytes.length );
#sql {CALL STORPROC(:IN uniStream)};
```

  - ASCII データ用の sqlj.runtime.AsciiStream パラメーター:

```
byte[] charDataBytes = charData.getBytes("US-ASCII");
java.io.ByteArrayInputStream byteStream =
  new java.io.ByteArrayInputStream (charDataBytes);
sqlj.runtime.AsciiStream asciiStream =
  new sqlj.runtime.AsciiStream (byteStream, charDataBytes.length);
#sql {CALL STORPROC(:IN asciiStream)};
```

  これらの呼び出しでは、入力データの正確な長さを指定する必要があります。この手法は INOUT パラメーターには使用できません。

- java.lang.String 入力パラメーターを使用します。

```
java.lang.String charData;
#sql {CALL STORPROC(:IN charData)};
```

CLOB 列の出力パラメーター

CLOB 列の出力パラメーターまたは INOUT パラメーターの場合、以下の手法のいずれかを使用できます。

- java.sql.Clob 出力変数を使用します。これは、CLOB 列に完全一致します。

```
java.sql.Clob clobData = null;
#sql CALL STORPROC(:OUT clobData)};
```

- java.lang.String 出力変数を使用します。

```
java.lang.String charData = null;
#sql CALL STORPROC(:OUT charData)};
```

この技法は、取り出されるデータの長さが 32 KB 以下であることが分かっている場合にのみ使用してください。 32 KB を超える場合、データは切り捨てられます。

DBCLOB 列の出力パラメーター

ストアード・プロシージャー用の DBCLOB 出力パラメーターまたは INOUT パラメーターはサポートされていません。

同じアプリケーションでの SQLJ および JDBC

SQLJ 節と JDBC 呼び出しを単一プログラムで結合できます。

これを効率的に行うには、以下の事柄を行えなければなりません。

- JDBC Connection を使用して SQLJ ConnectionContext を作成するか、JDBC Connection を SQLJ ConnectionContext から取得します。
- SQLJ イテレーターを使用して JDBC ResultSet からデータを取り出すか、JDBC ResultSet を SQLJ イテレーターから生成します。

JDBC Connection からの SQLJ ConnectionContext の作成: これを行うには、以下のようにします。

1. SQLJ 接続宣言節を実行して、ConnectionContext クラスを作成します。
2. ドライバーをロードするか、または DataSource インスタンスを取得します。
3. SQLJ DriverManager.getConnection または DataSource.getConnection メソッドを呼び出して、JDBC Connection を取得します。
4. 引数として Connection を使用して ConnectionContext コンストラクターを呼び出して、ConnectionContext オブジェクトを作成します。

SQLJ ConnectionContext からの JDBC Connection の取得: これを行うには、以下のようにします。

1. SQLJ 接続宣言節を実行して、ConnectionContext クラスを作成します。
2. ドライバーをロードするか、または DataSource インスタンスを取得します。
3. 引数としてドライバーの URL およびその他の必要なパラメーターを使用して ConnectionContext コンストラクターを呼び出して、ConnectionContext オブジェクトを作成します。

4. JDBC ConnectionContext.getConnection メソッドを呼び出して、JDBC Connection オブジェクトを作成します。

SQLJ 接続について詳しくは、『SQLJ を使用したデータ・ソースへの接続』を参照してください。

SQLJ イテレーターを使用した JDBC 結果セットの検索: *iterator conversion statement* を使用して、JDBC 結果セットを SQLJ イテレーターとして操作します。イテレーター変換ステートメントの一般的な形式は、次のとおりです。

```
#sql iterator={CAST :result-set};
```

結果セットをイテレーターに正常にキャストするには、その前にイテレーターが以下の規則に準拠している必要があります。

- イテレーターは public と宣言される必要があります。
- イテレーターが位置指定イテレーターの場合、結果セット内の列の数がイテレーター内の列の数に一致しなければなりません。さらに、結果セット内の列ごとのデータ・タイプがイテレーター内の対応する列のデータ・タイプに一致する必要があります。
- イテレーターが名前指定イテレーターの場合、各 accessor メソッドの名前が結果セット内の列の名前に一致しなければなりません。さらに、accessor メソッドが戻すオブジェクトのデータ・タイプが結果セット内の対応する列のデータ・タイプに一致する必要があります。

図 46 のコードは、JDBC 呼び出しを使用して照会を作成および実行し、iterator conversion ステートメントを実行して JDBC 結果セットを SQLJ イテレーターに変換し、イテレーターを使用して結果表から行を取得します。

```
#sql public iterator ByName(String LastName, Date HireDate);      1
public void HireDates(ConnectionContext connCtx, String whereClause)
{

  ByName nameiter;           // Declare object of ByName class
  Connection conn=connCtx.getConnection();
                             // Create JDBC connection
  Statement stmt = conn.createStatement();                         2
  String query = "SELECT LASTNAME, HIREDATE FROM EMPLOYEE";
  query+=whereClause;   // Build the query
  ResultSet rs = stmt.executeQuery(query);                         3
  #sql [connCtx] nameiter = {CAST :rs};                            4
  while (nameiter.next())
  {
    System.out.println( nameiter.LastName() + " was hired on "
      + nameiter.HireDate());

  }
  nameiter.close();                                                5
  stmt.close();
}
```

図 46. SQLJ イテレーターへの JDBC 結果セットの変換

191 ページの図 46 の注:

注	説明
1	この SQLJ 節は、名前指定イテレーター・クラス ByName を作成します。このクラスには、結果表列 LASTNAME および HIREDATE からのデータを戻す accessor メソッド LastName() および HireDate() があります。
2	このステートメントおよび次の 2 つのステートメントは、JDBC を使用して動的実行のために照会を作成し、準備します。
3	この JDBC ステートメントは SELECT ステートメントを実行し、その結果表を結果セット rs に割り当てます。
4	このイテレーター変換節は JDBC ResultSet rs を SQLJ イテレーター nameiter に変換し、次のステートメントは nameiter を使用して結果表から値を検索します。
5	nameiter.close() メソッドは、SQLJ イテレーターおよび JDBC ResultSet rs をクローズします。

SQLJ イテレーターからの JDBC ResultSet の生成: getResultSet メソッドを使用して、SQLJ イテレーターから JDBC ResultSet を生成します。すべての SQLJ イテレーターには getResultSet メソッドがあります。イテレーターの基礎となっている ResultSet にアクセスした後は、ResultSet だけを使用して行をフェッチする必要があります。

図 47 のコードは、照会用の位置指定イテレーターを生成し、そのイテレーターを結果セットに変換し、さらに JDBC メソッドを使用して表から行をフェッチします。

```
#sql iterator EmpIter(String, java.sql.Date);
{
...
  EmpIter iter=null;
  #sql [connCtx] iter=
    {SELECT LASTNAME, HIREDATE FROM EMPLOYEE};                1
  ResultSet rs=iter.getResultSet();                           2
  while (rs.next())                                           3
  { System.out.println(rs.getString(1) + " was hired in " +
      rs.getDate(2));
  }
  rs.close();                                                 4
}
```

図 47. JDBC ResultSet への SQLJ イテレーターの変換

図 47 の注:

注	説明
1	この SQLJ 節は SELECT ステートメントを実行し、その SELECT ステートメントの結果表を含むイテレーター・オブジェクトを構成し、そのイテレーター・オブジェクトを変数 iter に割り当てます。
2	getResultSet() メソッドは、イテレーター iter の基礎となっている ResultSet にアクセスします。
3	JDBC getString() および getDate() メソッドは ResultSet から値を検索します。next() メソッドは ResultSet 内の次の行にカーソルを移動します。
4	rs.close() メソッドは、ResultSet と同時に SQLJ イテレーターをクローズします。

SQLJ アプリケーションでの JDBC ResultSet の使用の規則および制限: JDBC 結果セットを含む SQLJ アプリケーションを作成する場合、以下の規則および制限を守ってください。

- ResultSet とイテレーターが異なる保持可能性属性を持つ場合、ResultSet を SQLJ イテレーターにキャストできません。

  COMMIT 操作後も JDBC ResultSet または SQLJ イテレーターをオープンしたままにすることができます。 JDBC ResultSet の場合、この特性は IBM Data Server Driver for JDBC and SQLJ プロパティー `resultSetHoldability` によって制御されます。 SQLJ イテレーターの場合、この特性はイテレーター宣言の `with holdability` パラメーターによって制御されます。保持可能性を持つ ResultSet を保持可能性を持たない SQLJ イテレーターにキャストしたり、保持可能性を持たない ResultSet を保持可能性を持つ SQLJ イテレーターにキャストしたりすることはサポートされていません。
- プログラムがイテレーターまたは ResultSet を使用しなくなり次第、さらにはプログラムを終了する前に、イテレーターまたは基礎となっている ResultSet オブジェクトをクローズします。

  イテレーターをクローズすると、ResultSet オブジェクトもクローズされます。ResultSet オブジェクトをクローズすると、イテレーター・オブジェクトもクローズされます。一般に、最後に使用されるオブジェクトをクローズするのが最善です。
- スクロール可能イテレーターおよびスクロール可能で更新可能な ResultSet オブジェクトをサポートする IBM Data Server Driver for JDBC and SQLJ の場合、以下の制限が適用されます。
  - スクロール可能イテレーターには、その基礎となる JDBC ResultSet オブジェクトと同じ制限があります。
  - 更新可能な SQLJ イテレーターに、更新できない JDBC ResultSet をキャストすることはできません。

SQLJ での SQL ステートメントの実行の制御

SQL ステートメントの実行を制御したりモニターしたりするには、SQLJ の ExecutionContext クラスから特定のメソッドを選択して使用します。

ExecutionContext メソッドを使用するには、以下の手順を実行します。

1. 接続コンテキストからデフォルトの実行コンテキストを取得します。

   実行コンテキストを取得するには、2 つの方法があります。
   - 接続コンテキストからデフォルトの実行コンテキストを取得します。以下に例を示します。

     ```
     ExecutionContext execCtx = connCtx.getExecutionContext();
     ```
   - ExecutionContext のコンストラクターを呼び出して、新しい実行コンテキストを作成します。以下に例を示します。

     ```
     ExecutionContext execCtx=new ExecutionContext();
     ```
2. 実行コンテキストを SQL ステートメントに関連付けます。

   そのためには、SQL ステートメントを組み込んだ実行節の接続コンテキストの後に実行コンテキストを指定します。
3. ExecutionContext メソッドを呼び出します。

ExecutionContext メソッドの中には、関連する SQL ステートメントの実行前に適用できるものもあれば、関連する SQL ステートメントの実行後にしか適用できないものもあります。

例えば、getUpdateCount メソッドは、DELETE ステートメントの実行後に、その DELETE ステートメントによって削除された行の数をカウントするために使用できます。

実行コンテキストを取得してから、その実行コンテキストで getUpdateCount メソッドを使用して、DELETE ステートメントによって削除された行の数を確認するためのコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
ExecutionContext execCtx=new ExecutionContext();                          1
#sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};      2
System.out.println("Deleted " + execCtx.getUpdateCount() + " rows");      3
```

IBM Data Server Driver for JDBC and SQLJ 使用時の SQLJ での ROWID

DB2 for z/OSおよび DB2 for i は、表内の列に対して ROWID データ・タイプをサポートします。ROWID とは、表内の行を一意的に識別する値です。

IBM Informix Dynamic Server (IDS) では ROWID もサポートされますが、それらの ROWID のデータ・タイプは INTEGER です。 4 バイトの整数データ・タイプを使用して IDS ROWID 列を変数に選択することができます。

SQLJ プログラムで ROWID データ・タイプの列を使用する場合、それらのプログラムをカスタマイズする必要があります。

JDBC 4.0 には、イテレーターおよび CALL ステートメント・パラメーターで使用できるインターフェース java.sql.RowId が含まれています。 JDBC 4.0 がない場合、 IBM Data Server Driver for JDBC and SQLJ 専用のクラス com.ibm.db2.jcc.DB2RowID を使用できます。イテレーターの場合、byte[] オブジェクト・タイプを使用して ROWID 値を検索することもできます。

以下のコーディング例は、ROWID 列から値を選択するために使用するイテレーターを示したものです。

```
#sql iterator PosIter(int,String,java.sql.RowId);
                                // Declare positioned iterator
                                // for retrieving ITEM_ID (INTEGER),
                                // ITEM_FORMAT (VARCHAR), and ITEM_ROWID (ROWID)
                                // values from table ROWIDTAB
{
  PosIter positrowid;           // Declare object of PosIter class
  java.sql.RowId rowid = null;
  int id = 0;
  String i_fmt = null;
                                // Declare host expressions
  #sql [ctxt] positrowid =
    {SELECT ITEM_ID, ITEM_FORMAT, ITEM_ROWID FROM ROWIDTAB
       WHERE ITEM_ID=3};
                                // Assign the result table of the SELECT
                                // to iterator object positrowid
  #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Retrieve the first row
  while (!positrowid.endFetch())
                                // Check whether the FETCH returned a row
    {System.out.println("Item ID " + id + " Item format " +
       i_fmt + " Item ROWID ");
       MyUtilities.printBytes(rowid.getBytes());
                           // Use the getBytes method to
                           // convert the value to bytes for printing.
                           // Call a user-defined method called
                           // printBytes (not shown) to print
                           // the value.
       #sql {FETCH :positrowid INTO :id, :i_fmt, :rowid};
                                // Retrieve the next row
    }
  positrowid.close();           // Close the iterator
}
```

図 48. イテレーターを使用した ROWID 値の検索の例

以下のコーディング例は、3 つの ROWID パラメーター、すなわち IN パラメーター、OUT パラメーター、および INOUT パラメーターを含むストアード・プロシージャーの呼び出しを示したものです。

```
java.sql.RowId in_rowid = rowid;
java.sqlRowId out_rowid = null;
java.sql.RowId inout_rowid = rowid;
                                   // Declare an IN, OUT, and
                                   // INOUT ROWID parameter
...
#sql [myConnCtx] {CALL SP_ROWID(:IN in_rowid,
                             :OUT out_rowid,
                             :INOUT inout_rowid)};
                                   // Call the stored procedure
System.out.println("Parameter values from SP_ROWID call: ");
System.out.println("OUT parameter value ");
MyUtilities.printBytes(out_rowid.getBytes());
                           // Use the getBytes method to
                           // convert the value to bytes for printing
                           // Call a user-defined method called
                           // printBytes (not shown) to print
                           // the value.
System.out.println("INOUT parameter value ");
MyUtilities.printBytes(inout_rowid.getBytes());
```

図 49. ROWID パラメーターを指定してストアード・プロシージャーを呼び出す例

SQLJ アプリケーションにおける特殊タイプ

SQLJ プログラムでは、特殊タイプは実行可能節で CREATE DISTINCT TYPE ステートメントを使用して作成することができます。

また、実行可能節で CREATE TABLE を使用して、このタイプの列を含む表を作成することもできます。このタイプの列からデータを取得するか、またはこのタイプの列を更新する場合は、特殊タイプのベースとなる組み込みタイプに対応するデータ・タイプと共に、Java ホスト変数または式を使用します。

以下の例では、INTEGER タイプをベースにした特殊タイプを作成し、このタイプの列を持つ表を作成して、行をその表に挿入し、さらにその表から行を取得します。

```
String empNumVar;
int shoeSizeVar;
...
#sql [myConnCtx] {CREATE DISTINCT TYPE SHOESIZE AS INTEGER WITH COMPARISONS};
                                          // Create distinct type
#sql [myConnCtx] {COMMIT}; // Commit the create
#sql [myConnCtx] {CREATE TABLE EMP_SHOE
  (EMPNO CHAR(6), EMP_SHOE_SIZE SHOESIZE)};
                                          // Create table using distinct type
#sql [myConnCtx] {COMMIT}; // Commit the create
#sql [myConnCtx] {INSERT INTO EMP_SHOE
  VALUES('000010',6)};                    // Insert a row in the table
#sql [myConnCtx] {COMMIT}; // Commit the INSERT
#sql [myConnCtx] {SELECT EMPNO, EMP_SHOE_SIZE
  INTO :empNumVar, :shoeSizeVar
  FROM EMP_SHOE};                         // Retrieve the row
System.out.println("Employee number: " + empNumVar +
  " Shoe size: " + shoeSizeVar);
```

図 50. 特殊タイプの定義および使用

ARRAY パラメーターを持つストアード・プロシージャーの、SQLJ アプリケーションでの呼び出し

IBM Data Server Driver for JDBC and SQLJ で実行され、 DB2 Database for Linux, UNIX, and Windows データ・ソースに接続する SQLJ アプリケーションは、ARRAY パラメーターを持つストアード・プロシージャーを呼び出すことができます。

java.sql.Array オブジェクトは、ストアード・プロシージャーで IN、OUT、または INOUT パラメーターとして使用できます。

IN または INOUT パラメーターの場合、DB2Connection.createArrayOf メソッド (JDBC 3.0 またはそれ以前) または Connection.createArrayOf メソッド (JDBC 4.0 またはそれ以降) を使用して、java.sql.Array オブジェクトを作成します。

ARRAY 出力ストアード・プロシージャー・パラメーターからデータを取り出すには、次の 2 つの方法があります。

- java.sql.Array.getArray メソッドを使用して出力パラメーターの内容を Java 配列に取り出します。

- java.sql.Array.getResultSet メソッドを使用して出力パラメーター・データを ResultSet オブジェクトに取り出します。次に ResultSet メソッドを使用して配列のエレメントを取り出します。ResultSet の各行には、次の 2 つの列が含まれています。
  - 配列の指標 (1 から始まります)
  - 配列エレメント

  配列エレメントは、getObject メソッドで ResultSet から取り出す必要があります。

例: ストアード・プロシージャー GET_EMP_DATA の入力パラメーター IN_PHONE と出力パラメーター OUT_PHONE が以下のように定義される配列だとします。

```
CREATE TYPE PHONENUMBERS AS VARCHAR(10) ARRAY[5]
```

GET_EMP_DATA をこの 2 つのパラメーターで呼び出します。

```
Connection con;
String type = "CHAR";
String [] contents = {"1234", "5678", "9101"};
...
com.ibm.db2.jcc.DB2Connection db2con = (com.ibm.db2.jcc.DB2Connection) con;
                                    // Cast the Connection as a DB2Connection
                                    // so you can use the
                                    // DB2Connection.createArrayOf method
java.sql.Array inPhoneData = db2con.createArrayOf(type, contents);
java.sql.Array outPhoneData;
try {
  #sql [db2con] {CALL GET_EMP_DATA(:IN inPhoneData, :OUT outPhoneData ) };
}
catch( SQLException e )
{
  throw e;
}
ResultSet rs = outPhoneData.getResultSet();
while (rs.next()) {
  String phoneNum = (String)rs.getObject(2); // Get phone number
  System.out.println("Phone number = " + phoneNum);
}
```

SQLJ アプリケーションにおけるセーブポイント

IBM Data Server Driver for JDBC and SQLJ では、任意の形式の SQL SAVEPOINT ステートメントを SQLJ プログラムに組み込むことができます。

SQL セーブポイントは、作業単位内の特定の時点のデータおよびスキーマの状態を表します。セーブポイントを設定したり、セーブポイントを解除したり、データおよびスキーマをセーブポイントが表す状態へリストアするための SQL ステートメントがあります。

以下の例は、セーブポイントの設定、セーブポイントへのロールバック、およびセーブポイントの解除の方法を示しています。

図 51. SQLJ アプリケーションでのセーブポイントの設定、ロールバック、および解除

```
#sql context Ctx;             // Create connection context class Ctx
String empNumVar;
int shoeSizeVar;
...
try {                         // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                              // Create JDBC connection object jdbccon
jdbccon.setAutoCommit(false); // Do not autocommit
Ctx ctxt=new Ctx(jdbccon);
                              // Create connection context object myConnCtx
                              // for the connection to NEWYORK
...                             // Perform some SQL
#sql [ctxt] {COMMIT};         // Commit the transaction
                              // Commit the create
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000010', 6)};
                              // Insert a row
#sql [ctxt]
  {SAVEPOINT SVPT1 ON ROLLBACK RETAIN CURSORS};
                              // Create a savepoint
...
#sql [ctxt]
  {INSERT INTO EMP_SHOE VALUES ('000020', 10)};
                               // Insert another row
#sql [ctxt] {ROLLBACK TO SAVEPOINT SVPT1};
                               // Roll back work to the point
                               //  after the first insert
...
#sql [ctxt] {RELEASE SAVEPOINT SVPT1};
                               // Release the savepoint
ctx.close();                   // Close the connection context
```

SQLJ アプリケーションでの XML データ

SQLJ アプリケーションでは、XML 列へのデータの保管、および XML 列からのデータの取り出しが可能です。

DB2 表では、XML 組み込みデータ・タイプを使用して XML データをノードの構造化セットとしてツリー形式で列に保管します。

アプリケーションでは、XML データは直列化されたストリング形式になります。

DB2 Database for Linux, UNIX, and Windows に接続する SQLJ アプリケーションでは、XML データはテキスト XML 形式になります。 DB2 for z/OS に接続する SQLJ アプリケーションでは、XML データはテキスト XML 形式またはバイナリー XML 形式が可能です。

SQLJ アプリケーションでは以下を行うことができます。

- INSERT、UPDATE、または MERGE ステートメントを使用して XML 文書全体を XML 列に保管します。
- 単一行の SELECT ステートメントまたはイテレーターを使用して XML 文書全体を XML 列から取得します。

- SQL XMLQUERY 関数を使用してシーケンスをデータベース内に取り出してから、単一行の SELECT ステートメントまたはイテレーターを使用して、シリアライズされた XML ストリング・データをアプリケーション変数に取り出すことによって、XML 列の文書からシーケンスを取り出します。
- ストリング 'XQUERY' が前に付加されている XQuery 式を使用して、シーケンスのエレメントをデータベース内の結果表に取り出すことによって、XML 列の文書からシーケンスを取り出します。結果表の各行はシーケンス内のアイテムを表します。続いて、単一行の SELECT ステートメントまたはイテレーターを使用してデータをアプリケーション変数に取り出します。
- SQL XMLTABLE 関数を使用して結果表を定義し、その表を取り出すことによって、XML 列の文書からユーザー定義表としてシーケンスを取り出します。続いて、単一行の SELECT ステートメントまたはイテレーターを使用して、データを結果表からアプリケーション変数に取り出します。

JDBC 4.0 の `java.sql.SQLXML` オブジェクトを使用して、XML 列内のデータの取り出しおよび更新を行うことができます。 ResultSetMetaData.getColumnTypeName などのメタデータ・メソッドを呼び出すと、XML 列タイプの整数値 `java.sql.Types.SQLXML` が戻されます。

SQLJ アプリケーションでの XML 列の更新

データベース表の XML 列についてデータを更新または挿入する場合は、SQLJ アプリケーション内の入力データは直列化されたストリング形式である必要があります。

XML 列の更新に使用できるホスト式のデータ・タイプは以下のとおりです。
- java.sql.SQLXML (SDK for Java バージョン 6 以降、および IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降が必要)
- com.ibm.db2.jcc.DB2Xml (非推奨)
- String
- byte
- Blob
- Clob
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

XML データのエンコード方式は、データ自体から導出する (*内部的にエンコードされた データという*) か、または外部ソースから導き出される (*外部的にエンコードされた データという*) ことができます。データベース・サーバーにバイナリー・データとして送信される XML データは、内部的にエンコードされたデータとして処理されます。データ・ソースに文字データとして送信される XML データは、外部的にエンコードされたデータとして処理されます。外部エンコード方式は JVM のデフォルトのエンコード方式です。

Java アプリケーションの外部エンコード方式は、常に Unicode エンコード方式です。

外部的にエンコードされたデータに、内部エンコード方式を含めることができます。つまり、このデータはデータ・ソースに文字データとして送信されますが、こ

のデータにエンコード方式の情報を含めることができます。データ・ソースにより、以下のように内部エンコード方式と外部エンコード方式の間の非互換性が処理されます。

- データ・ソースが DB2 Database for Linux, UNIX, and Windows の場合、外部エンコード方式と内部エンコード方式の間に互換性がなく、外部エンコード方式と内部エンコード方式が Unicode でない場合は、データ・ソースによりエラーが生成されます。外部エンコード方式と内部エンコード方式が Unicode の場合は、データ・ソースでは内部エンコード方式が無視されます。
- データ・ソースが DB2 for z/OS の場合は、データ・ソースでは内部エンコード方式が無視されます。

XML 列のデータは UTF-8 エンコード方式で保管されます。

例: 次のステートメントを使用して、データを String ホスト式 xmlString から表の XML 列に挿入するとします。 xmlString は文字タイプのため、内部エンコード方式が指定されているかどうかに関係なく、外部エンコード方式が使用されます。

```
#sql [ctx] {INSERT INTO CUSTACC VALUES (1, :xmlString)};
```

例: データを xmlString から CP500 エンコード方式のバイト配列にコピーするとします。データには、CP500 のエンコード方式宣言を持つ XML 宣言が含まれています。次に、byte[] ホスト式から表の XML 列にデータを挿入します。

```
byte[] xmlBytes = xmlString.getBytes("CP500");
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :xmlBytes)};
```

バイト・ストリングは、内部的にエンコードされたデータとみなされます。データはその内部コード化スキームから UTF-8 に変換され、必要な場合はデータ・ソース上に階層形式で保管されます。

例: データを xmlString から US-ASCII エンコード方式のバイト配列にコピーするとします。 sqlj.runtime.AsciiStream ホスト式を構成し、データを sqlj.runtime.AsciiStream ホスト式からデータ・ソース上にある表の XML 列に挿入します。

```
byte[] b = xmlString.getBytes("US-ASCII");
java.io.ByteArrayInputStream xmlAsciiInputStream =
  new java.io.ByteArrayInputStream(b);
sqlj.runtime.AsciiStream sqljXmlAsciiStream =
  new sqlj.runtime.AsciiStream(xmlAsciiInputStream,  b.length);
#sql[ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlAsciiStream)};
```

sqljXmlAsciiStream はストリーム・タイプであるため、内部エンコード方式が使用されます。データはその内部エンコード方式から UTF-8 エンコード方式に変換され、データ・ソース上に階層形式で保管されます。

例: sqlj.runtime.CharacterStream ホスト式: sqlj.runtime.CharacterStream ホスト式を構成し、データを sqlj.runtime.CharacterStream ホスト式から表の XML 列に挿入します。

```
java.io.StringReader xmlReader =
  new java.io.StringReader(xmlString);
sqlj.runtime.CharacterStream sqljXmlCharacterStream =
  new sqlj.runtime.CharacterStream(xmlReader,  xmlString.length());
#sql [ctx] {INSERT INTO CUSTACC VALUES (4, :sqljXmlCharacterStream)};
```

sqljXmlCharacterStream は文字タイプのため、内部エンコード方式が指定されているかどうかに関係なく、外部エンコード方式が使用されます。

例: 文書を XML 列から java.sql.SQLXML ホスト式に取り出し、このデータを表の XML 列に挿入します。

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
java.sql.SQLXML xmlObject = (java.sql.SQLXML)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

データを取り出した後、このデータは UTF-8 エンコード方式のままなので、このデータを他の XML 列に挿入する場合も変換は実行されません。

例: 文書を XML 列から com.ibm.db2.jcc.DB2Xml ホスト式に取り出し、このデータを表の XML 列に挿入します。

```
java.sql.ResultSet rs = s.executeQuery ("SELECT * FROM CUSTACC");
rs.next();
com.ibm.db2.jcc.DB2Xml xmlObject = (com.ibm.db2.jcc.DB2Xml)rs.getObject(2);
#sql [ctx] {INSERT INTO CUSTACC VALUES (6, :xmlObject)};
```

データを取り出した後、このデータは UTF-8 エンコード方式のままなので、このデータを他の XML 列に挿入する場合も変換は実行されません。

SQLJ アプリケーションにおける XML データの取り出し

SQLJ アプリケーションで、データをデータベース表の XML 列から取り出す場合は、出力データは明示的または暗黙的に直列化される必要があります。

データの XML 列からの取り出しに使用できるホスト式またはイテレーターのデータ・タイプは以下のとおりです。

- java.sql.SQLXML (SDK for Java バージョン 6 以降、および IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降が必要)
- com.ibm.db2.jcc.DB2Xml (非推奨)
- String
- byte[]
- sqlj.runtime.AsciiStream
- sqlj.runtime.BinaryStream
- sqlj.runtime.CharacterStream

アプリケーションで、データの取り出しの前に XMLSERIALIZE 関数が呼び出されない場合は、データは UTF-8 から文字データ・タイプ用の外部アプリケーション・エンコード方式か、またはバイナリー・データ・タイプ用の内部エンコード方式に変換されます。 XML 宣言は追加されません。ホスト式が java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml タイプのオブジェクトの場合は、追加のメソッドを呼び出して、データをこのオブジェクトから取り出す必要があります。呼び出すメソッドにより、出力データのエンコード方式、およびエンコード方式の指定を含む XML 宣言が追加されるかどうかが決まります。

次の表には、java.sql.SQLXML または com.ibm.db2.jcc.DB2Xml オブジェクトからのデータの取り出しに呼び出すことができるメソッド、および対応する出力データ・タイプと XML 宣言でのエンコード方式のタイプがリストされています。

表 32. SQLXML および DB2Xml メソッド、データ・タイプ、および追加されるエンコード仕様

メソッド	出力データ・タイプ	追加される XML 内部エンコード宣言のタイプ
SQLXML.getBinaryStream	InputStream	なし
SQLXML.getCharacterStream	Reader	なし
SQLXML.getSource	Source	なし
SQLXML.getString	String	なし
DB2Xml.getDB2AsciiStream	InputStream	なし
DB2Xml.getDB2BinaryStream	InputStream	なし
DB2Xml.getDB2Bytes	byte[]	なし
DB2Xml.getDB2CharacterStream	Reader	なし
DB2Xml.getDB2String	String	なし
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	getDB2XmlBinaryStream *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlBytes	byte[]	DB2Xml.getDB2XmlBytes *targetEncoding* パラメーターで指定
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

戻されるデータに対してアプリケーションが XMLSERIALIZE 関数を実行すると、関数の実行後に、そのデータには XML データ・タイプではなく XMLSERIALIZE 関数で指定されたデータ・タイプが含まれます。そのため、ドライバーは指定されたタイプとしてそのデータを処理し、内部エンコード宣言をすべて無視します。

例: データを XML 列から String ホスト式に取り出します。

```
#sql iterator  XmlStringIter (int, String);
#sql [ctx] siter = {SELECT C1, CADOC from CUSTACC};
#sql {FETCH :siter INTO :row, :outString};
```

String タイプは文字タイプであるため、データは UTF-8 から外部エンコード方式に変換され (これがデフォルトの JVM エンコード方式)、XML 宣言なしで返されます。

例: データを XML 列から byte[] ホスト式に取り出します。

```
#sql iterator  XmlByteArrayIter (int, byte[]);
XmlByteArrayIter biter = null;
#sql [ctx] biter =  {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :biter INTO :row, :outBytes};
```

byte[] タイプはバイナリー・タイプであるため、UTF-8 エンコード方式からのデータ変換は実行されず、データは XML 宣言なしで返されます。

例: 文書を XML 列から java.sql.SQLXML ホスト式に取り出しますが、バイナリー・ストリームのデータが必要な場合です。

```
#sql iterator  SqlXmlIter (int, java.sql.SQLXML);
SqlXmlIter SQLXMLiter = null;
java.sql.SQLXML outSqlXml = null;
```

```
#sql [ctx] SqlXmlIter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :SqlXmlIter INTO :row, :outSqlXml};
java.io.InputStream XmlStream = outSqlXml.getBinaryStream();
```

FETCH ステートメントでは、データが UTF-8 エンコード方式で SQLXML オブジェクトに取り出されます。 SQLXML.getBinaryStream では、データがバイナリー・ストリームで保管されます。

例: 文書を XML 列から com.ibm.db2.jcc.DB2Xml ホスト式に取り出しますが、UTF-8 の内部エンコード方式の指定が含まれた XML 宣言を持つバイト・ストリングとしてデータが必要な場合です。

```
#sql iterator  DB2XmlIter (int, com.ibm.db2.jcc.DB2Xml);
DB2XmlIter db2xmliter = null;
com.ibm.db2.jcc.DB2Xml outDB2Xml = null;
#sql [ctx] db2xmliter = {SELECT c1, CADOC from CUSTACC};
#sql {FETCH :db2xmliter INTO :row, :outDB2Xml};
byte[] byteArray = outDB2XML.getDB2XmlBytes("UTF-8");
```

FETCH ステートメントでは、データが UTF-8 エンコード方式で DB2Xml オブジェクトに取り出されます。 UTF-8 引数のある getDB2XmlBytes メソッドでは、UTF-8 エンコード方式の指定を含む XML 宣言が追加され、データがバイト配列で保管されます。

SQLJ アプリケーションの XMLCAST

XMLCAST を使用してホスト変数を SQLJ アプリケーションの XML データ・タイプにキャストするには、対応する SQL データ・タイプにホスト変数を事前にキャストしておく必要があります。

例: 以下のコードは、XMLCAST を使用して値を XML データ・タイプにキャストする前にストリング・ホスト変数を SQL 文字タイプ (VARCHAR など) にキャストすることが必要となる状況を示しています。

```
String xmlresult = null;
String varchar_hv = "San Jose";
...
#sql [con] {SELECT XMLCAST(CAST(:varchar_hv AS VARCHAR(32)) AS XML) INTO
  :xmlresult FROM SYSIBM.SYSDUMMY1};
```

SDK for Java バージョン 5 の機能の SQLJ での利用

SQLJ アプリケーションで、SDK for Java バージョン 5 で導入されたいくつかの機能を使用することができます。

静的インポート

import static 構文を使用すれば、静的メンバーが属しているクラスの名前でそれらのメンバーを修飾せずに、静的メンバーにアクセスすることができます。つまり SQLJ アプリケーションの場合、静的メンバーを修飾せずにホスト式でそれらのメンバーを使用することができます。

例: 次の形式のホスト式を宣言したいとします。

```
double r = cos(PI * E);
```

cos、PI、および E は java.lang.Math クラスのメンバーです。 cos、PI、および E を明示的に修飾しないで r を宣言するには、次の import static ステートメントをプログラム内に含めます。

```
import static java.lang.Math.*;
```

アノテーション

Java アノテーションとは、メタデータを Java プログラムに追加するための手段のことで、これらの Java プログラムがツールとライブラリーによって処理される方法に影響を与える可能性もあります。アノテーションは、インターフェース宣言と同様なアノテーション型宣言で宣言されます。 Java アノテーションは、以下のタイプのクラスまたはインターフェースに指定することができます。

- クラス宣言
- インターフェース宣言
- ネスト・クラス宣言
- ネスト・インターフェース宣言

Java アノテーションは SQLJ プログラムに直接組み込むことはできませんが、アノテーションを Java ソース・コードに組み込んだ後で、このソース・コードを SQLJ プログラムに組み込むことができます。

例: 次のマーカー・アノテーションを MyAnnot.java というプログラム内で宣言するとします。

```
public @interface MyAnot { }
```

次のマーカー・アノテーションも MyAnnot2.java というプログラム内で宣言します。

```
public @interface MyAnot2 { }
```

これで、これらのアノテーションを SQLJ プログラム内で以下のように使用することができます。

```
// Class annotations
@MyAnot2 public @MyAnot class  TestAnnotation
{
  // Field annotation
  @MyAnot
  private static final int field1 = 0;
  // Constructor annotation
  @MyAnot2 public @MyAnot TestAnnotation  () { }
   // Method annotation
   @MyAnot
   public static void main (String a[])
   {
      TestAnnotation TestAnnotation_o = new TestAnnotation();
      TestAnnotation_o.runThis();
   }
  // Inner class annotation
  public static @MyAnot class TestAnotherInnerClass { }
  // Inner interface annotation
  public static @MyAnot interface TestAnotInnerInterface { }
}
```

列挙型

列挙型とは、一連の順序付けられた値で構成されるデータ型のことです。 SDK for Java バージョン 5 では、列挙型用の enum 型が導入されました。

enum は以下の場所に組み込むことができます。

- SQLJ プログラムに組み込む Java ソース・ファイル (.java ファイル) 内
- SQLJ のクラス宣言内

例: 以下の TestEnum.sqlj クラス宣言に enum 型が組み込まれています。

```
public class TestEnum2
{
  public enum Color {
   RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET}
  Color color;
  ...                           // Get the value of color
  switch (color) {
 case RED:
    System.out.println("Red is at one end of the spectrum.");
    #sql[ctx] { INSERT INTO MYTABLE VALUES (:color) };
   break;
 case VIOLET:
    System.out.println("Violet is on the other end of the spectrum.");
   break;
 case ORANGE:
 case YELLOW:
 case GREEN:
  case BLUE:
  case INDIGO:
    System.out.println("Everything else is in the middle.");
   break;
}
```

総称

総称を Java プログラム内で使用して、型を Java コレクションに割り当てることができます。 SQLJ 変換プログラムでは、Java の総称構文が許容されます。 SQLJ プログラムで使用できる総称の例は以下のとおりです。

- List オブジェクトの List:

  ```
  List <List<String>> strList2 = new ArrayList<List<String>>();
  ```

- キーと値の組みに String 型を持つ HashMap:

  ```
  Map <String,String> map = new HashMap<String,String>();
  ```

- 任意の型のエレメントを持つ List を使用するメソッド:

  ```
  public void mthd(List <?> obj) {
  ...
  }
  ```

総称は SQLJ ホスト変数内で使用できますが、総称の使用の意義は、SQLJ 変換プログラムではそれらのホスト変数の型を判別できないため限られています。

拡張された for ループ

拡張された for を使用すれば、一連の命令をコレクションまたは配列の各メンバーに対して実行することを指定できます。イテレーターを、拡張された for ループ内でホスト式で使用することができます。

例: 配列 names 内の項目ごとに表 TAB に対して INSERT を実行します。

```
String[] names = {"ABC","DEF","GHI"};
for (String n : names)
{
  #sql {INSERT INTO TAB (VARCHARCOL) VALUES(:n) };
}
```

可変引数

可変引数により、任意の数の値をメソッドに受け渡すことが容易になります。メソッド宣言の最後の引数の位置にある可変引数により、最後の引数が配列または一続きの引数であることが示されます。 SQLJ プログラムでは、受け渡されたホスト式内の引数を使用することができます。

例: 任意の数の Object 型のパラメーターを、各パラメーター値を表 TAB に挿入するメソッドに受け渡します。

```
public void runThis(Object...  objects) throws SQLException
{
  for (Object obj : objects)
  {
    #sql { INSERT INTO TAB (VARCHARCOL) VALUES(:obj) };
  }
}
```

SQLJ アプリケーションにおけるトランザクション制御

SQLJ アプリケーションでは、他のタイプの SQL アプリケーションと同様に、トランザクション制御にトランザクションの明示的または暗黙的なコミットおよびロールバック、およびトランザクションの分離レベルの設定が関係します。

SQLJ トランザクションの分離レベルの設定

SQLJ プログラムの作業単位の分離レベルを設定するには、SET TRANSACTION ISOLATION LEVEL 節を使用します。

SET TRANSACTION ISOLATION LEVEL 節に指定できる値と、それぞれの値に対応する DB2 での等価値を以下の表にまとめます。

表 33. SQLJ と DB2 の分離レベルの等価値

SET TRANSACTION 値	DB2 の分離レベル
SERIALIZABLE	反復可能読み取り
REPEATABLE READ	読み取り固定
READ COMMITTED	カーソル固定
READ UNCOMMITTED	非コミット読み取り

分離レベルは、基礎になっている JDBC 接続と SQLJ 接続の両方に影響を及ぼします。

SQLJ トランザクションのコミットまたはロールバック

SQLJ 接続の自動コミットを使用不可にした場合は、明示的なコミット操作またはロールバック操作を実行する必要があります。

そのためには、SQL の COMMIT ステートメントまたは ROLLBACK ステートメントを含んだ実行節を使用します。

SQLJ プログラムでトランザクションをコミットするには、以下のようなステートメントを使用します。

```
#sql [myConnCtx] {COMMIT};
```

SQLJ プログラムでトランザクションをロールバックするには、以下のようなステートメントを使用します。

```
#sql [myConnCtx] {ROLLBACK};
```

SQLJ アプリケーションで SQL のエラーと警告を処理する

SQLJ 節は、SQL エラーが発生すると SQLException をスローしますが、ほとんどの SQL 警告では SQLException をスローしません。

SQLJ は、以下の状況で SQLException を生成します。

- SQL ステートメントが負の SQL エラー・コードを返したとき
- SELECT INTO SQL ステートメントが +100 の SQL エラー・コードを返したとき

その他の SQL 警告については、明示的に確認する必要があります。

- SQL エラー処理のために、SQLJ ステートメントの周りに try/catch ブロックを組み込みます。
- SQL 警告処理のために、各 SQLJ ステートメントの後で getWarnings メソッドを呼び出します。

SQLJ アプリケーションでの SQL エラーの処理

SQLJ の節では、エラー処理のために JDBC クラス java.sql.SQLException を使用します。

SQLJ アプリケーションで SQL エラーを処理するには、以下の手順を実行します。

1. java.sql.SQLException クラスをインポートします。
2. Java エラー処理の `try/catch` ブロックを使用して、SQL エラー発生時のプログラム・フローを変更します。
3. SQLException からエラー情報を取得します。

   getErrorCode メソッドによって SQL エラー・コードを、getSQLState メソッドによって SQLSTATE をそれぞれ取得できます。

   IBM Data Server Driver for JDBC and SQLJ を使用している場合は、SQLException を DB2Diagnosable オブジェクトにキャストすることによって、例外に関する追加情報を取得できます (JDBC アプリケーションで追加情報を取得する場合と同じです)。

   Linux、UNIX、Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) の場合は、標準の SQLException を使用して SQL エラー情報を取得します。

SELECT ステートメントが失敗したときに発生する SQL エラーを出力するコードを以下に示します。

```
try {
  #sql [ctxt] {SELECT LASTNAME INTO :empname
    FROM EMPLOYEE WHERE EMPNO='000010'};
}
catch(SQLException e) {
  System.out.println("Error code returned: " + e.getErrorCode());
}
```

SQLJ アプリケーションでの SQL 警告の処理

SELECT INTO ステートメントの +100 の SQL エラー・コードを除いて、 DB2 の警告では SQLException がスローされません。 DB2 の警告を処理するには、java.sql.SQLWarning クラスに対するアクセス権をプログラムに与える必要があります。

警告に関する DB2 固有の情報を取得するには、com.ibm.db2.jcc.DB2Diagnosable インターフェースと com.ibm.db2.jcc.DB2Sqlca クラスに対するアクセス権もプログラムに与える必要があります。その後、以下の手順を実行します。

1. SQL 節の実行コンテキストをセットアップします。実行コンテキストのセットアップ方法については、『SQLJ での SQL ステートメントの実行の制御』を参照してください。
2. DB2 の警告をチェックするには、SQLJ 節を実行した後に getWarnings メソッドを呼び出します。

   getWarnings からは、SQL ステートメントが生成する最初の SQLWarning オブジェクトが返されます。後続の SQLWarning オブジェクトは、最初のオブジェクトにチェーニングされます。
3. IBM Data Server Driver for JDBC and SQLJ 使用時に、SQLWarning オブジェクトから DB2 固有の情報を取得するには、『IBM Data Server Driver for JDBC and SQLJ 使用時の SQLException の処理』の手順を実行します。

実行コンテキスト execCtx を使用する SQL 節の SQLWarning オブジェクトを取得する例を以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
ExecutionContext execCtx=myConnCtx.getExecutionContext();    1
                                    // Get default execution context from
                                    // connection context
SQLWarning sqlWarn;
...
#sql [myConnCtx,execCtx] {SELECT LASTNAME INTO :empname
  FROM EMPLOYEE WHERE EMPNO='000010'};
if ((sqlWarn = execCtx.getWarnings()) != null)              2
System.out.println("SQLWarning " + sqlWarn);
```

SQLJ アプリケーションでのデータ・ソースへの接続のクローズ

データ・ソースへの接続を使用した作業が完了したら、データ・ソースへの接続を閉じる必要があります。そうすれば、接続コンテキスト・オブジェクトの DB2 と SQLJ のリソースがすぐに解放されます。

データ・ソースへの接続を閉じるには、いずれかの ConnectionContext.close メソッドを使用します。

- ConnectionContext.close() または ConnectionContext.close (ConnectionContext.CLOSE_CONNECTION) を実行すると、データ・ソースへの接続と接続コンテキストの両方が閉じます。
- ConnectionContext.close(ConnectionContext.KEEP_CONNECTION) を実行すると、接続コンテキストは閉じますが、データ・ソースへの接続は閉じません。

データ・ソースへの接続を閉じないで接続コンテキストだけを閉じるコードを以下に示します。

```
...
ctx = new EzSqljctx(con0);          // Create a connection context object
                                    // from JDBC connection con0
...                                   // Perform various SQL operations
 EzSqljctx.close(ConnectionContext.KEEP_CONNECTION);
                                    // Close the connection context but keep
                                    // the connection to the data source open
```

第 5 章 IBM Data Server Driver for JDBC and SQLJ 使用時のセキュリティー

IBM Data Server Driver for JDBC and SQLJ を使用する場合、securityMechanism プロパティーの値を指定してセキュリティー・メカニズムを選択します。

このプロパティーは以下の方法のいずれかで設定できます。

- DriverManager インターフェースを使用する場合、java.util.Properties パラメーターを含む getConnection メソッドの形式を呼び出す前に、java.util.Properties オブジェクトで securityMechanism を設定します。
- DataSource インターフェースを使用しており、独自の DataSource オブジェクトを作成およびデプロイしている場合、DataSource オブジェクトを作成した後で DataSource.setSecurityMechanism メソッドを呼び出します。

DB2Connection.getDB2SecurityMechanism メソッドを呼び出すことにより、接続に有効なセキュリティー・メカニズムを判別できます。

以下の表は、 IBM Data Server Driver for JDBC and SQLJ がサポートするセキュリティー・メカニズム、およびそれらのセキュリティー・メカニズムをサポートするデータ・ソースをリストしています。

表 34. IBM Data Server Driver for JDBC and SQLJ セキュリティー・メカニズムのデータベース・サーバー・サポート

セキュリティー・メカニズム	**サポート**			
	DB2 Database for Linux, UNIX, and Windows	DB2 for z/OS	IBM Informix Dynamic Server	DB2 for i
ユーザー ID およびパスワード	はい	はい	はい	はい
ユーザー ID のみ	はい	はい	はい	はい
ユーザー ID および暗号化されたパスワード	はい	はい	はい	はい[2]
暗号化されたユーザー ID	はい	はい	いいえ	いいえ
暗号化されたユーザー ID および暗号化されたパスワード	はい	はい	はい	はい[2]
暗号化されたユーザー ID および暗号化されたセキュリティー・センシティブ・データ	いいえ	はい	いいえ	いいえ
暗号化されたユーザー ID、暗号化されたパスワード、暗号化されたセキュリティー・センシティブ・データ	はい	はい	いいえ	いいえ

表 34. *IBM Data Server Driver for JDBC and SQLJ* セキュリティー・メカニズムのデータベース・サーバー・サポート *(続き)*

セキュリティー・メカニズム	サポート			
Kerberos[1]	はい	はい	いいえ	はい
プラグイン[1]	はい	いいえ	いいえ	いいえ

注:

1. IBM Data Server Driver for JDBC and SQLJ Type 4 接続にのみ使用可能。
2. データ・ソースのバージョンは、DB2 for i V6R1 以降でなければなりません。

以下の表は、 IBM Data Server Driver for JDBC and SQLJ がサポートするセキュリティー・メカニズム、および各セキュリティー・メカニズムを指定するために securityMechanism プロパティーに指定する必要がある値をリストしています。

デフォルトのセキュリティー・メカニズムは CLEAR_TEXT_PASSWORD_SECURITY です。サーバーが、CLEAR_TEXT_PASSWORD_SECURITY をサポートしない一方で、ENCRYPTED_USER_AND_PASSWORD_SECURITY をサポートするときは、 IBM Data Server Driver for JDBC and SQLJ ドライバーはセキュリティー・メカニズムを ENCRYPTED_USER_AND_PASSWORD_SECURITY に更新し、サーバーへの接続を試みます。リクエスターとサーバー間でこれ以外のセキュリティー・メカニズム・サポートの不一致がある場合には、エラーになります。

表 35. *IBM Data Server Driver for JDBC and SQLJ* によってサポートされるセキュリティー・メカニズム

セキュリティー・メカニズム	securityMechanism プロパティー値
ユーザー ID およびパスワード	DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY
ユーザー ID のみ	DB2BaseDataSource.USER_ONLY_SECURITY
ユーザー ID および暗号化されたパスワード	DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY
暗号化されたユーザー ID	DB2BaseDataSource.ENCRYPTED_USER_ONLY_SECURITY
暗号化されたユーザー ID および暗号化されたパスワード	DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY
暗号化されたユーザー ID および暗号化されたセキュリティー・センシティブ・データ	DB2BaseDataSource.ENCRYPTED_USER_AND_DATA_SECURITY
暗号化されたユーザー ID、暗号化されたパスワード、暗号化されたセキュリティー・センシティブ・データ	DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
Kerberos	DB2BaseDataSource.KERBEROS_SECURITY
プラグイン	DB2BaseDataSource.PLUGIN_SECURITY

以下の表は、考えられる DB2 Database for Linux, UNIX, and Windows サーバー認証タイプ、および互換性のある IBM Data Server Driver for JDBC and SQLJ securityMechanism プロパティー値を示しています。

表 36. 互換性のある DB2 Database for Linux, UNIX, and Windows サーバー認証タイプおよび IBM Data Server Driver for JDBC and SQLJ securityMechanism 値

DB2 Database for Linux, UNIX, and Windows サーバー認証タイプ	**securityMechanism 設定**
CLIENT	USER_ONLY_SECURITY
SERVER	CLEAR_TEXT_PASSWORD_SECURITY
SERVER_ENCRYPT	CLEAR_TEXT_PASSWORD_SECURITY、 ENCRYPTED_PASSWORD_SECURITY、または ENCRYPTED_USER_AND_PASSWORD_SECURITY
DATA_ENCRYPT	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
KERBEROS	KERBEROS_SECURITY または PLUGIN_SECURITY[2]
KRB_SERVER_ENCRYPT	KERBEROS_SECURITY、PLUGIN_SECURITY[1]、 ENCRYPTED_PASSWORD_SECURITY、または ENCRYPTED_USER_AND_PASSWORD_SECURITY
GSSPLUGIN	PLUGIN_SECURITY[1] または KERBEROS_SECURITY
GSS_SERVER_ENCRYPT[3]	CLEAR_TEXT_PASSWORD_SECURITY、 ENCRYPTED_PASSWORD_SECURITY、 ENCRYPTED_USER_AND_PASSWORD_SECURITY、 PLUGIN_SECURITY、または KERBEROS_SECURITY

注:

1. PLUGIN_SECURITY の場合、プラグインは Kerberos プラグインでなければなりません。
2. PLUGIN_SECURITY の場合、サーバーでのプラグインのうちの 1 つが、Kerberos をサポートするプラグインとして自らを識別します。
3. GSS_SERVER_ENCRYPT は GSSPLUGIN と SERVER_ENCRYPT の組み合わせです。

IBM Data Server Driver for JDBC and SQLJ 使用時のユーザー ID およびパスワード・セキュリティー

IBM Data Server Driver for JDBC and SQLJ を使用する場合、使用可能なセキュリティーの方法の 1 つが、ユーザー ID およびパスワード・セキュリティーです。

JDBC 接続で、ユーザー ID およびパスワード・セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: DriverManager.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。以下に例を示します。

```
import java.sql.*;          // JDBC base
...
String id = "dbadm";        // Set user ID
String pw = "dbadm";        // Set password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                            // Set URL for the data source

Connection con = DriverManager.getConnection(url, id, pw);
                            // Create connection
```

別の方法として、ユーザー ID とパスワードを URL ストリングに直接設定することができます。以下に例を示します。

```
import java.sql.*;         // JDBC base
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=dbadm;password=dbadm;";

                           // Set URL for the data source
Connection con = DriverManager.getConnection(url);
                           // Create connection
```

別の方法として、 Properties オブジェクトに user および password プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。オプションで、securityMechanism プロパティーを設定して、ユーザー ID およびパスワード・セキュリティーを使用していることを示すことができます。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "dbadm");            // Set user ID for the connection
properties.put("password", "dbadm");        // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                                            // Set security mechanism to
                                            // user ID and password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

DataSource インターフェースの場合: DataSource.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
Context ctx=new InitialContext();           // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Get DataSource object
String id = "dbadm";                        // Set user ID
String pw = "dbadm";                        // Set password
Connection con = ds.getConnection(id, pw);
                                            // Create connection
```

DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setUser および DataSource.setPassword メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。オプションで、DataSource.setSecurityMechanism メソッド・プロパティーを呼び出して、ユーザー ID およびパスワード・セキュリティーを使用していることを示すことができます。以下に例を示します。

```
...
com.ibm.db2.jcc.DB2SimpleDataSource ds =       // Create DB2SimpleDataSource object
  new com.ibm.db2.jcc.DB2SimpleDataSource();
ds.setDriverType(4);                           // Set driver type
ds.setDatabaseName("san_jose");                // Set location
ds.setServerName("mvs1.sj.ibm.com");           // Set server name
ds.setPortNumber(5021);                        // Set port number
```

```
ds.setUser("dbadm");                          // Set user ID
ds.setPassword("dbadm");                      // Set password
ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
                                              // Set security mechanism to
                                              // user ID and password
```

IBM Data Server Driver for JDBC and SQLJ 使用時のユーザー ID のみのセキュリティー

IBM Data Server Driver for JDBC and SQLJ を使用する場合、使用可能なセキュリティーの方法の 1 つが、ユーザー ID のみのセキュリティーです。

JDBC 接続で、ユーザー ID セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: Properties オブジェクトに、user および securityMechanism プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだ形式の getConnection メソッドを呼び出して、ユーザー ID およびセキュリティーのメカニズムを設定します。以下に例を示します。

```
import java.sql.*;                    // JDBC base
import com.ibm.db2.jcc.*;             // IBM Data Server Driver
                                      // for JDBC and SQLJ
                                      // implementation of JDBC
...
Properties properties = new Properties();
                                      // Create a Properties object
properties.put("user", "db2adm");     // Set user ID for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
                                      // Set security mechanism to
                                      // user ID only
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                      // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                      // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、DataSource オブジェクトの作成後に、DataSource.setUser および DataSource.setSecurityMechanism メソッドを呼び出して、ユーザー ID およびセキュリティーのメカニズムを設定することができます。以下に例を示します。

```
import java.sql.*;                    // JDBC base
import com.ibm.db2.jcc.*;             // IBM Data Server Driver
                                      // for JDBC and SQLJ
                                      // implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                      // Create DB2SimpleDataSource object
db2ds.setDriverType(4);               // Set the driver type
db2ds.setDatabaseName("san_jose");    // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                      // Set the server name
db2ds.setPortNumber(5021);            // Set the port number
db2ds.setUser("db2adm");              // Set the user ID
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
                                      // Set security mechanism to
                                      // user ID only
```

暗号化されたパスワード、ユーザー ID、または IBM Data Server Driver for JDBC and SQLJ 使用時のユーザー ID とパスワード・セキュリティー

IBM Data Server Driver for JDBC and SQLJ は、暗号化されたパスワード・セキュリティー、暗号化されたユーザー ID のセキュリティー、またはデータ・ソースにアクセスするための暗号化されたユーザー ID およびパスワードのセキュリティーをサポートします。

IBM Data Server Driver for JDBC and SQLJ では、56 ビット DES (弱い) 暗号化または 256 ビット AES (強い) 暗号化がサポートされます。 AES 暗号化は IBM Data Server Driver for JDBC and SQLJ Type 4 接続でのみ使用できます。 encryptionAlgorithm ドライバー・プロパティーを設定することにより、56 ビット DES 暗号化 (encryptionAlgorithm の値は 1) と 256 ビット AES 暗号化 (encryptionAlgorithm の値は 2) の間で選択します。 256 ビット AES 暗号化は、データベース・サーバーがこれをサポートし、これを使用するよう構成されている場合に限り、接続に使用されます。

暗号化パスワード・セキュリティー、暗号化されたユーザー ID セキュリティー、または暗号化ユーザー ID と暗号化パスワード・セキュリティーを使用する場合、IBM Java Cryptography Extension (JCE) がクライアントで有効になっている必要があります。 IBM JCE は、IBM SDK for Java バージョン 1.4.2 以降でその一部として含まれています。

IBM JCE は、 IBM Data Server Driver for JDBC and SQLJ ドライバーから DB2 Database for Linux, UNIX, and Windows サーバーへの 56 ビット DES または 256 ビット AES 暗号化クライアント/サーバー通信を使用する必要があります。

AES 暗号化の場合、JCE 用の無制限ポリシー・ファイルを入手する必要があります。これは、以下の URL で入手可能です。 https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk

DB2 for i V6R1 以降のサーバーへの接続では、暗号化されたパスワード・セキュリティー、または暗号化されたユーザー ID とパスワード・セキュリティーを使用することができます。暗号化パスワード・セキュリティーまたは暗号化ユーザー ID と暗号化パスワード・セキュリティーの場合、IBM Java Cryptography Extension (ibmjceprovidere.jar) がクライアントにインストールされていなければなりません。 IBM JCE は、IBM SDK for Java バージョン 1.4.2 以降でその一部として含まれています。

暗号化されたユーザー ID セキュリティーまたは暗号化されたユーザー ID とパスワード・セキュリティーに加えて、暗号化されたセキュリティー・センシティブ・データも使用することができます。セキュリティー・センシティブ・データの暗号化は、`ENCRYPTED_USER_AND_DATA_SECURITY` または `ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY securityMechanism` 値によって指定できます。`ENCRYPTED_USER_AND_DATA_SECURITY` は、 DB2 for z/OS サーバーへの接続にのみ有効であり、かつ DES 暗号化専用です (encryptionAlgorithm 値は 1)。

セキュリティー・センシティブ・データの暗号化を指定すると、 DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows データベース・サーバーは、以下のデータを暗号化します。

- 準備中、実行中、またはパッケージにバインドされている SQL ステートメント
- パラメーター情報の入出力
- 結果セット
- LOB データ
- XML データ
- 記述操作の結果

暗号化されたセキュリティー・センシティブ・データを使用する前に、z/OS 統合暗号化サービス機能 (Integrated Cryptographic Services Facility) を z/OS オペレーティング・システムにインストールし、使用可能にする必要があります。

JDBC 接続で、暗号化されたユーザー ID または暗号化されたパスワード・セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: Properties オブジェクトに、 user、password、および securityMechanism プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID、パスワード、およびセキュリティーのメカニズムを設定します。例えば、次のようなコードを使用して、AES 暗号化を使ったユーザー ID および暗号化されたパスワード・セキュリティー・メカニズムを設定します。

```
import java.sql.*;                              // JDBC base
import com.ibm.db2.jcc.*;                       // IBM Data Server Driver for JDBC
                                                // and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "dbadm");          // Set user ID for the connection
properties.put("password", "dbadm");      // Set password for the connection
properties.put("securityMechanism", "2");
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY +
  ""));
                                          // Set security mechanism to
                                          // user ID and encrypted password
properties.put("encryptionAlgorithm", "2");
                                          // Request AES security
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、DataSource オブジェクトの作成後に、DataSource.setUser、DataSource.setPassword、および DataSource.setSecurityMechanism メソッドを呼び出して、ユーザー ID、パスワード、およびセキュリティーのメカニズムを設定することができます。例えば、次のようなコードを使用して、AES 暗号化を使った暗号化されたユーザー ID および暗号化されたパスワード・セキュリティー・メカニズムを設定します。

```
import java.sql.*;                              // JDBC base
import com.ibm.db2.jcc.*;                       // IBM Data Server Driver for JDBC
                                                // and SQLJ implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
```

```
                                        // Create the DataSource object
ds.setDriverType(4);                    // Set the driver type
ds.setDatabaseName("san_jose");         // Set the location
ds.setServerName("mvs1.sj.ibm.com");
                                        // Set the server name
ds.setPortNumber(5021);                 // Set the port number
ds.setUser("db2adm");                   // Set the user ID
ds.setPassword("db2adm");               // Set the password
ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY);
                                        // Set security mechanism to
                                        // User ID and encrypted password
ds.setEncryptionAlgorithm(2);           // Request AES encryption
```

IBM Data Server Driver for JDBC and SQLJ 使用時の Kerberos セキュリティー

Kerberos セキュリティー用の JDBC サポートは、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続でのみ使用可能です。

Kerberos セキュリティーの JDBC サポートを使用可能にするには、Software Development Kit (SDK) for Java の以下のコンポーネントも使用可能にする必要があります。

- Java Cryptography Extension
- Java Generic Security Service (JGSS)
- Java Authentication and Authorization Service (JAAS)

これらのコンポーネントを使用可能にする方法については、ご使用の SDK for Java の資料を参照してください。

接続に Kerberos セキュリティーを指定する方法には、以下の 3 つがあります。

- ユーザー ID とパスワードを使用する
- ユーザー ID もパスワードも使用しない
- 委任証明書を使用する

ユーザー ID とパスワードを使用する Kerberos セキュリティー

この場合、Kerberos により、指定されたユーザー ID とパスワードが使用されてチケット許可チケット (TGT) が入手され、これによってデータベース・サーバーから認証を受けることができます。

user、password、kerberosServerPrincipal、および securityMechanism プロパティーを設定する必要があります。securityMechanism プロパティーを com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY (11) に設定します。kerberosServerPrincipal プロパティーには、データベース・サーバーにより Kerberos 鍵配布センター (KDC) に登録されるプリンシパル名を指定します。

DriverManager インターフェースの場合: Properties オブジェクトに、user、password、kerberosServerPrincipal、および securityMechanism プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだ形式の getConnection メソッドを呼び出して、ユーザー ID、パスワード、Kerberos サーバ

ーおよびセキュリティーのメカニズムを設定します。例えば、次のようなコードを使用して、ユーザー ID およびパスワードで Kerberos セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                            // Set the Kerberos server
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、DataSource オブジェクトの作成後に、DataSource.setKerberosServerPrincipal および DataSource.setSecurityMechanism メソッドを呼び出して、Kerberos サーバーおよびセキュリティーのメカニズムを設定します。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setUser("db2adm");                    // Set the user
db2ds.setPassword("db2adm");                // Set the password
db2ds.setServerName("mvs1.sj.ibm.com");
                                            // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                            // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                            // Set security mechanism to
                                            // Kerberos
```

ユーザー ID もパスワードを使用しない Kerberos セキュリティー

この場合、Kerberos のデフォルト証明書キャッシュにチケット許可チケット (TGT) が含まれている必要があり、これによってデータベース・サーバーから認証を受けることができます。

kerberosServerPrincipal および securityMechanism プロパティーを設定する必要があります。securityMechanism プロパティーを com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY (11) に設定します。

DriverManager インターフェースの場合: Properties オブジェクトに、kerberosServerPrincipal および securityMechanism プロパティーを設定し、Properties オブジェクトをパラメーターとして組み込んだ形式の getConnection メソッドを呼び出して、Kerberos サーバーおよびセキュリティーのメカニズムを設定します。例えば、次のようなコードを使用して、ユーザー ID およびパスワードなしで Kerberos セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                            // Set the Kerberos server
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、DataSource オブジェクトの作成後に、DataSource.setKerberosServerPrincipal および DataSource.setSecurityMechanism メソッドを呼び出して、Kerberos サーバーおよびセキュリティーのメカニズムを設定します。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                            // Create the DataSource object
db2ds.setDriverType(4);                     // Set the driver type
db2ds.setDatabaseName("san_jose");          // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                            // Set the server name
db2ds.setPortNumber(5021);                  // Set the port number
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                            // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                            // Set security mechanism to
                                            // Kerberos
```

別のプリンシパルからの委任証明書を使用した Kerberos セキュリティー

この場合、別のプリンシパルから受け渡される委任証明書を使用して、データベース・サーバーから認証を受けます。

kerberosServerPrincipal、gssCredential、および securityMechanism プロパティーを設定する必要があります。securityMechanism プロパティーを com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY (11) に設定します。

DriverManager インターフェースの場合: Properties オブジェクトに、kerberosServerPrincipal および securityMechanism プロパティーを設定して、Kerberos サーバー、委任証明書、およびセキュリティーのメカニズムを設定します。次に、Properties オブジェクトをパラメーターとして組み込んだ形式の getConnection メソッドを呼び出します。例えば、次のようなコードを使用して、ユーザー ID およびパスワードなしで Kerberos セキュリティー・メカニズムを設定します。

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // IBM Data Server Driver for JDBC
                                            // and SQLJ implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                            // Set the Kerberos server
properties.put("gssCredential",delegatedCredential);
                                            // Set the delegated credential
properties.put("securityMechanism",
  new String("" +
    com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                            // Set security mechanism to
                                            // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create the connection
```

DataSource インターフェースの場合: DataSource オブジェクトを作成してデプロイする場合、DataSource オブジェクトの作成後に、DataSource.setKerberosServerPrincipal、DataSource.setGssCredential、および DataSource.setSecurityMechanism メソッドを呼び出して、Kerberos サーバー、委任証明書、およびセキュリティーのメカニズムを設定します。以下に例を示します。

```
DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create the DataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");  // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Set the Kerberos server
db2ds.setGssCredential(delegatedCredential);
                                          // Set the delegated credential
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                          // Set security mechanism to
                                          // Kerberos
```

IBM Data Server Driver for JDBC and SQLJ のセキュリティー・プラグインのサポート

独自の認証メカニズムをロード可能なライブラリーまたはプラグインの形式で作成することができます。この認証メカニズムは、 DB2 Database for Linux, UNIX, and Windows でユーザー認証を実行するためにロードされます。 Java でのセキュリティー・プラグインの開発をサポートするために、 IBM Data Server Driver for JDBC and SQLJ ではセキュリティー・プラグインのサポートが提供されます。

IBM Data Server Driver for JDBC and SQLJ のセキュリティー・プラグインのサポートは、 DB2 Database for Linux, UNIX, and Windows サーバーに対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続でのみ使用可能です。

プラグイン・セキュリティーを使用するには、セキュリティー・プラグインがクライアント上に必要で、またサーバー上には別のプラグインが必要です。

セキュリティー・プラグインには、以下のものを組み込む必要があります。

- 抽象クラス com.ibm.db2.jcc.DB2JCCPlugin を拡張するクラス

  抽象クラス com.ibm.db2.jcc.DB2JCCPlugin に IBM Data Server Driver for JDBC and SQLJ が提供されます。
- クラス com.ibm.db2.jcc.DB2JCCPlugin 内の com.ibm.db2.jcc.DB2JCCPlugin.getTicket メソッド

  このメソッドでは、ユーザーの Kerberos チケットが検索され、セキュリティー・コンテキスト情報がバイト配列で返されます。このバイト配列の情報は、 IBM Data Server Driver for JDBC and SQLJ による DB2 データベース・サーバーへのアクセスに使用されます。
- org.ietf.jgss.GSSContext および `org.ietf.jgss.GSSCredential` インターフェースで定義されるいくつかのメソッドのインプリメンテーション

  これらのメソッドのインプリメンテーションは、Generic Security Service Application Program Interface, Version 2 (IETF RFC2743) および Generic Security Service API Version 2: Java-Bindings (IETF RFC2853) 仕様に準拠している必要があります。プラグインでは、以下の各メソッドをインプリメントして呼び出す必要があります。

  GSSContext.dispose
  コンテキスト・オブジェクトに保管されているシステム・リソースおよび暗号情報を解放し、コンテキストを無効にします。

  GSSContext.getCredDelegState
  信用証明情報の委任がコンテキストで使用可能かどうかを判別します。

  GSSContext.getMutualAuthState
  相互認証がコンテキストで使用可能かどうかを判別します。

  GSSContext.initSecContext
  コンテキスト作成フェーズを開始し、ピアの acceptSecContext メソッドによって生成されるすべてのトークンを処理します。

  GSSContext.requestCredDeleg
  コンテキストが確立される際に、イニシエーターの信用証明情報がアクセプターに委任されることを要求します。

  GSSContext.requestMutualAuth
  コンテキストが確立される際に、相互認証を要求します。

  GSSCredential.dispose
  GSSCredential オブジェクトに含まれる機密情報を解放します。

Java セキュリティー・プラグインの作成に役立つ、以下の 2 つの Java プラグイン・サンプルが sqllib/samples/java/jdbc に提供されています。

JCCSimpleGSSPlugin.java
サーバー用の GSS-API プラグインのインプリメンテーション。これにより、ユーザー ID とパスワードの検査が実行されます。このサンプルは、C 言語のサンプル・プログラム gssapi_simple.c の Java バージョンです。

JCCKerberosPlugin.java
クライアント用の Kerberos セキュリティー・プラグイン。このサンプルは、C 言語のサンプル・プログラム IBMkrb5.c の Java バージョンです。

アプリケーション・プログラムで JDBC プラグイン・セキュリティーが使用されて接続が取得される場合、以下の Connection または DataSource プロパティーを設定する必要があります。

表 37. Java セキュリティー・プラグインを使用するための Connection または DataSource プロパティーの設定

プロパティー	設定
com.ibm.db2.jcc.DB2BaseDataSource.user	Connection を取得する際のユーザー ID
com.ibm.db2.jcc.DB2BaseDataSource.password	ユーザー ID のパスワード
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	com.ibm.db2.jcc.DB2BaseDataSource.PLUGIN_SECURITY
com.ibm.db2.jcc.DB2BaseDataSource.pluginName	サーバー・サイド・セキュリティー・プラグインのプラグイン・モジュールの名前
com.ibm.db2.jcc.DB2BaseDataSource.plugin	クライアント・サイド・セキュリティー・プラグインのプラグイン・オブジェクト

例: 下のコードでは、GSS-API プラグイン・セキュリティーを使用する接続のプロパティーが設定されます。この接続では、クライアント・サイドでサンプル・プラグインの JCCSimpleGSSPlugin が使用され、サーバー・サイドでサンプル・プラグインの gssapi_simple が使用されます。

```
java.util.Properties properties = new java.util.Properties();
properties.put("user", "db2admin");
properties.put("password", "admindb2");
properties.put("pluginName", "gssapi_simple");
properties.put("securityMechanism",
  new String(""+com.ibm.db2.jcc.DB2BaseDataSource.PLUGIN_SECURITY+""));
com.ibm.db2.jcc.DB2JCCPlugin plugin =
  new com.ibm.db2.jcc.samples.plugins.JCCSimpleGSSplugin();
properties.put("plugin", plugin);
Connection con = java.sql.DriverManager.getConnection(url,
  properties);
```

IBM Data Server Driver for JDBC and SQLJ での代替セキュリティー・メカニズムの使用

IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用していて、retryWithAlternativeSecurityMechanism を com.ibm.db2.jcc.DB2BaseDataSource.YES (1) に設定しており、接続のためのオリジナルのセキュリティー・メカニズムが失敗した場合、ドライバーは最も機密保護機能の高い代替セキュリティー・メカニズムを使って接続を再試行します。

以下の表では、 IBM Data Server Driver for JDBC and SQLJ のセキュリティー・メカニズムと、オリジナルの接続で与信の失敗があったときに使用される代替セキュリティー・メカニズムをリストしています。

表 38. オリジナルおよび代替の *IBM Data Server Driver for JDBC and SQLJ* セキュリティー・メカニズム

サーバー認証タイプ	オリジナル接続のための IBM Data Server Driver for JDBC and SQLJ 認証タイプ	接続再試行のための IBM Data Server Driver for JDBC and SQLJ 認証タイプ
CLIENT	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	USER_ONLY_SECURITY
	USER_ONLY_SECURITY	なし。 USER_ONLY_SECURITY は、オリジナル接続では失敗しません。
SERVER	• USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	CLEAR_TEXT_PASSWORD_SECURITY
	CLEAR_TEXT_PASSWORD_SECURITY	なし。 CLEAR_TEXT_PASSWORD_SECURITY は、オリジナル接続では失敗しません。
DB2 Database for Linux, UNIX, and Windows バージョン 8 フィックスパック 9 以前のための SERVER_ENCRYPT	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	ENCRYPTED_USER_AND_PASSWORD_SECURITY
	• ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY	なし。 ENCRYPTED_PASSWORD_SECURITY および ENCRYPTED_USER_AND_PASSWORD_SECURITY は、オリジナル接続では失敗しません。
DB2 Database for Linux, UNIX, and Windows バージョン 8 フィックスパック 10 以降のための SERVER_ENCRYPT	• USER_ONLY_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	ENCRYPTED_USER_AND_PASSWORD_SECURITY
	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY	なし。 CLEAR_TEXT_PASSWORD_SECURITY、ENCRYPTED_PASSWORD_SECURITY、および ENCRYPTED_USER_AND_PASSWORD_SECURITY は、オリジナル接続では失敗しません。
DATA_ENCRYPT	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
	ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY	なし。 ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY は、オリジナル接続では失敗しません。

表 38. オリジナルおよび代替の IBM Data Server Driver for JDBC and SQLJ セキュリティー・メカニズム (続き)

サーバー認証タイプ	オリジナル接続のための IBM Data Server Driver for JDBC and SQLJ 認証タイプ	接続再試行のための IBM Data Server Driver for JDBC and SQLJ 認証タイプ
KERBEROS	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • PLUGIN_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	KERBEROS_SECURITY
	KERBEROS_SECURITY	なし。 KERBEROS_SECURITY は、オリジナル接続では失敗しません。
GSSPLUGIN	• CLEAR_TEXT_PASSWORD_SECURITY • USER_ONLY_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	PLUGIN_SECURITY
	PLUGIN_SECURITY	なし。 PLUGIN_SECURITY は、オリジナル接続では失敗しません。
KRB_SERVER_ENCRYPT	• USER_ONLY_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	KERBEROS_SECURITY
	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • PLUGIN_SECURITY	なし。 CLEAR_TEXT_PASSWORD_SECURITY、ENCRYPTED_PASSWORD_SECURITY、ENCRYPTED_USER_AND_PASSWORD_SECURITY、KERBEROS_SECURITY、および PLUGIN_SECURITY は、オリジナル接続では失敗しません。
GSS_SERVER_ENCRYPT	• USER_ONLY_SECURITY • ENCRYPTED_USER_AND_DATA_SECURITY • ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY • ENCRYPTED_USER_ONLY_SECURITY	KERBEROS_SECURITY
	• CLEAR_TEXT_PASSWORD_SECURITY • ENCRYPTED_PASSWORD_SECURITY • ENCRYPTED_USER_AND_PASSWORD_SECURITY • KERBEROS_SECURITY • PLUGIN_SECURITY	なし。 CLEAR_TEXT_PASSWORD_SECURITY、ENCRYPTED_PASSWORD_SECURITY、ENCRYPTED_USER_AND_PASSWORD_SECURITY、KERBEROS_SECURITY、および PLUGIN_SECURITY は、オリジナル接続では失敗しません。

IBM Data Server Driver for JDBC and SQLJ のトラステッド・コンテキストのサポート

IBM Data Server Driver for JDBC and SQLJ では、Java プログラム内でのトラステッド接続の確立および使用を可能にするメソッドが提供されます。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

3 層アプリケーション・モデルは、データベース・サーバー、WebSphere Application Server などのミドルウェア・サーバー、およびエンド・ユーザーで構成されています。このモデルでは、ミドルウェア・サーバーがエンド・ユーザーに代わってデータベース・サーバーにアクセスします。トラステッド・コンテキストのサポートにより、ミドルウェア・サーバーがエンド・ユーザーに代わって任意のデータベース要求を実行する際に、エンド・ユーザーのデータベース ID とデータベース特権が使用されます。

トラステッド・コンテキストとは、データベース管理者が定義し、システムの許可 ID およびトラスト属性のセットが含まれたオブジェクトのことです。現行の DB2 データベース・サーバーでは、データベース接続がサポートされているコンテキストの唯一のタイプです。トラスト属性により、接続がトラステッド接続と見なされるために必要な接続の特性のセットが識別されます。データベース接続とトラステッド・コンテキストの間の関係は、データベース・サーバーへの接続が最初に作成されたときに確立され、この関係はデータベース接続が存続する間持続します。

トラステッド・コンテキストが定義され、 DB2 データベース・サーバーへの初期トラステッド接続が確立されると、ミドルウェア・サーバーでは、データベース・サーバーで新規ユーザーの再認証を行うことなく、このデータベース接続を異なるユーザーの下で使用することができます。

セキュリティー・ブリーチ (抜け穴) へのぜい弱性を避けるために、これらのトラステッド・メソッドを使用するアプリケーション・サーバーでは、非トラステッド接続メソッドを使用しないでください。

DB2ConnectionPoolDataSource クラスでは getDB2TrustedPooledConnection メソッドのいくつかのバージョンが提供され、また DB2XADataSource クラスでは getDB2TrustedXAConnection メソッドのいくつかのバージョンが提供され、これらによってアプリケーション・サーバーでの初期トラステッド接続の確立が可能になります。受け渡す接続プロパティーのタイプ、および Kerberos セキュリティーを使用するかどうかに基づいて、メソッドを選択します。アプリケーション・サーバーでこれらのメソッドの 1 つが呼び出されると、 IBM Data Server Driver for JDBC and SQLJ により以下の 2 つのエレメントを持つ Object[] 配列が返されます。

- 最初のエレメントには、初期接続用の接続インスタンスが含まれています。
- 2 番目のエレメントには、接続インスタンス用の固有の cookie が含まれています。この cookie は JDBC ドライバーによって生成され、その後の接続の再利用時に認証用に使用されます。

DB2PooledConnection クラスでは getDB2Connection メソッドのいくつかのバージョンが提供され、また DB2Connection クラスでは reuseDB2Connection メソッドのいくつかのバージョンが提供されます。これらによって、新規ユーザーの代わりにア

プリケーション・サーバーで既存のトラステッド接続の再利用が可能になります。アプリケーション・サーバーでは、これらのメソッドの 1 つが使用されて以下の項目が新規ユーザーに受け渡されます。

- 初期接続からの cookie
- 再利用される接続用の新規接続プロパティー

JDBC ドライバーは、提供された cookie が基礎となるトラステッド物理接続の cookie と一致することを調べて、接続要求がそのトラステッド物理接続を確立したアプリケーション・サーバーから発信されていることを確認します。 cookie が一致する場合、新規プロパティーを持つ接続が使用可能になり、この新規ユーザーが即時使用することができます。

例: 初期トラステッド接続の取得:

```
// Create a DB2ConnectionPoolDataSource instance
com.ibm.db2.jcc.DB2ConnectionPoolDataSource dataSource =
  new com.ibm.db2.jcc.DB2ConnectionPoolDataSource();
// Set properties for this instance
dataSource.setDatabaseName ("STLEC1");
dataSource.setServerName ("v7ec167.svl.ibm.com");
dataSource.setDriverType (4);
dataSource.setPortNumber(446);
java.util.Properties properties = new java.util.Properties();
// Set other properties using
// properties.put("property", "value");
// Supply the user ID and password for the connection
String user = "user";
String password = "password";
// Call getDB2TrustedPooledConnection to get the trusted connection
// instance and the cookie for the connection
Object[] objects = dataSource.getDB2TrustedPooledConnection(
                        user,password, properties);
```

例: 既存のトラステッド接続の再利用:

```
// The first item that was obtained from the previous getDB2TrustedPooledConnection
// call is a connection object. Cast it to a PooledConnection object.
javax.sql.PooledConnection pooledCon =
  (javax.sql.PooledConnection)objects[0];
properties = new java.util.Properties();
// Set new properties for the reused object using
// properties.put("property", "value");
// The second item that was obtained from the previous getDB2TrustedPooledConnection
// call is the cookie for the connection. Cast it as a byte array.
byte[] cookie = ((byte[])(objects[1]);
// Supply the user ID for the new connection.
String newuser = "newuser";
// Supply the name of a mapping service that maps a workstation user
// ID to a z/OS RACF ID
String userRegistry = "registry";
// Do not supply any security token data to be traced.
byte[] userSecTkn = null;
// Do not supply a previous user ID.
String originalUser = null;
// Call getDB2Connection to get the connection object for the new
// user.
java.sql.Connection con =
  ((com.ibm.db2.jcc.DB2PooledConnection)pooledCon).getDB2Connection(
     cookie,newuser,password,userRegistry,userSecTkn,originalUser,properties);
```

SSL 用の IBM Data Server Driver for JDBC and SQLJ サポート

IBM Data Server Driver for JDBC and SQLJ では、Java Secure Socket Extension (JSSE) を介した Secure Sockets Layer (SSL) のサポートが提供されます。

DB2 for z/OS バージョン 9 以降、 DB2 Database for Linux, UNIX, and Windows バージョン 9.1 フィックスパック 2 以降、または IBM Informix Dynamic Server (IDS) バージョン 11.50 以降用の IBM Data Server Driver for JDBC and SQLJ Type 4 接続 を使用している場合は、Java アプリケーション内で SSL サポートを使用することができます。

DB2 for z/OS データ・ソースへの接続に SSL サポートを使用する場合、z/OS バージョンが V1.8、V1.9、または V1.10 のときは、APAR PK72201 の該当 PTF を Communication Server for z/OS IP Services に適用する必要があります。

SSL 接続を使用するには、以下を行う必要があります。

- SSL を使用するように、データ・ソースへの接続を構成する。
- SSL を使用するように、Java Runtime Environment を構成する。

SSL を使用するように IBM Data Server Driver for JDBC and SQLJ での接続を構成する

SSL を使用するように IBM Data Server Driver for JDBC and SQLJ でのデータベース接続を構成するには、DB2BaseDataSource.sslConnection プロパティーを `true` に設定する必要があります。

前提条件: データ・ソースへの接続で SSL を使用できるようにするには、データベース・サーバー内で、アプリケーションが接続するポートを SSL リスナー・ポートとして構成しておく必要があります。

1. Connection または DataSource インスタンスで DB2BaseDataSource.sslConnection を設定します。
2. オプション: Connection または DataSource インスタンスで DB2BaseDataSource.sslTrustStoreLocation を設定して、トラストストアの場所を識別します。 sslTrustStoreLocation プロパティーの設定は、Javajavax.net.ssl.trustStore プロパティーの設定に代わるものです。 DB2BaseDataSource.sslTrustStoreLocation を設定すると、javax.net.ssl.trustStore は使用されません。
3. オプション: Connection または DataSource インスタンスで DB2BaseDataSource.sslTrustStorePassword を設定して、トラストストアのパスワードを識別します。 sslTrustStorePassword プロパティーの設定は、Java javax.net.ssl.trustStorePassword プロパティーの設定に代わるものです。 DB2BaseDataSource.sslTrustStorePassword を設定すると、javax.net.ssl.trustStorePassword は使用されません。

以下の例は、Connection インスタンスで sslConnection プロパティーを設定する方法を示しています。

```
java.util.Properties properties = new java.util.Properties();
properties.put("user", "xxxx");
properties.put("password", "yyyy");
```

```
properties.put("sslConnection", "true");
java.sql.Connection con =
  java.sql.DriverManager.getConnection(url, properties);
```

SSL を使用するように、Java Runtime Environment を構成する

JDBC および SQLJ アプリケーションで Secure Sockets Layer (SSL) 接続を使用するには、SSL を使用するように、Java Runtime Environment を構成しておく必要があります。

SSL 用に Java Runtime Environment を構成する前に、以下の前提条件を満たしている必要があります。

- Java Runtime Environment には Java セキュリティー・プロバイダーが含まれていなければなりません。IBM JSSE プロバイダーまたは Sun JSSE プロバイダーがインストールされていなければなりません。IBM JSSE プロバイダーは IBM SDK for Java で自動的にインストールされます。

  制約事項: Sun JSSE プロバイダーを使用できるのは Sun Java Runtime Environment だけです。 Sun JSSE プロバイダーは IBM Java Runtime Environment では機能しません。
- SSL サポートがデータベース・サーバーで構成されていなければなりません。

SSL を使用するように、Java Runtime Environment を構成するには、以下のステップを実行します。

1. データベース・サーバーからクライアント上の Java トラストストアに証明書をインポートします。

   Java keytool ユーティリティーを使用して、証明書をトラストストアにインポートします。

   例えば、サーバー証明書が jcc.cacert というファイルに保管されているとします。ファイル jcc.cacert から証明書を読み取り、それを cacerts というトラストストアに保管するには、以下の keytool ユーティリティー・ステートメントを発行します。

   ```
   keytool -import -file jcc.cacert -keystore cacerts
   ```
2. 項目を java.security ファイルに追加して、Java セキュリティー・プロバイダー用に Java Runtime Environment を構成します。

   セキュリティー・プロバイダーの項目の形式は以下のとおりです。

   ```
   security.provider.n=provider-package-name
   ```

   n の値が小さいプロバイダーの方が、*n* の値が大きいプロバイダーよりも優先されます。

   追加する Java セキュリティー・プロバイダーの項目は、使用するのが IBM JSSE プロバイダーか Sun JSSE プロバイダーのどちらであるかによって異なります。

   - Sun JSSE プロバイダーを使用する場合、Sun セキュリティー・プロバイダーの項目を java.security ファイルに追加します。

- IBM JSSE プロバイダーを使用する場合、以下の方法のいずれかを使用します。
  - **IBMJSSE2 プロバイダーの使用 (IBM SDK for Java 1.4.2 以降でサポートされています)**

    推奨: IBMJSSE2 プロバイダーを FIPS モードで使用することをお勧めします。
    - FIPS 準拠モードで動作する必要がない場合
      - IBM SDK for Java 1.4.2 では、IBMJSSE2Provider の項目を java.security ファイルに追加します。IBMJCE プロバイダーの項目が java.security ファイルにあることを確認してください。IBM SDK for Java に付属している java.security ファイルには、IBMJCE の項目が含まれています。
      - それ以降のバージョンの IBM SDK for Java では、IBMJSSE2Provider および IBMJCE プロバイダーの項目が java.security ファイルにあることを確認してください。IBM SDK for Java に付属している java.security ファイルには、それらのプロバイダーの項目が含まれています。
    - FIPS 準拠モードで動作する必要がある場合
      - java.security ファイルで、IBMJCE プロバイダーの項目の前に IBMJCEFIPS プロバイダーの項目を追加します。 IBMJCE プロバイダーの項目は除去しないでください。
      - IBMJSSE2 プロバイダーで FIPS モードを有効にします。ステップ 3 (231 ページ) を参照してください。
  - **IBMJSSE プロバイダーの使用 (IBM SDK for Java 1.4.2 でのみサポートされています)**
    - FIPS 準拠モードで動作する必要がない場合、IBMJSSEProvider および IBMJCE プロバイダーの項目が java.security ファイルにあることを確認してください。IBM SDK for Java に付属している java.security ファイルには、それらのプロバイダーの項目が含まれています。
    - FIPS 準拠モードで動作する必要がある場合、java.security ファイルで IBMJCE プロバイダーの項目の前に、 FIPS 対応プロバイダーである IBMJSSEFIPSProvider および IBMJCEFIPS プロバイダーの項目を追加します。

  制約事項: Solaris オペレーティング・システムで IBMJSSE プロバイダーを使用する場合、IBMJCE、IBMJCEFIPS、IBMJSSE、または IBMJSSE2 プロバイダーの項目の前に、SunJSSE プロバイダーの項目を含める必要があります。

例: FIPS 準拠モードで実行する必要があり、IBMJSSE2 プロバイダーで FIPS モードを有効にする場合、以下のような java.security ファイルを使用します。

```
# Set the Java security providers
security.provider.1=com.ibm.jsse2.IBMJSSEProvider2
security.provider.2=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
```

例: FIPS 準拠モードで実行し、IBMJSSE プロバイダーを使用する必要がある場合、以下のような java.security ファイルを使用します。

```
# Set the Java security providers
security.provider.1=com.ibm.fips.jsse.IBMJSSEFIPSProvider
security.provider.2=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
```

例: Sun JSSE プロバイダーを使用する場合、以下のような java.security ファイルを使用します。

```
# Set the Java security providers
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
security.provider.3=com.sun.crypto.provider.SunJCE
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
```

3. IBM Data Server Driver for JDBC and SQLJ を FIPS 準拠モードで使用することを計画している場合、com.ibm.jsse2.JSSEFIPS Java システム・プロパティーを以下のように設定する必要があります。

   ```
   com.ibm.jsse2.JSSEFIPS=true
   ```

   制約事項: 非 FIPS モードの JSSE アプリケーションは FIPS モードの JVM では実行できません。

   制約事項: IBMJSSE2 プロバイダーが FIPS モードで実行する場合、ハードウェアの暗号化を使用することはできません。

4. 項目を java.security ファイルに追加して、SSL ソケット・ファクトリー・プロバイダー用に Java Runtime Environment を構成します。

   SSL ソケット・ファクトリー・プロバイダーの項目の形式は以下のとおりです。

   ```
   ssl.SocketFactory.provider=provider-package-name
   ssl.ServerSocketFactory.provider=provider-package-name
   ```

   使用する Java セキュリティー・プロバイダー用に SSL ソケット・ファクトリー・プロバイダーを指定します。

   例: IBMJSSE2 プロバイダーで FIPS モードを有効にする場合、java.security ファイルで以下のように SSL ソケット・ファクトリー・プロバイダーの項目を含めます。

   ```
   # Set the SSL socket factory provider
   ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
   ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
   ```

   例: IBMJSSE プロバイダーで FIPS モードを有効にする場合、java.security ファイルで以下のように SSL ソケット・ファクトリー・プロバイダーの項目を含めます。

   ```
   # Set the SSL socket factory provider
   ssl.SocketFactory.provider=com.ibm.fips.jsse.JSSESocketFactory
   ssl.ServerSocketFactory.provider=com.ibm.fips.jsse.JSSEServerSocketFactory
   ```

   例: Sun JSSE プロバイダーを使用する場合、以下のように SSL ソケット・ファクトリー・プロバイダーの項目を含めます。

```
# Set the SSL socket factory provider
ssl.SocketFactory.provider=com.sun.net.ssl.internal.ssl.SSLSocketFactoryImpl
ssl.ServerSocketFactory.provider=com.sun.net.ssl.internal.ssl.SSLServerSocketFactoryImpl
```

5. トラストストアを使用するように Java システム・プロパティーを構成します。

   これを行うには、以下の Java システム・プロパティーを設定します。

   javax.net.ssl.trustStore
   ステップ 1 (229 ページ) で keytool ユーティリティーの -keystore パラメーターで指定した、トラストストアの名前を指定します。

   IBM Data Server Driver for JDBC and SQLJ プロパティー DB2BaseDataSource.sslTrustStoreLocation が設定されている場合、この値によって javax.net.ssl.trustStore プロパティーの値がオーバーライドされます。

   javax.net.ssl.trustStorePassword (オプション)
   トラストストアのパスワードを指定します。トラストストアのパスワードの設定は必須ではありません。ただし、パスワードを設定しないと、トラストストアの保全性を保護することはできません。

   IBM Data Server Driver for JDBC and SQLJ プロパティー DB2BaseDataSource.sslTrustStorePassword が設定されている場合、この値によって javax.net.ssl.trustStorePassword プロパティーの値がオーバーライドされます。

   例: Java システム・プロパティーを設定する方法の 1 つは、 Java アプリケーションを実行するときに、それらを -D オプションの引数として指定することです。 SSL 接続を使用してデータ・ソースにアクセスする、MySSL.java という Java アプリケーションを実行するとします。 cacerts というトラストストアを定義しました。そのアプリケーションを実行するときに、以下のコマンドでトラストストア名を設定します。

   ```
   java -Djavax.net.ssl.trustStore=cacerts MySSL
   ```

IBM Data Server Driver for JDBC and SQLJ 使用時の SQLJ アプリケーション準備のセキュリティー

SQLJ アプリケーションの準備中にセキュリティーを提供する 2 つの方法があります。ユーザーにアプリケーションのカスタマイズのみ許可するという方法と、カスタマイズ中に特定の表のセットに対するアクセスを制限するという方法です。

ユーザーにカスタマイズのみ許可する

一群のユーザーに SQLJ アプリケーションのカスタマイズを許可し、それらのアプリケーションのバインドまたは実行を許可しないようにするには、以下のいずれかの手法を使用できます。

- **カスタマイズ専用のデータベース・システムを作成します (推奨ソリューション):** 以下のステップを実行してください。
  1. 新規のデータベース・マネージャー・インスタンスを作成します。これはカスタマイズ専用システムです。
  2. カスタマイズ専用システムでは、SQLJ アプリケーションがアクセスするすべての表およびビューを定義します。表またはビューの定義は、アプリケーショ

ンがバインドされて実行するデータベース・マネージャー・インスタンス (「バインドおよび実行」システム) 上の定義と同じでなければなりません。表またはビューに対して DESCRIBE ステートメントを実行した場合、カスタマイズ専用システムと「バインドおよび実行」システムでは同じ結果にならなければなりません。

3. カスタマイズ専用システムでは、必要な表またはビュー特権を、SQLJ アプリケーションをカスタマイズするユーザーに付与します。
4. カスタマイズ専用システムでは、ユーザーは -compile=true オプションを指定して sqlj コマンドを実行し、プログラム用の Java バイト・コードおよびシリアライズド・プロファイルを作成します。次に、-automaticbind NO オプションを指定して db2sqljcustomize コマンドを実行し、カスタマイズ済みシリアライズド・プロファイルを作成します。
5. java バイトコード・ファイルおよびカスタマイズ済みシリアライズド・プロファイルを「バインドおよび実行」システムにコピーします。
6. 「バインドおよび実行」システムでパッケージをバインドする権限を持つユーザーは、カスタマイズ専用システムからコピーされたカスタマイズ済みシリアライズド・プロファイルに対して db2sqljbind コマンドを実行します。

- **ストアード・プロシージャーを使用してカスタマイズを行います:** エンド・ユーザーのためにシリアライズド・プロファイルをカスタマイズし、SQLJ アプリケーションのパッケージをバインドする Java ストアード・プロシージャーを作成します。この Java ストアード・プロシージャーは、DYNAMICRULES オプションのいずれかを指定してバインドされた JDBC ドライバー・パッケージを使用する必要があります。このオプションを指定すると、動的 SQL はエンド・ユーザーの許可 ID とは別のユーザー ID で実行されます。例えば、DYNAMICRULES オプションの DEFINEBIND または DEFINERUN を使用して、Java ストアード・プロシージャーの作成者の許可 ID で動的 SQL を実行する場合があります。その場合、ストアード・プロシージャーでの EXECUTE 権限を、SQLJ カスタマイズを実行する必要があるユーザーに付与する必要があります。

  ストアード・プロシージャーは以下の処理を実行します。

  1. コンパイルされた SQLJ プログラムおよびシリアライズド・プロファイルを BLOB 入力パラメーターで受け取ります。
  2. 入力パラメーターをそのファイル・システムにコピーします。
  3. db2sqljcustomize を実行して、シリアライズド・プロファイルをカスタマイズし、SQLJ プログラム用のパッケージをバインドします。
  4. カスタマイズ済みのシリアライズド・プロファイルを出力パラメーターで戻します。

- **スタンドアロン・プログラムを使用してカスタマイズを行います:** この手法では、エンド・ユーザーのためにシリアライズド・プロファイルをカスタマイズし、SQLJ アプリケーションのパッケージをバインドする Java ストアード・プロシージャーと同じステップを実行するプログラムを作成することを伴います。ただし、プログラムをストアード・プロシージャーとして実行する代わりに、プログラムをライブラリー・サーバーの下でスタンドアロン・プログラムとして実行します。

カスタマイズ中の表アクセスの制限

シリアライズド・プロファイルをカスタマイズする場合、オンライン検査を実行して、プログラムがアクセスする表列のデータ・タイプおよび長さに関するアプリケーション・プログラム情報を提供する必要があります。デフォルトでは、カスタマイズにはオンライン検査が含まれています。

オンライン検査では、シリアライズド・プロファイルをカスタマイズするユーザーが、SQLJ プログラム内の SQL ステートメントに対して PREPARE および DESCRIBE ステートメントを実行する許可を持っている必要があります。その許可には、SQL ステートメントでアクセスされる表およびビューに対する SELECT 特権が含まれます。 SQL ステートメントに未修飾の表名が含まれる場合、オンライン検査中に使用される修飾子は、db2sqljcustomize -qualifier パラメーターの値になります。そのため、SQLJ アプリケーション内の未修飾名を持つ表およびビューをオンライン検査する場合、-qualifier パラメーターの値に一致する修飾子を持つ表およびビューに対してのみ SELECT 特権を付与できます。

第 6 章 DB2 JDBC Type 2 ドライバー使用時のセキュリティー

Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) では、ユーザー ID およびパスワード・セキュリティーがサポートされます。

ユーザー ID とパスワードを設定するか、あるいはそれらを両方とも設定しないようにしなければなりません。ユーザー ID とパスワードを設定しない場合、ドライバーは、オペレーティング・システムに現在ログオンしているユーザーのユーザー ID とパスワードを使用します。

JDBC 接続で、ユーザー ID およびパスワード・セキュリティーを指定する場合、以下のいずれかの技法を使用します。

DriverManager インターフェースの場合: DriverManager.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。以下に例を示します。

```
import java.sql.*;        // JDBC base
...
String id = "db2adm";      // Set user ID
Sring pw = "db2adm";       // Set password
String url = "jdbc:db2:toronto";
                           // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                           // Create connection
```

別の方法として、 Properties オブジェクトに user および password プロパティーを設定し、 Properties オブジェクトをパラメーターとして組み込んだフォームの getConnection メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 JDBC Type 2 driver
                                            // implementation of JDBC
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
String url = "jdbc:db2:toronto";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

DataSource インターフェースの場合: DataSource.getConnection 呼び出しで、ユーザー ID およびパスワードを直接指定することができます。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 JDBC Type 2 driver
                                            // implementation of JDBC
...
Context ctx=new InitialContext();           // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Get DataSource object
String id = "db2adm";                       // Set user ID
```

```
Sring pw = "db2adm";                        // Set password
Connection con = ds.getConnection(id, pw);
                                            // Create connection
```

DataSource オブジェクトを作成してデプロイする場合、 DataSource オブジェクトの作成後に、 DataSource.setUser および DataSource.setPassword メソッドを呼び出して、ユーザー ID およびパスワードを設定できます。以下に例を示します。

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 JDBC Type 2 driver
                                            // implementation of JDBC
...
DB2DataSource db2ds = new DB2DataSource();
                                              // Create DataSource object
db2ds.setDatabaseName("toronto");             // Set location
db2ds.setUser("db2adm");                      // Set user ID
db2ds.setPassword("db2adm");                  // Set password
```

第 7 章 Java データベース・アプリケーションの構築

JDBC と SQLJ のデータベース・アプリケーションは、手動で構築できます。別の方法として、Java makefile を使用して JDBC アプリケーションを構築したり、DB2 Database for Linux, UNIX, and Windows に用意されている bldsqlj 構築ファイルを使用して SQLJ アプリケーションを構築したりすることも可能です。

JDBC アプレットの構築

JDBC アプリケーションを構築するには、Java `makefile` を使用するか、または手動で javac コマンドを実行することができます。

以下のステップは、`Applt.java` サンプル JDBC アプレットを構築して実行する方法を示しています。

1. 以下のコマンドで、`Applt.java` をコンパイルして、ファイル `Applt.class` を作成します。

   ```
   javac Applt.java
   ```

2. 作業ディレクトリーが、Web ブラウザーまたは (使用する場合は) Java アプレット・ビューアーからアクセス可能であることを確認します。ディレクトリーがアクセスできない場合は、以下のファイルをアクセス可能なディレクトリーにコピーします。
   - Applt.html
   - Applt.class
3. `sqllib¥java¥db2jcc.jar` (Windows の場合) または `sqllib/java/db2jcc.jar` (UNIX の場合) を、`Applt.class` および `Applt.html` と同じディレクトリーにコピーします。

   JDBC 4.0 の機能を使用する場合は、db2jcc.jar ではなく db2jcc4.jar をコピーします。
4. IBM Data Server Driver for JDBC and SQLJ を使用する場合、ファイルに記載されている手順に従って、`Applt.html` ファイルを変更してこのドライバーで接続します。 TCP/IP ポート番号には、データベース・ポート番号 50000 を使用する必要があります。
5. このアプレットを実行するには、Web サーバーが DB2 マシン上にインストール済みで実行中であることを確認するか、または以下のコマンドをクライアント・マシンの作業ディレクトリーで入力して、SDK for Java に付属しているアプレット・ビューアーを使用することができます。

   ```
   appletviewer Applt.html
   ```

JDBC アプリケーションの構築

JDBC アプリケーションを構築するには、Java `makefile` を使用するか、または手動で javac コマンドを実行することができます。

以下のステップは、DbInfo サンプル JDBC アプリケーションを構築して実行する方法を示しています。

1. 以下のコマンドで、DbInfo.java をコンパイルして、ファイル DbInfo.class を作成します。

   ```
   javac DbInfo.java
   ```

2. Java アプリケーションを UNIX 上の 64 ビットの DB2 インスタンスで実行しているが、Software Development Kit for Java が 32 ビットである場合、アプリケーションを実行する前に DB2 ライブラリー・パスを変更する必要があります。例えば、AIX 上では以下のようにします。
   - bash または Korn シェルを使用している場合:

     ```
     export LIBPATH=$HOME/sqllib/lib32
     ```

   - C シェルを使用している場合:

     ```
     setenv LIBPATH $HOME/sqllib/lib32
     ```

3. Java インタープリターをアプリケーション上で、以下のコマンドで実行します。

   ```
   java DbInfo
   ```

JDBC ルーチンの構築

JDBC ルーチンを構築するには、Java makefile または javac コマンドを使用できます。これらのルーチンを構築した後で、それらをカタログする必要があります。

以下のステップは、これらのルーチンを構築して実行する方法を示しています。

- SpServer のサンプル JDBC ストアード・プロシージャー
- UDFsrv のサンプル・ユーザー定義関数。これには SQL ステートメントが入っていません。
- UDFsqlsv のサンプル・ユーザー定義関数。これには SQL ステートメントが入っています。
- SpServer.java ストアード・プロシージャーをサーバー上で構築して実行するには、コマンド行から以下のようにします。
  1. 以下のコマンドで、SpServer.java をコンパイルして、ファイル SpServer.class を作成します。

     ```
     javac SpServer.java
     ```

  2. SpServer.class を、Windows オペレーティング・システム上では sqllib¥function ディレクトリーに、UNIX 上では sqllib/function ディレクトリーにコピーします。
  3. サーバーで spcat スクリプトを実行してルーチンをカタログします。 spcat スクリプトは、サンプル・データベースに接続し、ルーチンが既にカタログ済みであれば SpDrop.db2 を呼び出してルーチンをアンカタログし、次に SpCreate.db2 を呼び出してそのルーチンをカタログし、そして最後にデータベースへの接続を切断します。また、SpDrop.db2 スクリプトと SpCreate.db2 スクリプトは、個別に実行することもできます。
  4. データベースを一度停止してから再始動し、新しいクラス・ファイルが認識されるようにします。必要であれば、クラス・ファイルのファイル・モードを「読み取り」に設定して、fenced ユーザーが読み取ることができるようにします。

5. SpClient クライアント・アプリケーションをコンパイルして実行し、ストアード・プロシージャー・クラスにアクセスします。

- UDFsrv.java ユーザー定義関数プログラム (SQL ステートメントがないユーザー定義関数) をサーバー上で構築して実行するには、コマンド行から以下のようにします。
  1. 以下のコマンドで、UDFsrv.java をコンパイルして、ファイル UDFsrv.class を作成します。

     ```
     javac UDFsrv.java
     ```

  2. UDFsrv.class を、Windows オペレーティング・システム上では sqllib¥function ディレクトリーに、UNIX 上では sqllib/function ディレクトリーにコピーします。
  3. UDFsrv を呼び出すクライアント・プログラムをコンパイルして実行します。

     UDFsrv ライブラリーにアクセスするには、UDFcli.java JDBC アプリケーションまたは UDFcli.sqlj SQLJ クライアント・アプリケーションを使用できます。どちらのバージョンのクライアント・プログラムにも、ユーザー定義関数をデータベースで登録するために使用する CREATE FUNCTION SQL ステートメントと、ユーザー定義関数を使用する SQL ステートメントが含まれています。
- UDFsqlsv.java ユーザー定義関数プログラム (SQL ステートメントを含むユーザー定義関数) をサーバー上で構築して実行するには、コマンド行から以下のようにします。
  1. 以下のコマンドで、UDFsqlsv.java をコンパイルして、ファイル UDFsqlsv.class を作成します。

     ```
     javac UDFsqlsv.java
     ```

  2. UDFsqlsv.class を、Windows オペレーティング・システム上では sqllib¥function ディレクトリーに、UNIX 上では sqllib/function ディレクトリーにコピーします。
  3. UDFsqlsv を呼び出すクライアント・プログラムをコンパイルして実行します。

     UDFsqlsv ライブラリーにアクセスするには、UDFsqlcl.java JDBC アプリケーションを使用できます。クライアント・プログラムには、ユーザー定義関数をデータベースで登録するために使用する CREATE FUNCTION SQL ステートメントと、ユーザー定義関数を使用する SQL ステートメントが含まれています。

SQLJ アプレットの構築

SQLJ アプレットを構築するには、Java makefile または bldsqlj ビルド・ファイルを使用できます。

以下のステップは、Applt サンプル SQLJ アプレットを構築して実行する方法を示しています。これらのステップは、ビルド・ファイル bldsqlj (UNIX) または bldsqlj.bat (Windows) を使用します。これには SQLJ アプレットまたはアプリケーションのいずれかを構築するためのコマンドが含まれています。

ビルド・ファイルは最大で 6 つのパラメーターを取ります。これは UNIX 上では $1、$2、$3、$4、$5、および $6、Windows 上では %1、%2、%3、%4、%5、および %6 です。最初のパラメーターはプログラムの名前を指定します。 2 番目のパラメーターはデータベース・インスタンスのユーザー ID を指定し、3 番目のパラメーターはパスワードを指定します。 4 番目のパラメーターはサーバー名を指定します。 5 番目のパラメーターはポート番号を指定します。 6 番目のパラメーターはデータベース名を指定します。最初のパラメーター (プログラム名) を除く他のすべてについては、デフォルト値を使用できます。デフォルトのパラメーター値の使用について詳しくは、ビルド・ファイルを参照してください。

1. アプレットをこのコマンドで構築するには、以下のようにします。

   ```
   bldsqlj Applt <userid> <password> <server_name> <port_number> <db_name>
   ```

2. 作業ディレクトリーが、Web ブラウザーまたは (使用する場合は) Java アプレット・ビューアーからアクセス可能であることを確認します。ディレクトリーがアクセスできない場合は、以下のファイルをアクセス可能なディレクトリーにコピーします。
   - Applt.html
   - Applt.class
   - Applt_Cursor1.class
   - Applt_Cursor2.class
   - Applt_SJProfileKeys.class
   - Applt_SJProfile0.ser
3. `sqllib¥java¥db2jcc.jar` (Windows の場合) または `sqllib/java/db2jcc.jar` (UNIX の場合) を、`Applt.class` および `Applt.html` と同じディレクトリーにコピーします。

   JDBC 4.0 の機能を使用する場合は、db2jcc.jar ではなく db2jcc4.jar をコピーします。
4. IBM Data Server Driver for JDBC and SQLJ を使用する場合、ファイルに記載されている手順に従って、`Applt.html` ファイルを変更してこのドライバーで接続します。 TCP/IP ポート番号には、データベース・ポート番号 50000 を使用する必要があります。
5. このアプレットを実行するには、Web サーバーが DB2 マシン上にインストール済みで実行中であることを確認するか、または以下のコマンドをクライアント・マシンの作業ディレクトリーで入力して、SDK for Java に付属しているアプレット・ビューアーを使用することができます。

   ```
   appletviewer Applt.html
   ```

SQLJ アプリケーションの構築

SQLJ アプリケーションを構築するには、Java `makefile` または `bldsqlj` ビルド・ファイルを使用できます。

以下のステップは、`TbMod` サンプル SQLJ アプリケーションを構築して実行する方法を示しています。これらのステップは、ビルド・ファイル `bldsqlj` (UNIX) または `bldsqlj.bat` (Windows) を使用します。これには SQLJ アプレットまたはアプリケーションのいずれかを構築するためのコマンドが含まれています。

ビルド・ファイルは最大で 6 つのパラメーターを取ります。これは UNIX 上では $1、$2、$3、$4、$5、および $6、Windows 上では %1、%2、%3、%4、%5、および %6 です。最初のパラメーターはプログラムの名前を指定します。 2 番目のパラメーターはデータベース・インスタンスのユーザー ID を指定し、3 番目のパラメーターはパスワードを指定します。 4 番目のパラメーターはサーバー名を指定します。 5 番目のパラメーターはポート番号を指定します。 6 番目のパラメーターはデータベース名を指定します。最初のパラメーター (プログラム名) を除く他のすべてについては、デフォルト値を使用できます。デフォルトのパラメーター値の使用について詳しくは、ビルド・ファイルを参照してください。

1. アプリケーションをこのコマンドで構築するには、以下のようにします。

   ```
   bldsqlj TbMod <userid> <password> <server_name> <port_number> <db_name>
   ```

2. Java アプリケーションを UNIX 上の 64 ビットの DB2 インスタンスで実行しているが、Software Development Kit for Java が 32 ビットである場合、アプリケーションを実行する前に DB2 ライブラリー・パスを変更する必要があります。例えば、AIX 上では以下のようにします。
   - bash または Korn シェルを使用している場合:

     ```
     export LIBPATH=$HOME/sqllib/lib32
     ```

   - C シェルを使用している場合:

     ```
     setenv LIBPATH $HOME/sqllib/lib32
     ```

3. Java インタープリターをアプリケーション上で、以下のコマンドで実行します。

   ```
   java TbMod
   ```

Java アプレットに関する考慮事項

DB2 データベースには、Java アプレットを使用してアクセスすることができます。

アプレットを使用する場合は、以下の点に注意してください。

- いくつかの Java クラスから構成される比較的大きな JDBC または SQLJ アプレットの場合は、そのすべてのクラスを単一の JAR ファイル内にパッケージすることができます。また、SQLJ アプレットの場合、シリアライズド・プロファイルをそのクラスと共にパッケージする必要もあります。このパッケージ化を行う場合は、JAR ファイルを「applet」タグ内の `archive` パラメーターに追加します。詳細については、ご使用の Software Development Kit for Java の資料を参照してください。

  SQLJ アプレットの場合、一部のブラウザーには、アプレットに関連付けられたリソース・ファイルからシリアライズド・オブジェクトを読み込むためのサポートがまだありません。例えば、このようなブラウザーで付属のサンプル・アプレット `Applt` を読み込もうとすると、次のエラー・メッセージが表示されます。

  ```
  java.lang.ClassNotFoundException: Applt_SJProfile0
  ```

  この回避策として、シリアライズド・プロファイルを Java クラス形式で保管されたプロファイルに変換するユーティリティーがあります。このユーティリティーは、sqlj.runtime.profile.util.SerProfileToClass という名前の Java クラスです。このユーティリティーでは、シリアライズド・プロファイルのリソース・ファイル

が入力として読み込まれ、このプロファイルが含まれた Java クラスが出力として生成されます。プロファイルは以下のいずれかのコマンドを使用して変換できます。

```
profconv Applt_SJProfile0.ser
```

または

```
java sqlj.runtime.profile.util.SerProfileToClass Applt_SJProfile0.ser
```

このコマンドの結果、クラス Applt_SJProfile0.class が生成されます。アプレットで使用される .ser 形式のすべてのプロファイルを .class 形式のプロファイルに置き換えれば、この問題は解消されるはずです。

- db2jcc.jar というファイルを、ご使用の Web サイトから読み込まれる可能性があるいくつかのアプレットによって共有される特定のディレクトリーに配置することができます。 db2jcc.jar は、 IBM Data Server Driver for JDBC and SQLJ を使用するアプレットまたは SQLJ アプレットのためのものです。このファイルは、Windows オペレーティング・システムの場合には sqllib¥java ディレクトリーに、UNIX の場合には sqllib/java ディレクトリーにあります。このディレクトリーを特定するために、codebase パラメーターを HTML ファイル内の「applet」タグに追加する必要がある場合もあります。詳細については、ご使用の Software Development Kit for Java の資料を参照してください。

  JDBC 4.0 の機能を使用する場合は、db2jcc.jar ではなく db2jcc4.jar をコピーしてください。
- JDBC アプレット・サーバー (リスナー) である db2jd には、サーバーを堅固にするためのシグナル処理が備わっています。このため、CTRL-C キー・シーケンスを使用して、db2jd を終了することはできません。したがって、リスナーを終了させる唯一の方法は、kill -9 (UNIX の場合) または「タスク マネージャ」(Windows の場合) を使用してこのプロセスを強制終了することです。

UNIX の SQLJ アプリケーションおよびアプレット・オプション

bldsqlj ビルド・スクリプトは、UNIX オペレーティング・システムで SQLJ アプリケーションおよびアプレットを構築します。 bldsqlj は SQLJ 変換プログラムおよびカスタマイザー・オプションのセットを指定します。

推奨: UNIX プラットフォームで SQLJ アプリケーションおよびアプレットを構築する際に bldsqlj が使用するものと同じ SQLJ 変換プログラムおよびカスタマイザー・オプションを使用してください。

bldsqlj が含んでいるオプションは、以下のとおりです。

sqlj
SQLJ 変換プログラム (プログラムのコンパイルも行います)。

"${progname}.sqlj"
SQLJ ソース・ファイル。拡張子が入力ファイル名に含まれていた場合、progname=${1%.sqlj} コマンドはその拡張子を除去します。そのため、拡張子を再び追加しても重複することはありません。

db2sqljcustomize
SQLJ プロファイル・カスタマイザー。

-url データベース接続を確立するための JDBC URL を指定します (jdbc:db2://servername:50000/sample など)。

-user ユーザー ID を指定します。

-password
パスワードを指定します。

"${progname}_SJProfile0"
プログラムのシリアライズド・プロファイルを指定します。

Windows の SQLJ アプリケーションおよびアプレット・オプション

bldsqlj.bat バッチ・ファイルは、Windows オペレーティング・システムで SQLJ アプリケーションおよびアプレットを構築します。bldsqlj.bat は SQLJ 変換プログラムおよびカスタマイザー・オプションのセットを指定します。

推奨: Windows オペレーティング・システムで SQLJ アプリケーションおよびアプレットを構築する際に bldsqlj.bat が使用するものと同じ SQLJ 変換プログラムおよびカスタマイザー・オプションを使用してください。

bldsqlj.bat が含んでいるオプションは、以下のとおりです。

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

%1.sqlj
SQLJ ソース・ファイル。

db2sqljcustomize
SQLJ プロファイル・カスタマイザー。

-url データベース接続を確立するための JDBC URL を指定します (jdbc:db2://servername:50000/sample など)。

-user ユーザー ID を指定します。

-password
パスワードを指定します。

%1_SJProfile0
プログラムのシリアライズド・プロファイルを指定します。

SQL ルーチンの構築

SQLJ ルーチンを構築するには、Java makefile または bldsqljs ビルド・ファイルを使用できます。これらのルーチンを構築した後で、それらをカタログする必要があります。

以下のステップは、SpServer サンプル SQLJ ストアード・プロシージャーを構築して実行する方法を示しています。これらのステップは、ビルド・ファイル bldsqljs (UNIX) または bldsqljs.bat (Windows) を使用します。これには SQLJ アプレットまたはアプリケーションのいずれかを構築するためのコマンドが含まれています。

ビルド・ファイルは最大で 6 つのパラメーターを取ります。これは UNIX 上では $1、$2、$3、$4、$5、および $6、Windows 上では %1、%2、%3、%4、%5、およ

び %6 です。最初のパラメーターはプログラムの名前を指定します。 2 番目のパラメーターはデータベース・インスタンスのユーザー ID を指定し、3 番目のパラメーターはパスワードを指定します。 4 番目のパラメーターはサーバー名を指定します。 5 番目のパラメーターはポート番号を指定します。 6 番目のパラメーターはデータベース名を指定します。最初のパラメーター (プログラム名) を除く他のすべてについては、デフォルト値を使用できます。デフォルトのパラメーター値の使用について詳しくは、ビルド・ファイルを参照してください。

1. ストアード・プロシージャー・アプリケーションをこのコマンドで構築するには、以下のようにします。

   ```
   bldsqljs SpServer <userid> <password> <server_name> <port_number> <db_name>
   ```

2. ストアード・プロシージャーを以下のコマンドでカタログします。

   ```
   spcat
   ```

   このスクリプトは、サンプル・データベースに接続し、ルーチンが既にカタログ済みであれば SpDrop.db2 を呼び出してルーチンをアンカタログし、次に SpCreate.db2 を呼び出してそのルーチンをカタログし、そして最後にデータベースへの接続を切断します。また、SpDrop.db2 スクリプトと SpCreate.db2 スクリプトは、個別に実行することもできます。

3. データベースを一度停止してから再始動し、新しいクラス・ファイルが認識されるようにします。必要であれば、クラス・ファイルのファイル・モードを読み取りに設定して、fenced ユーザーが読み取ることができるようにします。
4. `SpClient` クライアント・アプリケーションをコンパイルして実行し、ストアード・プロシージャー・クラスにアクセスします。 `SpClient` は、アプリケーション・ビルド・ファイルの `bldsqlj` (UNIX) または `bldsqlj.bat` (Windows) で構築できます。

UNIX の SQLJ ルーチン・オプション

`bldsqljs` ビルド・スクリプトは、UNIX オペレーティング・システムで SQLJ ルーチンを構築します。 `bldsqljs` は SQLJ 変換プログラムおよびカスタマイザー・オプションのセットを指定します。

推奨: UNIX プラットフォームで SQLJ ルーチンを構築する際に `bldsqljs` が使用するものと同じ SQLJ 変換プログラムおよびカスタマイザー・オプションを使用してください。

`bldsqljs` が含んでいるオプションは、以下のとおりです。

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

"${progname}.sqlj"
SQLJ ソース・ファイル。拡張子が入力ファイル名に含まれていた場合、`progname=${1%.sqlj}` コマンドはその拡張子を除去します。そのため、拡張子を再び追加しても重複することはありません。

db2sqljcustomize
SQLJ プロファイル・カスタマイザー。

-url データベース接続を確立するための JDBC URL を指定します (`jdbc:db2://servername:50000/sample` など)。

-user ユーザー ID を指定します。

-password
パスワードを指定します。

"${progname}_SJProfile0"
プログラムのシリアライズド・プロファイルを指定します。

Windows の SQLJ ルーチン・オプション

bldsqljs.bat バッチ・ファイルは、Windows オペレーティング・システムで SQLJ ルーチンを構築します。 bldsqljs.bat は SQLJ 変換プログラムおよびカスタマイザー・オプションのセットを指定します。

推奨: Windows オペレーティング・システムで SQLJ ルーチンを構築する際に bldsqljs.bat が使用するものと同じ SQLJ 変換プログラムおよびカスタマイザー・オプションを使用してください。

以下の SQLJ 変換プログラムおよびカスタマイザー・オプションが、Windows オペレーティング・システム上の bldsqljs.bat バッチ・ファイルで使用されます。これらのオプションは、 DB2 が SQLJ ルーチン (ストアード・プロシージャーおよびユーザー定義関数) の構築に使用するように推奨しているものです。

sqlj SQLJ 変換プログラム (プログラムのコンパイルも行います)。

%1.sqlj
SQLJ ソース・ファイル。

db2sqljcustomize
DB2 for Java プロファイル・カスタマイザー。

-url データベース接続を確立するための JDBC URL を指定します (jdbc:db2://servername:50000/sample など)。

-user ユーザー ID を指定します。

-password
パスワードを指定します。

%1_SJProfile0
プログラムのシリアライズド・プロファイルを指定します。

第 8 章 IBM Data Server Driver for JDBC and SQLJ での問題診断

IBM Data Server Driver for JDBC and SQLJ での SQLJ または JDBC 問題を診断するためのデータを取得するには、トレース・データを収集し、そのトレース・データをフォーマットするユーティリティーを実行します。

トレースおよび診断ユーティリティーは、IBM ソフトウェア・サポートの指示の下でのみ実行してください。

JDBC トレース・データの収集

以下の手順のいずれかを使用して、トレースを開始します。

手順 1: DB2 for Linux, UNIX and Windows 用の IBM Data Server Driver for JDBC and SQLJ Type 4 接続 または IBM Data Server Driver for JDBC and SQLJ Type 2 接続の場合、推奨されるメソッドは、 IBM Data Server Driver for JDBC and SQLJ 構成プロパティー・ファイルで db2.jcc.override.traceFile プロパティーまたは db2.jcc.override.traceDirectory プロパティーを設定して、トレースを開始することです。ドライバーの開始前に db2.jcc.tracePolling および db2.jcc.tracePollingInterval プロパティーを設定すれば、ドライバーの実行中にグローバル構成トレース・プロパティーを変更することができます。

手順 2: DataSource インターフェースを使用してデータ・ソースに接続する場合、次の方法で、トレースを開始します。

1. DB2BaseDataSource.setTraceLevel メソッドを呼び出して、必要なトレースのタイプを設定します。デフォルトのトレース・レベルは `TRACE_ALL` です。複数のトレースのタイプを指定する方法については、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』を参照してください。
2. DB2BaseDataSource.setJccLogWriter メソッドを呼び出して、トレースの宛先を指定し、トレースをオンにします。

手順 3:

DataSource インターフェースを使用してデータ・ソースに接続する場合、javax.sql.DataSource.setLogWriter メソッドを呼び出して、トレースをオンにします。このメソッドを使用する場合、使用可能なトレース・レベルは `TRACE_ALL` のみです。

DriverManager インターフェースを使用してデータ・ソースに接続する場合、以下の手順を実行して、トレースを開始します。

1. traceLevel プロパティーを必要なトレースのタイプの *info* パラメーターまたは *url* パラメーターで設定して、`DriverManager.getConnection` メソッドを呼び出します。デフォルトのトレース・レベルは `TRACE_ALL` です。複数のトレースのタイプを指定する方法については、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』を参照してください。

2. DriverManager.setLogWriter メソッドを呼び出して、トレースの宛先を指定し、トレースをオンにします。

接続が確立された後、トレースをオフにしたりオンに戻したり、トレースの宛先を変更したり、DB2Connection.setJccLogWriter メソッドを使用してトレース・レベルを変更したりできます。トレースをオフにするには、logWriter 値を `null` に設定します。

logWriter プロパティーはタイプ java.io.PrintWriter のオブジェクトです。アプリケーションが java.io.PrintWriter オブジェクトを処理できない場合、`traceFile` プロパティーを使用してトレース出力の宛先を指定できます。 `traceFile` プロパティーを使用するには、logWriter プロパティーを `null` に設定し、`traceFile` プロパティーをドライバーがトレース・データを書き込むファイルの名前に設定します。このファイルおよびこのファイルが置かれているディレクトリーは書き込み可能でなければなりません。ファイルが既に存在している場合、ドライバーがこのファイルを上書きします。

手順 4: DriverManager インターフェースを使用している場合、ドライバーのロード時に URL の一部として `traceFile` および `traceLevel` プロパティーを指定します。以下に例を示します。

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose" +
 ":traceFile=/u/db2p/jcctrace;" +
 "traceLevel=" + com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS + ";";
```

手順 5: DB2TraceManager メソッドを使用します。 DB2TraceManager クラスは、すべてのタイプのログ書き込みプログラムによるトレースを中断および再開する機能を提供します。

構成プロパティーを使用してトレースを開始する例: 構成パラメーターを使用してトレース・データを収集する完全な例については、『構成プロパティーを使用して JDBC トレースを開始する例』を参照してください。

トレース例プログラム: IBM Data Server Driver for JDBC and SQLJ でのトレース用のプログラムの完全な例については、『IBM Data Server Driver for JDBC and SQLJ 使用時のトレース・プログラムの例』を参照してください。

カスタマイズまたはバインド時の SQLJ トレース・データの収集

SQLJ のカスタマイズまたはバインド処理時に問題を診断するためにトレース・データを収集するには、db2sqljcustomize または db2sqljbind バインド・ユーティリティーの実行時に -tracelevel および -tracefile オプションを指定します。

SQLJ シリアライズド・プロファイルに関する情報のフォーマット

`profp` ユーティリティーは、シリアライズド・プロファイル内の各 SQLJ 節に関する情報をフォーマットします。 `profp` ユーティリティーのフォーマットは次のとおりです。

エラーが発生している、接続用のシリアライズド・プロファイルに対して profp ユーティリティーを実行します。例外がスローされる場合、Java スタック・トレースが生成されます。例外がスタック・トレースからスローされたときに、どのシリアライズド・プロファイルが使用されていたかを判別できます。

SQLJ カスタマイズ済みシリアライズド・プロファイルに関する情報のフォーマット

db2sqljprint ユーティリティーは、 IBM Data Server Driver for JDBC and SQLJ 用にカスタマイズされたシリアライズド・プロファイル内の各 SQLJ 節に関する情報をフォーマットします。

エラーが発生している、接続用のカスタマイズ済みシリアライズド・プロファイルに対して db2sqljprint ユーティリティーを実行します。

構成プロパティーを使用して JDBC トレースを開始する例

JDBC アプリケーションを変更せずに、そのアプリケーションのトレースを制御できます。

IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用する Test.java という名前のプログラムのトレース・データを収集すると仮定します。 Test.java はトレースを行いませんが、プログラムを変更するつもりはありません。そのため、構成プロパティーを使用してトレースを使用可能にします。トレース出力に以下の特性を持たせることにします。

- 同じ DataSource での各接続のトレース情報は別個のトレース・ファイルに書き込まれます。出力は /Trace という名前のディレクトリーに入れられます。
- 各トレース・ファイルの名前は jccTrace1 で始まります。
- 同じ名前のトレース・ファイルが既に存在する場合、トレース・データがそれらに追加されます。

Test1.java にはトレースを行うコードが含まれていませんが、今後トレースを行うようにアプリケーションが変更される場合にプログラム内の設定が構成プロパティーの設定より優先するように、その構成プロパティーを設定します。これを行うには、db2.jcc.override ではなく、db2.jcc で始まる構成プロパティーの設定を使用します。

構成プロパティー設定は次のようになります。

- db2.jcc.override.traceDirectory=/Trace
- db2.jcc.traceFile=jccTrace1
- db2.jcc.traceFileAppend=true

トレース設定がスタンドアロン・プログラム Test1.java にのみ適用されるようにします。そのため、そのような設定のファイルを作成し、-Ddb2.jcc.propertiesFile オプションを指定して Java プログラムを呼び出すときにそのファイルを参照するようにします。その設定を含むファイルが /Test/jcc.properties であるとします。 Test1.java の実行時にトレースを使用可能にするには、次のようなコマンドを発行します。

```
java -Ddb2.jcc.propertiesFile=/Test/jcc.properties Test1
```

Test1.java が 1 つの DataSource に 2 つの接続を作成するとします。プログラムは logWriter オブジェクトを定義しません。そのため、ドライバーはトレース出力用にグローバル logWriter オブジェクトを作成します。プログラムが完了すると、以下のファイルにトレース・データが含まれています。

- /Trace/jccTrace1_global_0
- /Trace/jccTrace1_global_1

IBM Data Server Driver for JDBC and SQLJ 使用時のトレース・プログラムの例

トレース用のメソッドを組み込んでいる単一のクラスを DriverManager インターフェースおよび DataSource インターフェースの下に作成することができます。

以下の例は、そのようなクラスを示しています。この例では、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続が使用されています。

図 52. IBM Data Server Driver for JDBC and SQLJ 使用時のトレースの例

```
public class TraceExample
{

  public static void main(String[] args)
  {
    sampleConnectUsingSimpleDataSource();
    sampleConnectWithURLUsingDriverManager();
  }

  private static void sampleConnectUsingSimpleDataSource()
  {
    java.sql.Connection c = null;
    java.io.PrintWriter printWriter =
     new java.io.PrintWriter(System.out, true);
                                       // Prints to console, true means
                                       // auto-flush so you don't lose trace
    try {
      javax.sql.DataSource ds =
       new com.ibm.db2.jcc.DB2SimpleDataSource();
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setPortNumber(5021);
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDatabaseName("san_jose");
      ((com.ibm.db2.jcc.DB2BaseDataSource) ds).setDriverType(4);

     ds.setLogWriter(printWriter);    // This turns on tracing

     // Refine the level of tracing detail
     ((com.ibm.db2.jcc.DB2BaseDataSource) ds).
      setTraceLevel(com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_CONNECTS |
      com.ibm.db2.jcc.DB2SimpleDataSource.TRACE_DRDA_FLOWS);

     // This connection request is traced using trace level
     // TRACE_CONNECTS | TRACE_DRDA_FLOWS
     c = ds.getConnection("myname", "mypass");

     // Change the trace level to TRACE_ALL
     // for all subsequent requests on the connection
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);
     // The following INSERT is traced using trace level TRACE_ALL
     java.sql.Statement s1 = c.createStatement();
     s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
```

```
   s1.close();

   // This code disables all tracing on the connection
   ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

   // The following INSERT statement is not traced
   java.sql.Statement s2 = c.createStatement();
   s2.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
   s2.close();

   c.close();
  }
  catch(java.sql.SQLException e) {
   com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
    printWriter, "[TraceExample]");
  }
  finally {
    cleanup(c, printWriter);
    printWriter.flush();
  }
}

// If the code ran successfully, the connection should
// already be closed. Check whether the connection is closed.
// If so, just return.
// If a failure occurred, try to roll back and close the connection.

private static void cleanup(java.sql.Connection c,
 java.io.PrintWriter printWriter)
{
  if(c == null) return;

  try {
   if(c.isClosed()) {
      printWriter.println("[TraceExample] " +
       "The connection was successfully closed");
      return;
   }

   // If we get to here, something has gone wrong.
   // Roll back and close the connection.
   printWriter.println("[TraceExample] Rolling back the connection");
   try {
     c.rollback();
   }
   catch(java.sql.SQLException e) {
     printWriter.println("[TraceExample] " +
      "Trapped the following java.sql.SQLException while trying to roll back:");
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
      "[TraceExample]");
     printWriter.println("[TraceExample] " +
      "Unable to roll back the connection");
    }
    catch(java.lang.Throwable e) {
      printWriter.println("[TraceExample] Trapped the " +
       "following java.lang.Throwable while trying to roll back:");
      com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e,
       printWriter, "[TraceExample]");
      printWriter.println("[TraceExample] Unable to " +
       "roll back the connection");
    }

    // Close the connection
    printWriter.println("[TraceExample] Closing the connection");
    try {
      c.close();
    }
```

```
    catch(java.sql.SQLException e) {
      printWriter.println("[TraceExample] Exception while " +
       "trying to close the connection");
      printWriter.println("[TraceExample] Deadlocks could " +
       "occur if the connection is not closed.");
      com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
       "[TraceExample]");
    }
    catch(java.lang.Throwable e) {
      printWriter.println("[TraceExample] Throwable caught " +
       "while trying to close the connection");
      printWriter.println("[TraceExample] Deadlocks could " +
       "occur if the connection is not closed.");
      com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
       "[TraceExample]");
    }
  }
  catch(java.lang.Throwable e) {
    printWriter.println("[TraceExample] Unable to " +
     "force the connection to close");
    printWriter.println("[TraceExample] Deadlocks " +
     "could occur if the connection is not closed.");
    com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
     "[TraceExample]");
  }
}
private static void sampleConnectWithURLUsingDriverManager()
{
  java.sql.Connection c = null;

  // This time, send the printWriter to a file.
  java.io.PrintWriter printWriter = null;
  try {
    printWriter =
      new java.io.PrintWriter(
        new java.io.BufferedOutputStream(
          new java.io.FileOutputStream("/temp/driverLog.txt"), 4096), true);
  }
  catch(java.io.FileNotFoundException e) {
    java.lang.System.err.println("Unable to establish a print writer for trace");
    java.lang.System.err.flush();
    return;
  }

  try {
    Class.forName("com.ibm.db2.jcc.DB2Driver");
  }
  catch(ClassNotFoundException e) {
    printWriter.println("[TraceExample] " +
     "IBM Data Server Driver for JDBC and SQLJ type 4 connectivity " +
     "is not in the application classpath. Unable to load driver.");
    printWriter.flush();
    return;
  }

  // This URL describes the target data source for Type 4 connectivity.
  // The traceLevel property is established through the URL syntax,
  // and driver tracing is directed to file "/temp/driverLog.txt"
  // The traceLevel property has type int. The constants
  // com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS and
  // com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS represent
  // int values. Those constants cannot be used directly in the
  // first getConnection parameter. Resolve the constants to their
  // int values by assigning them to a variable. Then use the
  // variable as the first parameter of the getConnection method.
  String databaseURL =
   "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
```

```
      "/sample:traceFile=/temp/driverLog.txt;traceLevel=" +
      (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS |
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS) + ";";

    // Set other properties
    java.util.Properties properties = new java.util.Properties();
    properties.setProperty("user", "myname");
    properties.setProperty("password", "mypass");

    try {
     // This connection request is traced using trace level
     // TRACE_CONNECTS | TRACE_DRDA_FLOWS
     c = java.sql.DriverManager.getConnection(databaseURL, properties);

     // Change the trace level for all subsequent requests
     // on the connection to TRACE_ALL
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(printWriter,
      com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);

     // The following INSERT is traced using trace level TRACE_ALL
     java.sql.Statement s1 = c.createStatement();
     s1.executeUpdate("INSERT INTO sampleTable(sampleColumn) VALUES(1)");
     s1.close();

     // Disable all tracing on the connection
     ((com.ibm.db2.jcc.DB2Connection) c).setJccLogWriter(null);

      // The following SQL insert code is not traced
      java.sql.Statement s2 = c.createStatement();
      s2.executeUpdate("insert into sampleTable(sampleColumn) values(1)");
      s2.close();

      c.close();
    }
    catch(java.sql.SQLException e) {
     com.ibm.db2.jcc.DB2ExceptionFormatter.printTrace(e, printWriter,
      "[TraceExample]");
    }
    finally {
      cleanup(c, printWriter);
      printWriter.flush();
    }
  }
}
```

IBM Data Server Driver for JDBC and SQLJ Sysplex サポートのモニター技法

IBM Data Server Driver for JDBC and SQLJ Sysplex サポートをモニターするには、グローバル・トランスポート・オブジェクト・プールをモニターする必要があります。

次のいずれかの方法で、グローバル・トランスポート・オブジェクト・プールをモニターすることができます。

- IBM Data Server Driver for JDBC and SQLJ の構成プロパティーを設定して開始するトレースの使用
- アプリケーション・プログラミング・インターフェースの使用

グローバル・トランスポート・オブジェクト・プールのモニターのための構成プロパティー

db2.jcc.dumpPool、db2.jcc.dumpPoolStatisticsOnSchedule、および db2.jcc.dumpPoolStatisticsOnScheduleFile 構成プロパティーにより、グローバル・トランスポート・オブジェクト・プールのトレースが制御されます。

例えば、以下の一組の構成プロパティーの設定では、エラー・メッセージとダンプ・プール・エラー・メッセージが 60 秒ごとに /home/WAS/logs/srv1/poolstats というファイルに書き込まれます。

```
db2.jcc.dumpPool=DUMP_SYSPLEX_MSG|DUMP_POOL_ERROR
db2.jcc.dumpPoolStatisticsOnSchedule=60
db2.jcc.dumpPoolStatisticsOnScheduleFile=/home/WAS/logs/srv1/poolstats
```

プール統計ファイル内の項目は次のようになります。

```
time Scheduled PoolStatistics npr:2575 nsr:2575 lwroc:439 hwroc:1764 coc:372
aooc:362 rmoc:362 nbr:2872 tbt:857520 tpo:10
```

各フィールドの意味は次のとおりです。

npr
プールの作成時以降にプールに対して IBM Data Server Driver for JDBC and SQLJ から出された要求の総数。

nsr
プールの作成時以降にプールに対して IBM Data Server Driver for JDBC and SQLJ から出されて正常に完了した要求の数。正常に完了した要求とは、プールからオブジェクトが返されたことを意味します。

lwroc
再利用されたがプール内に存在しなかったオブジェクトの数。これは、接続オブジェクトによってトランスポート・オブジェクトがトランザクション境界で解放された場合に発生することがあります。この接続オブジェクトで、そのトランスポート・オブジェクトが後で必要になった際に、元のトランスポート・オブジェクトが他のどの接続オブジェクトによってもまだ使用されていない場合は、この接続オブジェクトでそのトランスポート・オブジェクトを使用することができます。

hwroc
プールから再利用されたオブジェクトの数。

coc
プールの作成時以降に IBM Data Server Driver for JDBC and SQLJ により作成されたオブジェクトの数。

aooc
db2.jcc.maxTransportObjectIdleTime で指定されていたアイドル時間を超過したためにプールから削除されたオブジェクトの数。

rmoc
プールの作成時以降にプールから削除されたオブジェクトの数。

nbr
プールに対して IBM Data Server Driver for JDBC and SQLJ から出されたが、プールがその最大容量に達したためプールによってブロックされた要求の数。ブ

ロックされた要求が正常に完了する可能性があるのは、db2.jcc.maxTransportObjectWaitTime を超過して例外がスローされる前にオブジェクトがプールに返された場合です。

tbt プールによってブロックされた要求のミリ秒単位の合計時間。アプリケーションによって複数のスレッドが使用される場合は、この時間は、アプリケーションの経過実行時間よりはるかに長くなる可能性があります。

sbt
1 つのスレッドがプールからのトランスポート・オブジェクトの取得を待機したミリ秒単位の最短時間。この時間が 1 ミリ秒未満の場合、このフィールドの値はゼロになります。

lbt 1 つのスレッドがプールからのトランスポート・オブジェクトの取得を待機したミリ秒単位の最長時間。

abt
各スレッドがプールからのトランスポート・オブジェクトの取得を待機したミリ秒単位の平均時間。この値は tbt/nbr です。

tpo
プール内に現在存在するオブジェクトの数。

グローバル・トランスポート・オブジェクト・プールのモニターのためのアプリケーション・プログラミング・インターフェース

グローバル・トランスポート・オブジェクト・プールに関する統計を収集するためのアプリケーションを作成することができます。このようなアプリケーションでは、オブジェクトが DB2PoolMonitor クラスに作成され、プールに関する情報を取得するためにメソッドが呼び出されます。

例えば、次のコードでは、グローバル・トランスポート・オブジェクト・プールのモニター用のオブジェクトが作成されます。

```
import com.ibm.db2.jcc.DB2PoolMonitor;
DB2PoolMonitor transportObjectPoolMonitor =
        DB2PoolMonitor.getPoolMonitor (DB2PoolMonitor.TRANSPORT_OBJECT);
```

DB2PoolMonitor オブジェクトの作成後に、DB2PoolMonitor クラス内のメソッドを使用してプールをモニターすることができます。

第 9 章 IBM Data Server Driver for JDBC and SQLJ のシステム・モニター

IBM Data Server Driver for JDBC and SQLJ の使用時にアプリケーションのパフォーマンスをモニターする作業のために、このドライバーには、接続に関する情報を収集するためのメソッドが 2 つ用意されています。

その情報は、以下のとおりです。

コア・ドライバー時間
システム・モニターが有効になっている間に収集されたモニター対象の API の経過時間の合計 (マイクロ秒単位)。一般にモニター対象になるのは、ネットワーク I/O またはデータベース・サーバー対話が発生する可能性のある API だけです。

ネットワーク I/O 時間
システム・モニターが有効になっている間に収集されたネットワーク I/O の経過時間の合計 (マイクロ秒単位)。

サーバー時間
システム・モニターが有効になっている間に収集されたすべてのレポート対象データベース・サーバーの経過時間の合計 (マイクロ秒単位)。

現時点で、 IBM Informix Dynamic Server データベースはこの機能をサポートしていません。

アプリケーション時間
アプリケーション、JDBC ドライバー、ネットワーク I/O、データベース・サーバーの経過時間の合計 (ミリ秒単位)。

2 つのメソッドは、以下のとおりです。

- DB2SystemMonitor インターフェース
- TRACE_SYSTEM_MONITOR トレース・レベル

*DB2SystemMonitor インターフェースを使用してシステム・モニター・データを収集*するには、以下の基本手順を実行します。

1. DB2Connection.getDB2SystemMonitor メソッドを呼び出して、DB2SystemMonitor オブジェクトを作成します。
2. DB2SystemMonitor.enable メソッドを呼び出して、接続の DB2SystemMonitor オブジェクトを使用可能にします。
3. DB2SystemMonitor.start メソッドを呼び出して、システム・モニターを開始します。
4. モニター対象のアクティビティーが完了したら、DB2SystemMonitor.stop を呼び出して、システム・モニターを停止します。
5. DB2SystemMonitor.getCoreDriverTimeMicros、
   DB2SystemMonitor.getNetworkIOTimeMicros、

DB2SystemMonitor.getServerTimeMicros、
DB2SystemMonitor.getApplicationTimeMillis のいずれかのメソッドを呼び出して、経過時間データを取得します。

例えば、各タイプの経過時間データを収集するコードを以下に示します。選択されたステートメントの右にある番号は、以前に説明されたステップに対応しています。

```
import java.sql.*;
import com.ibm.db2.jcc.*;
public class TestSystemMonitor
{
  public static void main(String[] args)
  {
    String url = "jdbc:db2://sysmvs1.svl.ibm.com:5021/san_jose";
    String user="db2adm";
    String password="db2adm";
    try
    {
      // Load the IBM Data Server Driver for JDBC and SQLJ
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      System.out.println("**** Loaded the JDBC driver");

      // Create the connection using the IBM Data Server Driver for JDBC and SQLJ
      Connection conn = DriverManager.getConnection (url,user,password);
      // Commit changes manually
      conn.setAutoCommit(false);
      System.out.println("**** Created a JDBC connection to the data source");
      DB2SystemMonitor systemMonitor =                                    1
        ((DB2Connection)conn).getDB2SystemMonitor();
      systemMonitor.enable(true);                                         2
      systemMonitor.start(DB2SystemMonitor.RESET_TIMES);                  3
      Statement stmt = conn.createStatement();
      int numUpd = stmt.executeUpdate(
        "UPDATE EMPLOYEE SET PHONENO='4657' WHERE EMPNO='000010'");
      systemMonitor.stop();                                               4
      System.out.println("Server elapsed time (microseconds)="
        + systemMonitor.getServerTimeMicros());                           5
      System.out.println("Network I/O elapsed time (microseconds)="
        + systemMonitor.getNetworkIOTimeMicros());
      System.out.println("Core driver elapsed time (microseconds)="
        + systemMonitor.getCoreDriverTimeMicros());
      System.out.println("Application elapsed time (milliseconds)="
        + systemMonitor.getApplicationTimeMillis());
      conn.rollback();
      stmt.close();
      conn.close();
    }
    // Handle errors
    catch(ClassNotFoundException e)
    {
      System.err.println("Unable to load the driver, " + e);
    }
    catch(SQLException e)
    {
      System.out.println("SQLException: " + e);
      e.printStackTrace();
    }
  }
}
```

図 53. DB2SystemMonitor メソッドを使用してシステム・モニター・データを収集する例

*トレース・メソッドを使用してシステム・モニター情報を収集するには、*構成プロパティー、または Connection、DataSource のいずれかのプロパティーを使用して JDBC トレースを開始します。 traceLevel プロパティーを設定するときに、TRACE_SYSTEM_MONITOR を組み込んでください。以下に例を示します。

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose" +
 ":traceFile=/u/db2p/jcctrace;" +
 "traceLevel=" + com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR + ";";
```

システム・モニター情報のトレース・レコードは、以下のようになります。

```
[jcc][SystemMonitor:start]
...
[jcc][SystemMonitor:stop] core: 565.67ms | network: 211.695ms | server: 207.771ms
```

IBM Data Server Driver for JDBC and SQLJ のリモート・トレース・コントローラー

IBM Data Server Driver for JDBC and SQLJ では、 IBM Data Server Driver for JDBC and SQLJ のトレースを動的に制御するための機能が提供されます。

このリモート・トレース・コントローラーを使用すれば、複数のドライバー・インスタンス用のものと同様な以下の操作を実行することができます。

- トレースの開始、停止、または再開
- 出力トレース・ファイルまたはディレクトリーの場所の変更
- トレース・レベルの変更

リモート・トレース・コントローラーでは、Java Standard Edition バージョン 6 以降でその一部となっている Java Management Extensions (JMX) アーキテクチャーが使用されます。 JMX は、以下のものから構成されています。

- 組み込みの管理ユーティリティー一式 (これらを使用すると、Java Monitoring and Management Console (JConsole) などの管理コンソールからモニターを実行することができます)。
- 同じ機能を実行するためのアプリケーションを作成できる API 一式。

リモート・トレース・コントローラーの使用可能化

リモート・トレース・コントローラーを使用可能にするには、 IBM Data Server Driver for JDBC and SQLJ で Java Management Extensions (JMX) を使用可能にし、クライアントから JMX エージェントを使用できるようにする必要があります。

リモート・トレース・コントローラーでは、Java Standard Edition バージョン 6 以降が必要です。

リモート・トレース・コントローラーを使用可能にする手順は、以下のとおりです。

1. IBM Data Server Driver for JDBC and SQLJ で JMX を使用可能にするために、db2.jcc.jmxEnabled グローバル構成プロパティーを true または yes に設定します。

   例えば、DB2JccConfiguration.properties に以下のストリングを組み込みます。

```
db2.jcc.jmxEnabled=true
```

2. ローカル・クライアントまたはリモート・クライアントから JMX エージェント (プラットフォームの MBean サーバー) を使用できるようにします。
   - ローカル・クライアントの場合は、以下のようにします。

     JVM の始動時に、モニター機能と管理機能が自動的に使用可能になります。アプリケーションを開始した後に、JConsole などの JMX クライアントを使用して、Java プロセスにローカルで接続できます。
   - リモート・クライアントの場合は、以下のいずれかの方法を使用します。
     - すぐに使用可能な JMX エージェントを使用します。

       すぐに使用可能な管理機能では、JMX 組み込み管理ユーティリティーを使用します。すぐに使用可能な管理機能を有効にするには、いくつかの Java システム・プロパティーを設定する必要があります。少なくとも以下のプロパティーを設定しなければなりません。

       ```
       com.sun.management.jmxremote.port=portNum
       ```

       さらに、認証と SSL が正しく構成されていることを確認してください。

       すぐに使用可能な管理機能を有効にするための詳細については、以下の URL を参照してください。

       ```
       http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html
       ```
     - JMX エージェントを作成します。この技法については、以下の資料も参照できます。

       ```
       http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html
       ```

       以下の例では、MyCustomJMXAuthenticator オブジェクトを使用して、PlatformMBeanServer の RMI コネクター・サーバーを作成します。MyCustomJMXAuthenticator クラスでは、JMXAuthenticator インターフェースを実装して、リモート証明書を JAAS Subject に変換する方法を定義します。

       ```
       ...
       HashMap<String> env = new HashMap<String>();
       env.put(JMXConnectorServer.AUTHENTICATOR, new MyCustomJMXAuthenticator());
       env.put("jmx.remote.x.access.file", "my.access.file");

       MBeanServer mbs =
         java.lang.management.ManagementFactory.getPlatformMBeanServer();
       JMXServiceURL url =
         new JMXServiceURL("service:jmx:rmi:///jndi/rmi://:9999/jmxrmi");

       JMXConnectorServer cs =
         JMXConnectorServerFactory.newJMXConnectorServer(url, env, mbs);
       cs.start();
       ...
       public class MyCustomJMXAuthenticator implements JMXAuthenticator {

         public Subject authenticate(Object credentials) {
           // the hash contains username, password, etc...
           Hashtable <String> credentialsHash
             = (Hashtable <String>) credentials;

           ...
           // Authenticate using the provided credentials
       ```

```
        ...
        if (authentication-successful) {
          return new Subject(true,
            Collections.singleton
              (new JMXPrincipal(credentialsHash.get("username"))),
            Collections.EMPTY_SET,
            Collections.EMPTY_SET);
        }
        throw new SecurityException("Invalid credentials");
      }
    }
```

リモート・トレース・コントローラーへのアクセス

リモート・トレース・コントローラーにアクセスするには、すぐに使用可能な管理ツールを使用するか、アプリケーションを使用します。

すぐに使用可能な管理機能は、JConsole などの JMX 準拠の管理クライアントから使用できます。JConsole は、Java Standard Edition バージョン 6 に組み込まれています。すぐに使用可能な管理機能を JConsole から使用するための詳細については、以下の URL を参照してください。

`http://java.sun.com/javase/6/docs/technotes/guides/management/jconsole.html`

リモート・トレース・コントローラーにアクセスするアプリケーションでは、リモート・トレース・コントローラーが管理対象 Bean (MBean) になります。 JMX は、JMX エージェントによってリソースを管理します。JMX エージェントは、MBean サーバーです。それぞれの MBean は、1 つのリソースに対応します。すべての MBean には名前があり、その名前は、クラス javax.management.ObjectName のオブジェクトによって定義します。ObjectName オブジェクトを使用して、MBean を MBeanServer に登録したり、MBeanServer から MBean を取得したりすることも可能です。

MBean 名は、ドメインとキー・プロパティーという 2 つの部分からなっています。 IBM Data Server Driver for JDBC and SQLJ のリモート・トレース・コントローラーの ObjectName の場合、ドメインは com.ibm.db2.jcc、キー・プロパティーは name=DB2TraceManager になります。

リモート・トレース・コントローラーにアクセスするアプリケーションには、以下の手順を組み込む必要があります。

1. MBean サーバーに対するリモート・メソッド呼び出し (RMI) 接続を確立します。
2. MBean サーバーでリモート・トレース・コントローラーに対する検索を実行します。
3. MBean に対するトレース操作を呼び出します。

   MBean を操作するには、以下の方法があります。
   - MBean プロキシーを使用する方法
   - プロキシーを使用しないで MBeanServerConnection を使用する方法

例: プロキシーを使用しないでリモート・トレース・コントローラーにアクセスする方法: この例では、MBeanServerConnection から MBean に直接アクセスする方法

を示します。この方法は、JMX クライアント・アプリケーションでインターフェース定義を一致させる必要がないので非常に汎用的です。

```
Hashtable<String> env = new Hashtable<String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
 "com.sun.jndi.fscontext.RefFSContextFactory");

try {
  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Establish an RMI connection to an MBeanServer");
  System.out.println ("-------------------------------------------------");
  JMXServiceURL url =
    new JMXServiceURL ("service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi");
  JMXConnector jmxc = JMXConnectorFactory.connect (url, env);
  MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Processing MBean");
  System.out.println ("-------------------------------------------------");
  String objectNameString = "com.ibm.db2.jcc:name=DB2TraceManager";
  ObjectName name = new ObjectName(objectNameString);
  System.out.println ("ObjectName="+objectNameString);

  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Print all attributes of the MBean");
  System.out.println ("-------------------------------------------------");

  System.out.println(
    "TraceDirectory  = "+mbsc.getAttribute (name, "TraceDirectory"));
  System.out.println(
    "TraceFile       = "+mbsc.getAttribute (name, "TraceFile"));
  System.out.println(
    "TraceFileAppend = "+mbsc.getAttribute (name, "TraceFileAppend"));
  System.out.println(
    "TraceLevel      = "+mbsc.getAttribute (name, "TraceLevel"));

  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Invoke some operations on the MBean");
  System.out.println ("-------------------------------------------------");
  System.out.print ("Invoking suspendTrace()...");
  mbsc.invoke (name, "suspendTrace", null , null);
  System.out.println ("success");

  System.out.print ("Invoking resumeTrace()...");
  mbsc.invoke (name, "resumeTrace", null , null);
  System.out.println ("success");
}
catch (Exception e) {
  System.out.println ("failure");
  e.printStackTrace ();
}
```

例: プロキシーを使用してリモート・トレース・コントローラーにアクセスする方法: この例では、MBean のプロキシーを作成する方法を示します。このプロキシーでは、com.ibm.db2.jcc.mx.DB2TraceManagerMXBean インターフェースを実装します。アプリケーションは、そのプロキシーに対して呼び出しを直接実行し、基礎になっているプロキシー実装がリモート MBean サーバーに対する MBean 操作を呼び出します。

```
Hashtable<String> env = new Hashtable<String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
 "com.sun.jndi.fscontext.RefFSContextFactory");

try {
  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Establish an RMI connection to an MBeanServer");
  System.out.println ("-------------------------------------------------");
  JMXServiceURL url =
    new JMXServiceURL ("service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi");
  JMXConnector jmxc = JMXConnectorFactory.connect (url, env);
  MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Processing MBean");
  System.out.println ("-------------------------------------------------");
  String objectNameString = "com.ibm.db2.jcc:name=DB2TraceManager";
  ObjectName name = new ObjectName(objectNameString);
  System.out.println ("ObjectName="+objectNameString);

  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Print all attributes of the MBean");
  System.out.println ("-------------------------------------------------");
  com.ibm.db2.jcc.mx.DB2TraceManagerMXBean mbeanProxy =
    JMX.newMBeanProxy(mbsc, name,
      com.ibm.db2.jcc.mx.DB2TraceManagerMXBean.class, true);
  System.out.println ("TraceDirectory  = "+mbeanProxy.getTraceDirectory ());
  System.out.println ("TraceFile       = "+mbeanProxy.getTraceFile ());
  System.out.println ("TraceFileAppend = "+mbeanProxy.getTraceFileAppend ());
  System.out.println ("TraceLevel      = "+mbeanProxy.getTraceLevel ());
  System.out.println ("");
  System.out.println ("-------------------------------------------------");
  System.out.println ("Invoke some operations on the MBean");
  System.out.println ("-------------------------------------------------");
  System.out.print ("Invoking suspendTrace()...");
  mbeanProxy.suspendTrace();
  System.out.println ("success");
  System.out.print ("Invoking resumeTrace()...");
  mbeanProxy.resumeTrace();
  System.out.println ("success");
}
catch (Exception e) {
  System.out.println ("failure");
  e.printStackTrace ();
}
```

第 10 章 Java 2 Platform, Enterprise Edition

Java 2 Platform, Enterprise Edition (J2EE) を使用すれば、下に述べるような複数層サービスの開発に伴うコストと複雑さが低減し、企業の要件に基づいて迅速にデプロイして容易に拡張できるサービスを実現することができます。

今日のグローバルなビジネス環境においては、組織は活動範囲を広げ、コストを削減し、また応答時間を短縮するために、顧客、従業員、サプライヤー、および他のビジネス・パートナーが容易にアクセスできるサービスを提供する必要があります。これらのサービスには、以下の特性が備わっている必要があります。

- 高可用性 - グローバルなビジネス環境の要件を満たすため
- セキュリティー - ユーザーのプライバシーと企業の保全性を保護するため
- 信頼性と拡張性 - 商取引が正確かつ迅速に処理されるため

これらのサービスは、多くの場合、各層が特定の目的を満たす複数層アプリケーションによって提供されます。

J2EE では、以下のような要素として提供されている標準アーキテクチャーを定義することにより、これらの利点が実現されます。

- J2EE Application Model - 複数層のシン・クライアント・サービスを開発するための標準アプリケーション・モデル
- J2EE Platform - J2EE アプリケーションをホストするための標準のプラットフォーム
- J2EE Compatibility Test Suite - 特定の J2EE Platform 製品の J2EE Platform 標準への準拠の検査
- J2EE Reference Implementation - J2EE 機能のデモおよび J2EE Platform の作動可能な定義の提供

Java 2 Platform, Enterprise Edition のアプリケーション・コンポーネントのサポート

Java 2 Platform, Enterprise Edition (J2EE) では、J2EE アプリケーションをホストするためのランタイム環境が提供されます。

このランタイム環境では、J2EE 製品でサポートされる必要がある以下の 4 つのアプリケーション・コンポーネント・タイプが定義されます。

- アプリケーション・クライアントは、通常デスクトップ・コンピューター上で実行する GUI プログラムである Java プログラミング言語プログラムです。アプリケーション・クライアントでは、J2EE 中間層の機能のすべてにアクセスできます。
- アプレットは、通常 Web ブラウザーで実行される GUI コンポーネントですが、アプレット・プログラミング・モデルをサポートしている他のさまざまなアプリケーションや装置でも実行できます。

- サーブレット、JavaServer Pages (JSP)、フィルター、および Web イベント・リスナーは、通常 Web サーバーで実行され、Web クライアントからの HTTP 要求に応答できます。サーブレット、JSP、およびフィルターは、アプリケーションのユーザー・インターフェースである HTML ページを生成するために使用できます。また、他のアプリケーション・コンポーネントによって取り込まれる XML や他のフォーマットのデータも生成できます。サーブレット、JSP テクノロジーによって作成されたページ、Web フィルター、および Web イベント・リスナーは、この仕様では *Web コンポーネント* と総称されています。 Web アプリケーションは、Web コンポーネントと HTML ページなどのその他のデータから構成されます。
- Enterprise JavaBeans™ (EJB) コンポーネントは、トランザクションをサポートしている管理下環境で実行されます。エンタープライズ Bean には、通常 J2EE アプリケーション用のビジネス・ロジックが含まれています。

上記の各アプリケーション・コンポーネントは、デプロイ方法や管理方法に基づいて 3 つのカテゴリーに分類することができます。

- J2EE サーバー上でデプロイ、管理、および実行されるコンポーネント。
- J2EE サーバー上でデプロイおよび管理されるが、クライアント・マシンにロードされて実行されるコンポーネント。
- デプロイメント方法と管理方法がこの仕様では完全に定義されていないコンポーネント。アプリケーション・クライアントはこのカテゴリーに含めることができます。

これらのコンポーネントのランタイム・サポートは、コンテナー によって提供されます。

Java 2 Platform, Enterprise Edition コンテナー

コンテナーでは、ベースとなる Java 2 Platform, Enterprise Edition (J2EE) API のフェデレーテッド・ビューを、アプリケーション・コンポーネントに提供します。

標準的な J2EE 製品では、アプリケーション・コンポーネント・タイプごとに 1 つのコンテナー (アプリケーション・クライアント・コンテナー、アプレット・コンテナー、Web コンテナー、およびエンタープライズ Bean コンテナー) が備わっています。また、コンテナー・ツールでは、アプリケーション・コンポーネントをデプロイメント用にパッケージするためのファイル形式も認識されます。

仕様によると、これらのコンテナーは、Java 互換のランタイム環境を提供する必要があります。この仕様では、各 J2EE 製品がサポートする必要がある一連の標準サービスが定義されています。これらの標準サービスは以下のとおりです。

- HTTP Service
- HTTPS Service
- Java Transaction API
- Remote Invocation Method
- Java IDL
- JDBC API
- Java Message Service
- Java Naming and Directory Interface

- JavaMail
- JavaBeans Activation Framework
- Java API for XML Parsing
- Connector Architecture
- Java Authentication and Authorization Service

Java 2 Platform, Enterprise Edition Server

Java 2 Platform, Enterprise Edition (J2EE) コンテナーのベースとなるものは、このコンテナーがその一部を構成しているサーバーです。

通常 J2EE Product Provider により、J2EE のサーバー・サイド機能がインプリメントされます。また、J2EE クライアント機能は、通常 J2SE テクノロジーに基づいて構築されます。

IBM WebSphere Application Server は、J2EE に準拠したサーバーです。

Java 2 Platform, Enterprise Edition のデータベース要件

Java 2 Platform, Enterprise Edition では、ビジネス・データの保管用に JDBC API を介してアクセス可能なデータベースが必要です。

このデータベースには、Web コンポーネント、エンタープライズ Bean、およびアプリケーション・クライアント・コンポーネントからアクセスすることができます。また、アプレットからアクセスできる必要はありません。

Java Naming and Directory Interface (JNDI)

JNDI を使用すれば、Java プラットフォーム・ベースのアプリケーションによる複数の命名およびディレクトリー・サービスへのアクセスが可能になります。

これは、Java Enterprise アプリケーション・プログラミング・インターフェース (API) セットの一部です。 JNDI により、開発者は複数の異なる命名およびディレクトリー・サービスに対応可能なポータブル・アプリケーションを作成できます。これらの使用可能なサービスには、ファイル・システム、Lightweight Directory Access Protocol (LDAP) や Novell Directory Services などのディレクトリー・サービス、および Common Object Request Broker Architecture (CORBA)、Java Remote Method Invocation (RMI)、Enterprise JavaBeans (EJB) などの分散オブジェクト・システムがあります。

JNDI API には次の 2 つの部分があります。ひとつは、アプリケーション・コンポーネントにより命名およびディレクトリー・サービスへのアクセスに使用されるアプリケーション・レベル・インターフェースであり、もうひとつは、命名およびディレクトリー・サービスのプロバイダーにアタッチするためのサービス・プロバイダー・インターフェースです。

Java トランザクション管理

Java 2 Platform, Enterprise Edition (J2EE) では、分散トランザクション管理用のアプリケーション・プログラミングが容易になります。

J2EE には、Java Transaction API (JTA) および Java Transaction Service (JTS) という 2 つの仕様による分散トランザクションのサポートが組み込まれています。 JTA はインプリメンテーションやプロトコルに依存しない高水準 API で、アプリケーションやアプリケーション・サーバーによるトランザクションへのアクセスを可能にします。また、JTA は常に使用可能になっています。

IBM Data Server Driver for JDBC and SQLJ および Linux、UNIX、Windows 用の DB2 JDBC Type 2 ドライバーでは、JTA および JTS の各仕様がインプリメントされています。

IBM Data Server Driver for JDBC and SQLJ Type 4 接続の場合、分散トランザクションは DB2 Database for Linux, UNIX, and Windows、 DB2 for z/OS、および DB2 for i の各サーバーでサポートされています。

JTA では、トランザクション・マネージャーと分散トランザクション・システムに関連する部分 (リソース・マネージャー、アプリケーション・サーバー、およびトランザクション・アプリケーション) との間の標準の Java インターフェースが指定されます。

JTS では、JTA をサポートし、OMG Object Transaction Service (OTS) 1.1 仕様の Java マッピングをこの API の下のレベルでインプリメントするトランザクション・マネージャーのインプリメンテーションが指定されます。 JTS では、IIOP を使用してトランザクションが伝搬されます。

JTA および JTS により、アプリケーション用の J2EE サーバーでは、トランザクション管理の負担がコンポーネント開発者から取り除かれます。開発者は、設計時またはデプロイメント時にデプロイメント記述子内で宣言ステートメントを使用して、EJB テクノロジー・ベースのコンポーネントのトランザクション・プロパティーを定義できます。アプリケーション・サーバーにより、トランザクション管理の責任が引き受けられます。

DB2 と WebSphere Application Server の環境では、WebSphere Application Server がトランザクション・マネージャーの役割を担い、 DB2 がリソース・マネージャーとして機能します。 WebSphere Application Server および DB2 による整合分散トランザクションの提供が可能になるように、WebSphere Application Server により JTS と JTA の一部がインプリメントされ、また JDBC ドライバーでも JTA の一部がインプリメントされます。

WebSphere Application Server 環境では、JDBC ドライバーによりこの環境が自動的に検出されるため、 DB2 を JTA 対応に構成する必要はありません。

DB2 JDBC Type 2 Driver では、次の 2 つの DataSource クラスが用意されています。

- COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
- COM.ibm.db2.jdbc.DB2XADataSource

IBM Data Server Driver for JDBC and SQLJ では、次の 2 つの DataSource クラスが用意されています。

- com.ibm.db2.jcc.DB2ConnectionPoolDataSource
- com.ibm.db2.jcc.DB2XADataSource

WebSphere Application Server では、データベースへのプール接続が提供されます。アプリケーションが分散トランザクションに関連する場合は、WebSphere Application Server で DB2 データ・ソースを定義する際に com.ibm.db2.jdbc.DB2XADataSource クラスを使用する必要があります。

WebSphere Application Server を DB2 と共に使用するように構成する方法について詳しくは、次のサイトにある WebSphere Application Server インフォメーション・センターを参照してください。

`http://www.ibm.com/software/webservers/appserv/library.html`

JTA メソッドを使用する分散トランザクションの例

分散トランザクションには通常、様々な製造元からのデータ・ソースを含むことができる同じデータ・ソースまたは異なるデータ・ソースへの複数の接続が含まれます。

分散トランザクションについて説明する最善の方法は、それをローカル・トランザクションと比較することです。ローカル・トランザクションでは、JDBC アプリケーションはデータベースへの変更を永久的なものとし、以下のいずれかの方法で作業単位の終わりを示します。

- 1 つ以上の SQL ステートメントを実行した後で Connection.commit または Connection.rollback メソッドを呼び出す。
- 毎回の SQL ステートメントの後で変更をコミットするために、アプリケーションの開始時に Connection.setAutoCommit(true) メソッドを呼び出す。

図 54 では、ローカル・トランザクションを実行するコードを概説しています。

```
con1.setAutoCommit(false);  // Set autocommit off
// execute some SQL
...
con1.commit();              // Commit the transaction
// execute some more SQL
...
con1.rollback();            // Roll back the transaction
con1.setAutoCommit(true);   // Enable commit after every SQL statement
...
// Execute some more SQL, which is automatically committed after
// every SQL statement.
```

図 54. ローカル・トランザクションの例

対照的に、分散トランザクションに関与するアプリケーションは、分散トランザクション内で Connection.commit、Connection.rollback、または Connection.setAutoCommit(true) メソッドを呼び出すことができません。分散トランザクションでは、Connection.commit または Connection.rollback メソッドはトランザクション境界を示しません。その代わりに、アプリケーションはアプリケーション・サーバーにトランザクション境界を管理させます。

270 ページの図 55 は、分散トランザクションを使用するアプリケーションを示しています。例の中のコードが実行している間、アプリケーション・サーバーはこの同じ分散トランザクションの一部である他の EJB も実行します。すべての EJB が utx.commit() を呼び出した場合、分散トランザクション全体がアプリケーション・サーバーによってコミットされます。いずれかの EJB が失敗した場合、アプリケーション・サーバーは分散トランザクションに関連するすべての EJB によって行われた

すべての作業をロールバックします。

```
javax.transaction.UserTransaction utx;
// Use the begin method on a UserTransaction object to indicate
// the beginning of a distributed transaction.
utx.begin();
...
// Execute some SQL with one Connection object.
// Do not call Connection methods commit or rollback.
...
// Use the commit method on the UserTransaction object to
// drive all transaction branches to commit and indicate
// the end of the distributed transaction.

utx.commit();
...
```

図 55. アプリケーション・サーバーで実行する分散トランザクションの例

図 56 は、JTA メソッドを使用して分散トランザクションを実行するプログラムを示しています。このプログラムはトランザクション・マネージャーおよびトランザクション・アプリケーションとして機能します。 2 つの異なるデータ・ソースへの 2 つの接続は、単一の分散トランザクションの下で SQL 作業を行います。

図 56. JTA を使用する分散トランザクションの例

```
class XASample
{
  javax.sql.XADataSource xaDS1;
  javax.sql.XADataSource xaDS2;
  javax.sql.XAConnection xaconn1;
  javax.sql.XAConnection xaconn2;
  javax.transaction.xa.XAResource xares1;
  javax.transaction.xa.XAResource xares2;
  java.sql.Connection conn1;
  java.sql.Connection conn2;

  public static void main (String args []) throws java.sql.SQLException
  {
    XASample xat = new XASample();
    xat.runThis(args);
  }
  // As the transaction manager, this program supplies the global
  // transaction ID and the branch qualifier. The global
  // transaction ID and the branch qualifier must not be
  // equal to each other, and the combination must be unique for
  // this transaction manager.
  public void runThis(String[] args)
  {
    byte[] gtrid = new byte[] { 0x44, 0x11, 0x55, 0x66 };
    byte[] bqual = new byte[] { 0x00, 0x22, 0x00 };
    int rc1 = 0;
    int rc2 = 0;

    try
    {

      javax.naming.InitialContext context = new javax.naming.InitialContext();
      /*
        * Note that javax.sql.XADataSource is used instead of a specific
        * driver implementation such as com.ibm.db2.jcc.DB2XADataSource.
        */
      xaDS1 = (javax.sql.XADataSource)context.lookup("checkingAccounts");
```

```
      xaDS2 = (javax.sql.XADataSource)context.lookup("savingsAccounts");

      // The XADatasource contains the user ID and password.
      // Get the XAConnection object from each XADataSource
      xaconn1 = xaDS1.getXAConnection();
      xaconn2 = xaDS2.getXAConnection();

      // Get the java.sql.Connection object from each XAConnection
      conn1 = xaconn1.getConnection();
      conn2 = xaconn2.getConnection();

      // Get the XAResource object from each XAConnection
      xares1 = xaconn1.getXAResource();
      xares2 = xaconn2.getXAResource();
      // Create the Xid object for this distributed transaction.
      // This example uses the com.ibm.db2.jcc.DB2Xid implementation
      // of the Xid interface. This Xid can be used with any JDBC driver
      // that supports JTA.
      javax.transaction.xa.Xid xid1 =
                               new com.ibm.db2.jcc.DB2Xid(100, gtrid, bqual);

      // Start the distributed transaction on the two connections.
      // The two connections do NOT need to be started and ended together.
      // They might be done in different threads, along with their SQL operations.
      xares1.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
      xares2.start(xid1, javax.transaction.xa.XAResource.TMNOFLAGS);
...
      // Do the SQL operations on connection 1.
      // Do the SQL operations on connection 2.
...
      // Now end the distributed transaction on the two connections.
      xares1.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);
      xares2.end(xid1, javax.transaction.xa.XAResource.TMSUCCESS);

      // If connection 2 work had been done in another thread,
      // a thread.join() call would be needed here to wait until the
      // connection 2 work is done.

      try
      { // Now prepare both branches of the distributed transaction.
        // Both branches must prepare successfully before changes
        // can be committed.
        // If the distributed transaction fails, an XAException is thrown.
        rc1 = xares1.prepare(xid1);
        if(rc1 == javax.transaction.xa.XAResource.XA_OK)
        { // Prepare was successful. Prepare the second connection.
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // Both connections prepared successfully and neither was read-only.
            xares1.commit(xid1, false);
            xares2.commit(xid1, false);
          }
          else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
          { // The second connection is read-only, so just commit the
            // first connection.
            xares1.commit(xid1, false);
          }
        }
        else if(rc1 == javax.transaction.xa.XAException.XA_RDONLY)
        { // SQL for the first connection is read-only (such as a SELECT).
          // The prepare committed it. Prepare the second connection.
          rc2 = xares2.prepare(xid1);
          if(rc2 == javax.transaction.xa.XAResource.XA_OK)
          { // The first connection is read-only but the second is not.
            // Commit the second connection.
            xares2.commit(xid1, false);
          }
```

```
      else if(rc2 == javax.transaction.xa.XAException.XA_RDONLY)
      { // Both connections are read-only, and both already committed,
        // so there is nothing more to do.
      }
    }
  }      catch (javax.transaction.xa.XAException xae)
  { // Distributed transaction failed, so roll it back.
    // Report XAException on prepare/commit.
    System.out.println("Distributed transaction prepare/commit failed. " +
                       "Rolling it back.");
    System.out.println("XAException error code = " + xae.errorCode);
    System.out.println("XAException message = " + xae.getMessage());
    xae.printStackTrace();
    try
    {
      xares1.rollback(xid1);
    }
    catch (javax.transaction.xa.XAException xae1)
    { // Report failure of rollback.
      System.out.println("distributed Transaction rollback xares1 failed");
      System.out.println("XAException error code = " + xae1.errorCode);
      System.out.println("XAException message = " + xae1.getMessage());
    }
    try
    {
      xares2.rollback(xid1);
    }
    catch (javax.transaction.xa.XAException xae2)
    { // Report failure of rollback.
      System.out.println("distributed Transaction rollback xares2 failed");
      System.out.println("XAException error code = " + xae2.errorCode);
      System.out.println("XAException message = " + xae2.getMessage());
    }
  }

  try
  {
    conn1.close();
    xaconn1.close();
  }
  catch (Exception e)
  {
    System.out.println("Failed to close connection 1: " + e.toString());
    e.printStackTrace();
  }
  try
  {
    conn2.close();
    xaconn2.close();
  }
  catch (Exception e)
  {
    System.out.println("Failed to close connection 2: " + e.toString());
    e.printStackTrace();
  }
}
catch (java.sql.SQLException sqe)
{
  System.out.println("SQLException caught: " + sqe.getMessage());
  sqe.printStackTrace();
}
catch (javax.transaction.xa.XAException xae)
{
  System.out.println("XA error is " + xae.getMessage());
  xae.printStackTrace();
}
catch (javax.naming.NamingException nme)
```

```
    {
      System.out.println(" Naming Exception: " + nme.getMessage());
    }
  }
}
```

推奨事項: パフォーマンスを向上させるには、別の分散トランザクションまたはローカル・トランザクションを開始する前に、分散トランザクションを完了してください。

XAResource インスタンスのトランザクション・タイムアウト値の設定

XAResource.setTransactionTimeout メソッドを使用して、分散トランザクションのターゲットになっている DB2 データベースでデッドロックの回数を減らします。

準備ができなくても終了する DB2 Database for Linux, UNIX, and Windows の分散トランザクションは、未確定トランザクションではありません。したがって、トランザクション・マネージャーは、そのトランザクションを回復できませんし、DB2 リソース・マネージャーも、そのトランザクションを未確定トランザクションのリストに入れません。DB2 リソース・マネージャーは、そのトランザクションをすぐにロールバックしないで、すべてのデータベース接続が解放されるまで待機します。その間はアクティビティーがないまま、そのトランザクションがデータベースに対するロックを保持することになります。トランザクション・マネージャーがすべてのデータベース接続を切断してロールバックを可能にしなければ、終了したトランザクションがいつまでもデータベース・レコードをロックし続けてしまいます。ロックされているレコードに別のアプリケーションがアクセスしようとすると、デッドロックが発生する可能性があります。

分散トランザクションと IBM Data Server Driver for JDBC and SQLJ Type 4 接続を使用する Java アプリケーションでは、XAResource.setTransactionTimeout メソッドを呼び出してトランザクションのタイムアウト値を設定することによって、トランザクションがデータベースのロックをいつまでも保持する事態を回避できます。そのためには、以下の手順を実行します。

1. DB2 Database for Linux, UNIX, and Windows インスタンスで以下のコマンドを実行して、インスタンスがタイムアウト値をチェックするようにします。

   ```
   DB2 UPDATE DBM CFG USING RESYNC_INTERVAL seconds
   ```

   seconds は、トランザクションに設定する最小タイムアウト値より小さい値にする必要があります。
2. アプリケーションで XAResource オブジェクトを作成してから、XAResource.setTransactionTimeout メソッドを呼び出してタイムアウト値を設定します。

   現在のタイムアウト値をチェックするには、XAResource.getTransactionTimeout を呼び出します。

Enterprise Java Beans

Enterprise Java Beans アーキテクチャーは、コンポーネント・ベースの分散ビジネス・アプリケーションを開発およびデプロイするためのコンポーネント・アーキテクチャーです。

Enterprise Java Beans アーキテクチャーを使用して作成されたアプリケーションは、一度作成すれば、Enterprise Java Beans 仕様をサポートするすべてのサーバー・プラットフォームにデプロイすることができます。Java 2 Platform, Enterprise Edition (J2EE) アプリケーションによって、セッション Bean とエンティティー Bean が含まれた Enterprise Java Beans (EJB) が使用され、サーバー・サイドのビジネス・コンポーネントがインプリメントされます。

セッション Bean はビジネス・サービスを表し、ユーザー間では共有されません。エンティティー Bean とは、永続データを表すマルチユーザーの分散トランザクション・オブジェクトです。 EJB アプリケーションのトランザクション境界は、コンテナー管理トランザクションまたは Bean 管理トランザクションを指定することによって設定できます。

サンプル・プログラムの `AccessEmployee.ear` により、Enterprise Java Beans が使用され、データ・ソースにアクセスする J2EE アプリケーションがインプリメントされます。このサンプルは `SQLLIB/samples/websphere` ディレクトリーにあります。

この EJB サンプル・アプリケーションにより、次の 2 つのビジネス・サービスが提供されます。1 つのサービスにより、ユーザーは従業員の従業員番号でその従業員に関する情報 (**sample** データベースの EMPLOYEE 表に保管されている) にアクセスできます。もう 1 つのサービスにより、ユーザーは従業員番号のリストを検索して、従業員データの照会に使用する従業員番号を取得できます。

下のサンプルでは EJB が使用されて、データ・ソースにアクセスする J2EE アプリケーションがインプリメントされます。このサンプルでは、一般的に使用される GUI アーキテクチャーの Model-View-Controller (MVC) アーキテクチャーが使用されます。 JSP は、View (表示コンポーネント) のインプリメントに使用されます。サーブレットは、このサンプルで Controller (コントローラー) として機能します。サーブレットによりワークフローが制御され、ユーザーの要求は EJB によってインプリメントされた Model (モデル) に委任されます。このサンプルの Model コンポーネントは、2 つの EJB (1 つのセッション Bean と 1 つのエンティティー Bean) によって構成されます。コンテナー管理パーシスタンス (CMP) Bean である Employee により、サンプル・データベースの EMPLOYEE 表にある永続データを表す分散トランザクション・オブジェクトが表されます。「コンテナー管理パーシスタンス」という用語は、エンティティー Bean で必要なすべてのデータベース・アクセスが EJB コンテナーによって処理されることを意味します。この Bean のコードには、データベース・アクセス (SQL) の呼び出しが含まれていません。このため、この Bean のコードは特定の永続ストレージ機構 (データベース) に束縛されません。セッション Bean の AccessEmployee はエンティティー Bean の Façade (ファサード) として機能し、統一的なクライアント・アクセス戦略が提供されます。この Façade 設計により、EJB クライアントとエンティティー Bean 間のネットワーク・トラフィックが低減し、エンティティー Bean が EJB クライアントから直接アクセスされる場合よりも分散トランザクションが効率的になります。データベー

ス・サーバーへのアクセスは、セッション Bean またはエンティティー Bean から提供できます。このサンプル・アプリケーションの 2 つのサービスでは、データベース・サーバーへのアクセス方法が両方とも示されています。次の最初のサービスでは、エンティティー Bean が使用されています。

```
//=====================================================
// This method returns an employee's information by
// interacting with the entity bean located by the
// provided employee number
public EmployeeInfo getEmployeeInfo(String empNo)
throws java.rmi.RemoteException
}
Employee employee = null;
try
}
employee = employeeHome.findByPrimaryKey(new EmployeeKey(empNo));
EmployeeInfo empInfo = new EmployeeInfo(empNo);
//set the employee's information to the dependent value object
empInfo.setEmpno(employee.getEmpno());
empInfo.setFirstName (employee.getFirstName());
empInfo.setMidInit(employee.getMidInit());
empInfo.setLastName(employee.getLastName());
empInfo.setWorkDept(employee.getWorkDept());
empInfo.setPhoneNo(employee.getPhoneNo());
empInfo.setHireDate(employee.getHireDate());
empInfo.setJob(employee.getJob());
empInfo.setEdLevel(employee.getEdLevel());
empInfo.setSex(employee.getSex());
empInfo.setBirthDate(employee.getBirthDate());
empInfo.setSalary(employee.getSalary());
empInfo.setBonus(employee.getBonus());
empInfo.setComm(employee.getComm());
return empInfo;
}
catch (java.rmi.RemoteException rex)
{
......
```

従業員番号を表示する 2 番目のサービスでは、セッション Bean の AccessEmployee によりデータベース表が直接アクセスされています。

```
/=============================================
* Get the employee number list.
* @return Collection
*/
public Collection getEmpNoList()
{
ResultSet rs = null;
PreparedStatement ps = null;
Vector list = new Vector();
DataSource ds = null;
Connection con = null;
try
{
ds = getDataSource();
con = ds.getConnection();
String schema = getEnvProps(DBschema);
String query = "Select EMPNO from " + schema + ".EMPLOYEE";
ps = con.prepareStatement(query);
ps.executeQuery();
rs = ps.getResultSet();
EmployeeKey pk;
while (rs.next())
{
pk = new EmployeeKey();
pk.employeeId = rs.getString(1);
```

```
list.addElement(pk.employeeId);
}
rs.close();
return list;
```

第 11 章 JDBC および SQLJ 接続プールのサポート

接続プール は、JDBC の DataSource サポートの一部であり、 IBM Data Server Driver for JDBC and SQLJ によってサポートされます。

IBM Data Server Driver for JDBC and SQLJ では、WebSphere Application Server または他のアプリケーション・サーバーによって使用されるプール接続のファクトリーが提供されます。このプールは、実際にはアプリケーション・サーバーにより実行されます。接続プールは、JDBC または SQLJ アプリケーションには全く認識されません。

接続プールとは、 DB2 スレッドに相当する物理データ・ソース接続をキャッシュするためのフレームワークです。 JDBC により物理データ・ソース接続が再利用されると、java.sql.Connection オブジェクトの作成とそれに続く終了に必要な負荷のかかる操作が最小化されます。

接続プールがない場合、各 java.sql.Connection オブジェクトがデータ・ソースへの物理接続を表します。アプリケーションによりデータ・ソースへの接続が確立されると、 DB2 によりデータ・ソースへの新規の物理接続が作成されます。アプリケーションにより java.sql.Connection.close メソッドが呼び出されると、 DB2 により、このデータ・ソースへの物理接続が終了されます。

これとは対照的に、接続プールでは、java.sql.Connection オブジェクトが物理データ・ソース接続を一時的および論理的に表すものです。物理データ・ソース接続は、論理 java.sql.Connection インスタンスによって連続的に再利用されます。アプリケーションでは、接続プール・サポートがない場合に java.sql.Connection オブジェクトを使用するときと全く同様に、論理 java.sql.Connection オブジェクトを使用できます。

接続プールでは、JDBC アプリケーションにより DataSource.getConnection メソッドが呼び出されると、データ・ソースで適切な物理接続が存在するかどうかが判定されます。適切な物理接続が既存する場合、データ・ソースにより java.sql.Connection インスタンスがアプリケーションに返されます。 JDBC アプリケーションにより java.sql.Connection.close メソッドが呼び出されても、JDBC では物理データ・ソース接続はクローズされません。その代わりに、JDBC では Statement オブジェクトや ResultSet オブジェクトなどの JDBC リソースのみがクローズされます。データ・ソースにより、物理接続が再利用のために接続プールに返されます。

接続プールは、*同種* または*異種* にすることができます。

同種プールの場合、1 つの接続プールからのすべての Connection オブジェクトに同じプロパティーがあります。 DataSource で作成された最初の論理 Connection には、DataSource に定義されたプロパティーがあります。ただし、アプリケーションでそれらのプロパティーを変更できます。 Connection が接続プールに返されると、アプリケーション・サーバーまたはプール・モジュールにより、これらのプロパティーは元の値にリセットされます。ただし、アプリケーション・サーバーまたはプール・モジュールにより、変更されたプロパティーがリセットされない場合があり

ます。 JDBC ドライバーでは、これらのプロパティーは変更されません。したがって、アプリケーション・サーバーまたはプール・モジュールの設計に応じて、再利用される論理 Connection が、DataSource に定義されたものと同じプロパティーを持つこともあれば、異なるプロパティーを持つこともあります。

異種プールでは、異なるプロパティーの Connection オブジェクトが、同じ接続プールを共有することができます。

第 12 章 JDBC および SQLJ の参照情報

JDBC および SQLJ の IBM のインプリメンテーションでは、JDBC および SQLJ アプリケーションを開発するための数多くのアプリケーション・プログラミング・インターフェース、プロパティー、およびコマンドが提供されています。

Java アプリケーションでデータベース・データ・タイプにマップするデータ・タイプ

効果的な JDBC および SQLJ プログラムを作成するには、Java データ・タイプと表列データ・タイプの間で最適なマッピングを使用する必要があります。

以下の表は、 DB2 Database for Linux, UNIX, and Windows、 DB2 for z/OS、または IBM Informix Dynamic Server (IDS) システムの場合の、Java データ・タイプの JDBC およびデータベース・データ・タイプへのマッピングを要約したものです。

表列を更新するためのデータ・タイプ

以下の表は、JDBC プログラム内の PreparedStatement.setXXX または ResultSet.updateXXX メソッド、および SQLJ プログラム内の入力ホスト式の場合の、Java データ・タイプのデータベース・データ・タイプへのマッピングを要約したものです。複数の Java データ・タイプがリストされている場合、最初のデータ・タイプが推奨されるデータ・タイプです。

表 39. データベース表を更新するための、Java データ・タイプのデータベース・サーバー・データ・タイプへのマッピング

Java データ・タイプ	データベース・データ・タイプ
short	SMALLINT
boolean[1] 、byte[1]、java.lang.Boolean	SMALLINT
int、java.lang.Integer	INTEGER
long、java.lang.Long	BIGINT[11]
float、java.lang.Float	REAL
double、java.lang.Double	DOUBLE
java.math.BigDecimal	DECIMAL(*p*,*s*)[2]
java.math.BigDecimal	DECFLOAT(*n*)[3、 4]
java.lang.String	CHAR(*n*)[5]
java.lang.String	GRAPHIC(*m*)[6]
java.lang.String	VARCHAR(*n*)[7]
java.lang.String	VARGRAPHIC(*m*)[8]
java.lang.String	CLOB[9]
java.lang.String	XML[10]
byte[]	CHAR(*n*) FOR BIT DATA[5]
byte[]	VARCHAR(*n*) FOR BIT DATA[7]
byte[]	BINARY(*n*)[5、 12]

表 39. データベース表を更新するための、Java データ・タイプのデータベース・サーバー・データ・タイプへのマッピング (続き)

Java データ・タイプ	データベース・データ・タイプ
byte[]	VARBINARY(*n*)[7, 12]
byte[]	BLOB[9]
byte[]	ROWID
byte[]	XML[10]
java.sql.Blob	BLOB
java.sql.Blob	XML[10]
java.sql.Clob	CLOB
java.sql.Clob	DBCLOB[9]
java.sql.Clob	XML[10]
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP または TIMESTAMP(*p*)[13]
java.io.ByteArrayInputStream	BLOB
java.io.StringReader	CLOB
java.io.ByteArrayInputStream	CLOB
java.io.InputStream	XML[10]
com.ibm.db2.jcc.DB2RowID (非推奨)	ROWID
java.sql.RowId	ROWID
com.ibm.db2.jcc.DB2Xml (非推奨)	XML[10]
java.sql.SQLXML	XML[10]

表 39. データベース表を更新するための、Java データ・タイプのデータベース・サーバー・データ・タイプへのマッピング (続き)

Java データ・タイプ	データベース・データ・タイプ

注:

1. データベース・サーバーには、Java Boolean 値またはバイト・データ・タイプに厳密に相当するものはありませんが、最も近いものが SMALLINT です。
2. *p* は 10 進数精度で、*s* は表列のスケールです。

   java.math.BigDecimal 列が DECIMAL 列にマップするように金融アプリケーションを設計するとします。DECIMAL 列の精度およびスケールがわかっている場合、DECIMAL 列のデータを java.math.BigDecimal 変数の持つデータで更新すると、他のデータ・タイプの組み合わせを使用する場合よりもパフォーマンスが向上します。
3. *n*=16 または *n*=34。
4. DECFLOAT は、DB2 バージョン 9.1 for z/OS、 DB2 V9.5 for Linux, UNIX, and Windows、または DB2 for i V6R1、またはそれ以降のデータベース・サーバーへの接続に有効です。DECFLOAT を使用するには、SDK for Java バージョン 5 (1.5) 以降が必要です。
5. *n*<=254。
6. *m*<=127。
7. *n*<=32672。
8. *m*<=16336。
9. このマッピングは、データベース・サーバーが列のデータ・タイプを判別できる場合にのみ有効です。
10. XML は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、または DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバーへの接続に有効です。
11. BIGINT は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、 DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバー、およびサポートされているすべての DB2 for i データベース・サーバーへの接続に有効です。
12. BINARY および VARBINARY は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、 DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバー、および DB2 for i5/OS® V5R3 以降のデータベース・サーバーへの接続に有効です。
13. *p* は、タイム・スタンプの精度を示しています。これは、タイム・スタンプの小数部分の桁数です。 0<=*p*<=12。デフォルトは 6 です。TIMESTAMP(*p*) がサポートされているのは、 DB2 Database for Linux, UNIX, and Windows V9.7 以降への接続のみです。

表列を検索するためのデータ・タイプ

以下の表は、JDBC プログラム内の ResultSet.getXXX メソッド、および SQLJ プログラム内のイテレーターの場合の、DB2 または IDS データ・タイプの Java データ・タイプへのマッピングを要約したものです。この表では、Java 数値ラッパー・オブジェクト・タイプはリストしていません。これは、ResultSet.getObject を使用して検索されます。

表 40. データベース・サーバー表からデータを検索するための、データベース・サーバー・データ・タイプの Java データ・タイプへのマッピング

SQL データ・タイプ	推奨される Java データ・タイプまたは Java オブジェクト・タイプ	その他のサポートされる Java データ・タイプ
SMALLINT	short	byte、 int、 long、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String
INTEGER	int	short、 byte、 long、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String
BIGINT[5]	long	int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String
DECIMAL(*p*,*s*) または NUMERIC(*p*,*s*)	java.math.BigDecimal	long、 int、 short、 byte、 float、 double、 boolean、 java.lang.String
DECFLOAT(*n*)[1、 2]	java.math.BigDecimal	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.lang.String
REAL	float	long、 int、 short、 byte、 double、 java.math.BigDecimal、 boolean、 java.lang.String
DOUBLE	double	long、 int、 short、 byte、 float、 java.math.BigDecimal、 boolean、 java.lang.String
CHAR(*n*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
VARCHAR(*n*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
CHAR(*n*) FOR BIT DATA	byte[]	java.lang.String、java.io.InputStream、 java.io.Reader
VARCHAR(*n*) FOR BIT DATA	byte[]	java.lang.String、 java.io.InputStream、 java.io.Reader
BINARY(*n*)[6]	byte[]	なし
VARBINARY(*n*)[6]	byte[]	なし
GRAPHIC(*m*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader

表 40. データベース・サーバー表からデータを検索するための、データベース・サーバー・データ・タイプの Java データ・タイプへのマッピング (続き)

SQL データ・タイプ	推奨される Java データ・タイプまたは Java オブジェクト・タイプ	その他のサポートされる Java データ・タイプ
VARGRAPHIC(*m*)	java.lang.String	long、 int、 short、 byte、 float、 double、 java.math.BigDecimal、 boolean、 java.sql.Date、 java.sql.Time、 java.sql.Timestamp、 java.io.InputStream、 java.io.Reader
CLOB(*n*)	java.sql.Clob	java.lang.String
BLOB(*n*)	java.sql.Blob	byte[][3]
DBCLOB(*m*)	厳密な対応なし。java.sql.Clob を使用。	
ROWID	java.sql.RowId	byte[]、com.ibm.db2.jcc.DB2RowID (非推奨)
XML[4]	java.sql.SQLXML	byte[]、java.lang.String、 java.io.InputStream、java.io.Reader
DATE	java.sql.Date	java.sql.String、java.sql.Timestamp
TIME	java.sql.Time	java.sql.String、java.sql.Timestamp
TIMESTAMP または TIMESTAMP(*p*)[7]	java.sql.Timestamp	java.sql.String、java.sql.Date、 java.sql.Time、java.sql.Timestamp

注:

1. *n*=16 または *n*=34。
2. DECFLOAT は、DB2 バージョン 9.1 for z/OS、 DB2 V9.5 for Linux, UNIX, and Windows、または DB2 for i V6R1、またはそれ以降のデータベース・サーバーへの接続に有効です。DECFLOAT を使用するには、SDK for Java バージョン 5 (1.5) 以降が必要です。
3. このマッピングは、データベース・サーバーが列のデータ・タイプを判別できる場合にのみ有効です。
4. XML は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、または DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバーへの接続に有効です。
5. BIGINT は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、 DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバー、およびサポートされているすべての DB2 for i データベース・サーバーへの接続に有効です。
6. BINARY および VARBINARY は、DB2 for z/OS バージョン 9.1 以降のデータベース・サーバー、 DB2 for Linux, UNIX, and Windows V9.1 以降のデータベース・サーバー、および DB2 for i5/OS V5R3 以降のデータベース・サーバーへの接続に有効です。
7. *p* は、タイム・スタンプの精度を示しています。これは、タイム・スタンプの小数部分の桁数です。 0<=*p*<=12。デフォルトは 6 です。TIMESTAMP(*p*) がサポートされているのは、 DB2 Database for Linux, UNIX, and Windows V9.7 以降への接続のみです。

ストアード・プロシージャーおよびユーザー定義関数を呼び出すためのデータ・タイプ

以下の表は、ユーザー定義関数およびストアード・プロシージャー・パラメーターを呼び出すための、Java データ・タイプの JDBC データ・タイプおよび DB2 または IDS データ・タイプへのマッピングを要約したものです。 Java データ・タイプの JDBC データ・タイプへのマッピングは、JDBC プログラム内の

CallableStatement.registerOutParameter メソッドを対象としています。 Java データ・タイプのデータベース・サーバー・データ・タイプへのマッピングは、ストアード・プロシージャーまたはユーザー定義関数呼び出しでのパラメーターを対象としています。

複数の Java データ・タイプが以下の表でリストされている場合、最初のデータ・タイプが**推奨される**データ・タイプです。

表 41. ストアード・プロシージャーおよびユーザー定義関数を呼び出すための Java、JDBC、および SQL データ・タイプのマッピング

Java データ・タイプ	JDBC データ・タイプ	SQL データ・タイプ
boolean[1]	BIT	SMALLINT
byte[1]	TINYINT	SMALLINT
short、java.lang.Short	SMALLINT	SMALLINT
int、java.lang.Integer	INTEGER	INTEGER
long	BIGINT	BIGINT[5]
float、java.lang.Float	REAL	REAL
float、java.lang.Float	FLOAT	REAL
double、java.lang.Double	DOUBLE	DOUBLE
java.math.BigDecimal	NUMERIC	DECIMAL
java.math.BigDecimal	DECIMAL	DECIMAL
java.math.BigDecimal	java.types.OTHER	DECFLOAT*n*[2]
java.math.BigDecimal	com.ibm.db2.jcc.DB2Types.DECFLOAT	DECFLOAT*n*[2]
java.lang.String	CHAR	CHAR
java.lang.String	CHAR	GRAPHIC
java.lang.String	VARCHAR	VARCHAR
java.lang.String	VARCHAR	VARGRAPHIC
java.lang.String	LONGVARCHAR	VARCHAR
java.lang.String	VARCHAR	CLOB
java.lang.String	LONGVARCHAR	CLOB
java.lang.String	CLOB	CLOB
byte[]	BINARY	CHAR FOR BIT DATA
byte[]	VARBINARY	VARCHAR FOR BIT DATA
byte[]	BINARY	BINARY[4]
byte[]	VARBINARY	VARBINARY[4]
byte[]	LONGVARBINARY	VARCHAR FOR BIT DATA
byte[]	VARBINARY	BLOB[3]
byte[]	LONGVARBINARY	BLOB[3]
java.sql.Date	DATE	DATE
java.sql.Time	TIME	TIME
java.sql.Timestamp	TIMESTAMP	TIMESTAMP
java.sql.Blob	BLOB	BLOB

表 41. ストアード・プロシージャーおよびユーザー定義関数を呼び出すための Java、JDBC、および SQL データ・タイプのマッピング (続き)

Java データ・タイプ	JDBC データ・タイプ	SQL データ・タイプ
java.sql.Clob	CLOB	CLOB
java.sql.Clob	CLOB	DBCLOB
java.io.ByteArrayInputStream	なし	BLOB
java.io.StringReader	なし	CLOB
java.io.ByteArrayInputStream	なし	CLOB
com.ibm.db2.jcc.DB2RowID (非推奨)	com.ibm.db2.jcc.DB2Types.ROWID	ROWID
java.sql.RowId	java.sql.Types.ROWID	ROWID
com.ibm.db2.jcc.DB2Xml (非推奨)	com.ibm.db2.jcc.DB2Types.XML	XML AS CLOB
java.sql.SQLXML	java.sql.Types.SQLXML	XML
java.sql.SQLXML	java.sql.Types.SQLXML	XML AS CLOB
java.sql.Array[6]	java.sql.Types.ARRAY	ARRAY
java.sql.ResultSet	com.ibm.db2.jcc.DB2Types.CURSOR	CURSOR タイプ

注:

1. SMALLINT パラメーターで定義されたストアード・プロシージャーまたはユーザー定義関数は、Boolean パラメーターまたはバイト・パラメーターで呼び出すことができます。ただし、これは推奨されていません。
2. Java ルーチンの DECFLOAT パラメーターは、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバーへの接続にのみ有効です。 Java ルーチンの DECFLOAT パラメーターは、for Linux, UNIX, and Windows または DB2 for i への接続にはサポートされていません。DECFLOAT を使用するには、SDK for Java バージョン 5 (1.5) 以降が必要です。
3. このマッピングは、データベース・サーバーが列のデータ・タイプを判別できる場合にのみ有効です。
4. BINARY および VARBINARY は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、 DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバー、および DB2 for i5/OS V5R3 以降のデータベース・サーバーへの接続に有効です。
5. BIGINT は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、 DB2 V9.1 for Linux, UNIX, and Windows 以降のデータベース・サーバー、およびサポートされているすべての DB2 for i データベース・サーバーへの接続に有効です。
6. ARRAY パラメーターはストアード・プロシージャーにのみサポートされます。

Java ストアード・プロシージャーおよびユーザー定義関数でのデータ・タイプ

以下の表は、CREATE PROCEDURE または CREATE FUNCTION ステートメント内の SQL パラメーターのデータ・タイプを、対応する Java ストアード・プロシージャーまたはユーザー定義関数メソッド内のデータ・タイプにマッピングすることについて要約したものです。

DB2 Database for Linux, UNIX, and Windows の場合、複数の Java データ・タイプが SQL データ・タイプについてリストされている場合は、**最初の** Java データ・タイプのみ有効です。

DB2 for z/OS の場合、複数の Java データ・タイプがリストされており、最初のデータ・タイプ以外のデータ・タイプをメソッド・パラメーターとして使用する場合は、そのメソッド・パラメーターの Java データ・タイプを指定する CREATE

PROCEDURE または CREATE FUNCTION ステートメントの EXTERNAL 節にメソッド・シグニチャーを含める必要があります。

表 42. CREATE PROCEDURE または CREATE FUNCTION ステートメントの SQL データ・タイプを、対応する Java ストアード・プロシージャーまたはユーザー定義関数プログラム内のデータ・タイプにマッピングする

CREATE PROCEDURE または CREATE FUNCTION での SQL データ・タイプ	**Java ストアード・プロシージャーまたはユーザー定義関数メソッドでのデータ・タイプ[1]**
SMALLINT	short、java.lang.Integer
INTEGER	int、java.lang.Integer
BIGINT[3]	long、java.lang.Long
REAL	float、java.lang.Float
DOUBLE	double、java.lang.Double
DECIMAL	java.math.BigDecimal
DECFLOAT[2]	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
CHAR FOR BIT DATA	byte[]
VARCHAR FOR BIT DATA	byte[]
BINARY[3]	byte[]
VARBINARY[3]	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BLOB	java.sql.Blob
CLOB	java.sql.Clob
DBCLOB	java.sql.Clob
ROWID	java.sql.Types.ROWID
XML AS CLOB	java.sql.Types.SQLXML

注:

1. DB2 Database for Linux, UNIX, and Windows サーバー上のストアード・プロシージャーまたはユーザー定義関数の場合、**最初の**データ・タイプのみ有効です。
2. Java ルーチンの DECFLOAT パラメーターは、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバーへの接続にのみ有効です。 Java ルーチンの DECFLOAT パラメーターは、for Linux, UNIX, and Windows または DB2 for i への接続にはサポートされていません。DECFLOAT を使用するには、SDK for Java バージョン 5 (1.5) 以降が必要です。
3. BIGINT、BINARY、および VARBINARY は、DB2 バージョン 9.1 for z/OS 以降のデータベース・サーバー、または DB2V9.1for Linux, UNIX, and Windows 以降のデータベース・サーバーへの接続に有効です。

JDBC および SQLJ アプリケーションで問題が発生する可能性のある日付、時刻、およびタイム・スタンプの値

日付、時刻、およびタイム・スタンプの値として、現実の日付と時刻に対応していないものを使用すると、JDBC および SQLJ アプリケーションが予期しない結果になる場合があります。

以下の項目は、問題を起こす可能性があります。

- 深夜を指定するのに「24 時」を使用
- 1582 年 10 月 5 日から 1582 年 10 月 14 日までの間 (両日を含む) の日付を使用

「24」時を夜の 12 時として使用した場合の問題

IBM Data Server Driver for JDBC and SQLJ では、JDBC および SQLJ アプリケーションにおける入出力パラメーターや ResultSet の内容の内部処理において Java のデータ・タイプが使用されています。ドライバーで使用される Java データ・タイプは、ターゲット SQL タイプがドライバーに認識されている場合に、対応する SQL タイプとの一致度が最高のものになります。

SQL タイプの DATE、TIME、または TIMESTAMP に代入される値やそれらから取り出した値について、 IBM Data Server Driver for JDBC and SQLJ は、SQL の DATE タイプについては java.sql.Date を、SQL の TIME タイプについては java.sql.Time を、そして SQL の TIMESTAMP タイプについては java.sql.Timestamp を使用します。

DATE、TIME、または TIMESTAMP ターゲットにストリング値を代入すると、IBM Data Server Driver for JDBC and SQLJ は、Java の機能を使用することによって、そのストリング値を java.sql.Date、java.sql.Time、または java.sql.Timestamp の値に変換します。日付、時刻、またはタイム・スタンプ値のストリング表記が現実の日付や時刻に対応していない場合、Java によりその値は現実の日付または時刻の値に合わせて調整されます。特に、Java では「24」時の値を次の日の「00」時に調整します。この調整のため、'9999-12-31 24:00:00.0' というタイム・スタンプ値で例外が発生することがあります。調整後の年の値が '10000' になるためです。

重要: JDBC または SQLJ アプリケーションで日付、時刻、またはタイム・スタンプの値を代入したり取り出したりする際に、予期しない結果になるのを回避するため、値が現実の日付、時刻、またはタイム・スタンプ値として妥当なものになるようにしてください。さらに、時刻またはタイム・スタンプの値の「時」コンポーネントとして 24 を使用しないようにしてください。

「時」コンポーネントが 24 の値など、現実の日付や時刻に対応しない値が TIME または TIMESTAMP 列に格納されている場合、ResultSet を定義する SELECT ステートメントの中でその列に対して SQL の CHAR 関数を実行することにより、値取り出しの際に調整処理が実行されるのを回避できます。 CHAR 関数を実行すると、日付または時刻の値が、データベース側の文字ストリングの値に変換されます。しかし、ResultSet からその値を取り出すのに getTime または getTimestamp メソッドを使用する場合は、 IBM Data Server Driver for JDBC and SQLJ によってその値が java.sql.Time または java.sql.Timestamp タイプに変換され、Java によって値の調整がなされます。日付の調整処理を回避するには、列値に対して CHAR 関数を実行し、なおかつ、ResultSet から値を取り出すのに getString メソッドを使用するようにしてください。

以下の例は、JDBC または SQLJ アプリケーション側のデータが現実の日付や時刻を表すものではない場合に、アプリケーションで DATE、TIME、TIMESTAMP 列を更新した結果を示すものです。

表 43. 現実の日付または時刻を表すものではない Java の日付、時刻、またはタイム・スタンプ値により、SQL の DATE、TIME、または TIMESTAMP 値を更新する例

ストリング入力値	データベースのターゲット・タイプ	表の列に入れられる値、または例外
2008-13-35	DATE	2009-02-04
25:00:00	TIME	01:00:00
24:00:00	TIME	00:00:00
2008-15-36 28:63:74.0	TIMESTAMP	2009-04-06 05:04:14.0
9999-12-31 24:00:00.0	TIMESTAMP	例外。調整後の値 (10000-01-01 00:00:00.0) が年の最大値 9999 を超えるため。

以下の例は、TIMESTAMP 列の値が現実の日付または時刻を表すものではない場合に、JDBC または SQLJ アプリケーションでその列からデータを取り出した結果を示します。

表 44. 現実の日付または時刻を表すものではない SQL の DATE、TIME、または TIMESTAMP 値を取り出して、Java アプリケーションの変数に入れた結果

SELECT ステートメント	TIMESTAMP 列 TS_COL の値	アプリケーションのターゲット・タイプ (getXXX メソッドにより取り出す)	表列から取り出される値
SELECT TS_COL FROM TABLE1	2000-01-01 24:00:00.000000	java.sql.Timestamp (getTimestamp)	2000-01-02 00:00:00.000000
SELECT TS_COL FROM TABLE1	2000-01-01 24:00:00.000000	String (getString)	2000-01-02 00:00:00.000000
SELECT CHAR(TS_COL) FROM TABLE1	2000-01-01 24:00:00.000000	java.sql.Timestamp (getTimestamp)	2000-01-02 00:00:00.000000
SELECT CHAR(TS_COL) FROM TABLE1	2000-01-01 24:00:00.000000	String (getString)	2000-01-01 24:00:00.000000 (Java による調整なし)

1582 年 10 月 5 日から 1582 年 10 月 14 日まで範囲の日付を使用した場合の問題

Java クラスの java.util.Date と java.util.Timestamp は、1582 年 10 月 4 日より前の日付にはユリウス暦を使用し、1582 年 10 月 4 日以降の日付にはグレゴリオ暦を使用します。グレゴリオ暦では、1582 年 10 月 4 日の次は 1582 年 10 月 15 日となります。 Java プログラムが 1582 年 10 月 5 日から 1582 年 10 月 14 日までの間 (両日を含む) の java.util.Date 値または java.util.Timestamp 値を検出すると、Java はその日付に 10 日を加えます。そのため、DB2 表の DATE または TIMESTAMP の値が 1582 年 10 月 5 日から 1582 年 10 月 14 日までの間 (両日を含む) の値である場合、Java プログラムでは、その値は 1582 年 10 月 15 日から 1582 年 10 月 24 日までの間 (両日を含む) の java.util.Date 値または java.util.Timestamp 値として取得されます。 Java プログラムで java.util.Date 値または java.util.Timestamp 値が 1582 年 10 月 5 日から 1582 年 10 月 14 日までの間

(両日を含む) の値である場合、その値は 1582 年 10 月 15 日から 1582 年 10 月 24 日までの間 (両日を含む) の DATE 値または TIMESTAMP 値として DB2 表に格納されます。

例: DATE 列から 1582 年 10 月 10 日を取り出します。

```
// DATETABLE has one date column with one row.
// Its value is 1582-10-10.
java.sql.ResultSet rs =
 statement.executeQuery(select * from DATETABLE);
rs.next();
System.out.println(rs.getDate(1)); // Value is retrieved as 1582-10-20
```

例: 1582 年 10 月 10 日を DATE 列に格納します。

```
java.sql.Date d = java.sql.Date.valueOf("1582-10-10");
java.sql.PreparedStatement ps =
 c.prepareStatement("Insert into DATETABLE values(?)");
ps.setDate(1, d);
ps.executeUpdate(); // Value is inserted as 1582-10-20
```

1582 年 10 月 5 日から 1582 年 10 月 14 日までの範囲の値を日付調整なしで DB2 表から取り出すには、ResultSet を定義した SELECT ステートメントで、DATE 列または TIMESTAMP 列に対して SQL CHAR 関数を実行します。 CHAR 関数を実行すると、日付または時刻の値が、データベース側の文字ストリングの値に変換されます。

1582 年 10 月 5 日から 1582 年 10 月 14 日までの範囲の値を日付調整なしで DB2 表に格納するには、以下のいずれかの手法を使用できます。

- JDBC または SQLJ アプリケーションの場合は、setString メソッドを使用して値を String 入力パラメーターに割り当てます。入力パラメーターを VARCHAR としてキャストし、キャストの結果に対して DATE または TIMESTAMP 関数を実行します。次に、DATE または TIMESTAMP 関数の結果を DATE または TIMESTAMP 列に格納します。
- JDBC アプリケーションの場合は、Connection または DataSource プロパティーの sendDataAsIs を **true** に設定し、setString メソッドを使用して日付値またはタイム・スタンプ値を入力パラメーターに割り当てます。次に、SQL ステートメントを実行して String 値を DATE または TIMESTAMP 列に割り当てます。

例: DATE 列から 1582 年 10 月 10 日を日付調整なしで取り出します。

```
// DATETABLE has one date column called DATECOL with one row.
// Its value is 1582-10-10.
java.sql.ResultSet rs =
 statement.executeQuery(SELECT CHAR(DATECOL) FROM DATETABLE);
rs.next();
System.out.println(rs.getString(1)); // Value is retrieved as 1582-10-10
```

例: 1582 年 10 月 10 日を日付調整なしで DATE 列に格納します。

```
String s = "1582-10-10";
java.sql.Statement stmt = c.createStatement;
java.sql.PreparedStatement ps =
 c.prepareStatement("Insert INTO DATETABLE VALUES " +
  "(DATE(CAST (? AS VARCHAR)))");
ps.setString(1, s);
ps.executeUpdate(); // Value is inserted as 1582-10-10
```

JDBC および SQLJ アプリケーションでのタイム・スタンプ・データの損失

TIMESTAMP(*p*) 列の小数部分は、最大で 12 桁までの精度を持つことができます。Java タイム・スタンプ値の小数部分は、最大で 9 桁までの精度を持つことができます。列定義によっては、TIMESTAMP(*p*) 列を更新するとき、または TIMESTAMP(*p*) 列からデータを取得するときにデータ損失が発生する可能性があります。

入力データのデータ損失

setTimestamp 呼び出しを使ってタイム・スタンプ値を TIMESTAMP(*p*) 列に渡す場合、データ・ソースに送られる Java 値の最大精度は 9 です。 setTimestamp 呼び出しを使って、TIMESTAMP(*p*) をサポートしないデータ・ソースの TIMESTAMP 列にタイム・スタンプ値を渡す場合には、データ・ソースに送られる Java 値の最大精度は 6 です。 TIMESTAMP(*p*) 列への入力では、ターゲット列の精度が入力値の精度より小さい場合、データ・ソースはタイム・スタンプの小数部分の余分な桁を切り捨てます。

setString 呼び出しを使って入力値を渡す場合、9 より大きい精度の値をデータ・ソースに送ることができます。データ損失は、deferPrepares プロパティーおよび sendDataAsIs プロパティーの設定値によって次のように異なります。

- sendDataAsIs が `true` に設定されている場合、 IBM Data Server Driver for JDBC and SQLJ はストリングをデータ・ソースにそのまま送信します。このためタイム・スタンプ値の小数部分は 9 桁を超えることができます。 TIMESTAMP(*p*) 列の *p* の値が入力データの小数部分の桁数以上である場合には、データ損失が発生しません。
- deferPrepares が `true` に設定されている場合、UPDATE ステートメントが*初めて*実行されるときに、 IBM Data Server Driver for JDBC and SQLJ はストリングをデータ・ソースにそのまま送信します。このためタイム・スタンプ値の小数部分は 9 桁を超えることができます。 TIMESTAMP(*p*) 列の *p* の値が入力データの小数部分の桁数以上である場合には、データ損失が発生しません。

  それ以降の UPDATE ステートメントの実行では、 IBM Data Server Driver for JDBC and SQLJ はターゲット・データ・タイプが TIMESTAMP データ・タイプであることを判別できます。データ・ソースで TIMESTAMP(*p*) 列がサポートされる場合、ドライバーは最大精度 9 の java.sql.Timestamp 値に入力値を変換します。データ・ソースで TIMESTAMP(*p*) 列がサポートされない場合、ドライバーは最大精度 6 の java.sql.Timestamp 値に入力値を変換します。変換後の java.sql.Timestamp 値に比べて元の値の精度がより大きい場合、または TIMESTAMP(*p*) 列に比べて java.sql.Timestamp 値の精度がより大きい場合には、データ損失が発生します。
- deferPrepares が `false` に設定されている場合、 IBM Data Server Driver for JDBC and SQLJ はターゲット・データ・タイプが TIMESTAMP データ・タイプであることを判別できます。データ・ソースで TIMESTAMP(*p*) 列がサポートされる場合、ドライバーは最大精度 9 の java.sql.Timestamp 値に入力値を変換します。データ・ソースで TIMESTAMP(*p*) 列がサポートされない場合、ドライバーは最大精度 6 の java.sql.Timestamp 値に入力値を変換します。変換後の

java.sql.Timestamp 値に比べて元の値の精度がより大きい場合、または TIMESTAMP(*p*) 列に比べて java.sql.Timestamp 値の精度がより大きい場合には、データ損失が発生します。

- sendDataAsIs が `false` に設定されている場合、データ損失は deferPrepares の設定に応じて異なります。

setString 呼び出しを使用し、sendDataAsIs を `true` に設定することにより、入力タイム・スタンプ値のデータ損失を減らすことができます。ただし sendDataAsIs を `true` に設定する場合には、アプリケーションのデータ・タイプとデータ・ソースのデータ・タイプの間に互換性があることを確認する必要があります。

出力データのデータ損失

getTimestamp または getString 呼び出しを使って TIMESTAMP(*p*) 列からデータを取得する場合、 IBM Data Server Driver for JDBC and SQLJ はその値を最大精度 9 の java.sql.Timestamp 値に変換します。ソース値の精度が 9 より大きい場合、ドライバーは取得された値の小数部分を 9 桁に切り捨てます。

IBM Data Server Driver for JDBC and SQLJ のプロパティー

IBM Data Server Driver for JDBC and SQLJ のプロパティーは、特定のデータ・ソースへの接続の作成方法を定義します。ほとんどのプロパティーを、DataSource オブジェクトまたは Connection オブジェクトに関して設定できます。

プロパティーを設定する方法

プロパティーは以下の方法のいずれかで設定できます。

- setXXX メソッドを使用する。ここで `XXX` は、先頭文字が大文字になっている非修飾プロパティー名です。

  プロパティーは、com.ibm.db2.jcc.DB2BaseDataSource から継承される、以下の IBM Data Server Driver for JDBC and SQLJ 固有のインプリメンテーションに適用可能です。
  - com.ibm.db2.jcc.DB2SimpleDataSource
  - com.ibm.db2.jcc.DB2ConnectionPoolDataSource
  - com.ibm.db2.jcc.DB2XADataSource
- DriverManager.getConnection 呼び出しの *info* パラメーターの java.util.Properties 値。
- DriverManager.getConnection 呼び出しの *url* パラメーターの `java.lang.String` 値。

  int データ・タイプのプロパティーには、事前定義定数フィールド値を持つものがあります。定数フィールド値は、*url* パラメーターで使用する前に、整数値に解決する必要があります。例えば、com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL を *url* パラメーターに使用することはできません。しかし、com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL を含む URL ストリングを作成して、その URL ストリングをストリング変数に割り当てることができます。その後、以下のようにして、そのストリング変数を *url* パラメーターで使用できます。

```
String url =
 "jdbc:db2://sysmvs1.stl.ibm.com:5021" +
 "user=dbadm;password=dbadm;" +
 "traceLevel=" +
 (com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL) + ";";

 Connection con =
   java.sql.DriverManager.getConnection(url);
```

サポートされるすべてのデータベース製品に共通の IBM Data Server Driver for JDBC and SQLJ のプロパティー

大半の IBM Data Server Driver for JDBC and SQLJ のプロパティーは、ドライバーがサポートするすべてのデータベース製品に適用されます。

特に注記されていない限り、すべてのプロパティーは com.ibm.db2.jcc.DB2BaseDataSource にあります。

それらのプロパティーは以下のとおりです。

allowNextOnExhaustedResultSet

ResultSet の最終行の後に配置された前方スクロール・カーソルに関する ResultSet.next() 呼び出しが IBM Data Server Driver for JDBC and SQLJ でどのように処理されるかを指定します。このプロパティーのデータ・タイプは int です。

可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)

TYPE_FORWARD_ONLY と定義された ResultSet に関する ResultSet.next() は、カーソルが ResultSet の最終行の後に配置された場合には `false` を戻します。カーソルがオープン状態かクローズ状態かにかかわらず、`false` が戻されます。

DB2BaseDataSource.NO (2)

TYPE_FORWARD_ONLY と定義された ResultSet では、ResultSet.next() が呼び出され、カーソルが ResultSet の最終行の後に配置された場合には、″Invalid operation: result set is closed.″ というエラー・テキストとともに java.sql.SQLException がドライバーによってスローされます。これはデフォルトです。

atomicMultiRowInsert

PreparedStatement メソッドを使って表を変更するバッチ操作がアトミックであるか、それとも非アトミックであるかを指定します。このプロパティーのデータ・タイプは int です。

DB2 for z/OS への接続の場合、このプロパティーはバッチ INSERT または MERGE 操作にのみ適用されます。

DB2 Database for Linux, UNIX, and Windows または IBM Informix Dynamic Server への接続の場合、このプロパティーはバッチ INSERT、MERGE、UPDATE、または DELETE 操作に適用されます。

可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)

バッチ操作はアトミックです。バッチでのすべての行の挿入操作は、単

一の操作と見なされます。 1 つの行の挿入が失敗した場合、操作全体が失敗して BatchUpdateException が出されます。自動生成キーを戻すバッチ・ステートメントを使用した場合、BatchUpdateException が出されて失敗します。

atomicMultiRowInsert を DB2BaseDataSource.YES (1) に設定した場合、次のようになります。

- 異種バッチでステートメントを実行することはできません。
- ターゲット・データ・ソースが DB2 for z/OS である場合、以下の操作を行うことはできません。
  - 32767 個を超える行をバッチで挿入すると、結果として BatchUpdateException が出されます。
  - 異なる行にある同じパラメーターに対して、以下のメソッドのうち複数を呼び出した場合には、結果として BatchUpdateException が出されます。
    - PreparedStatement.setAsciiStream
    - PreparedStatement.setCharacterStream
    - PreparedStatement.setUnicodeStream

DB2BaseDataSource.NO (2)

バッチによる挿入は非アトミックです。各行の挿入は、それぞれ別個の実行であると見なされます。それぞれの挿入操作が成功したことを示す情報は、Statement.executeBatch によって戻される int[] 配列に含まれます。

DB2BaseDataSource.NOT_SET (0)

バッチによる挿入は非アトミックです。各行の挿入は、それぞれ別個の実行であると見なされます。それぞれの挿入操作が成功したことを示す情報は、Statement.executeBatch によって戻される int[] 配列に含まれます。これはデフォルトです。

このプロパティーは SQLJ アプリケーションには影響を与えません。

blockingReadConnectionTimeout

接続ソケットの読み取りがタイムアウトになるまでの秒単位の時間。このプロパティーは IBM Data Server Driver for JDBC and SQLJ Type 4 接続にのみ適用され、接続が正常に確立した後でデータ・ソースに送信されるすべての要求に影響を与えます。デフォルトは 0 です。0 の値は、タイムアウトがないことを意味します。

clientRerouteAlternateServerName

クライアント・リルート用の 1 つ以上のサーバー名を指定します。このプロパティーのデータ・タイプは String です。

enableClientAffinitiesList=DB2BaseDataSource.YES (1) の場合、clientRerouteAlternateServerName に 1 次サーバーの名前と代替サーバーの名前が含まれている必要があります。 serverName および portNumber によって識別されるサーバーは、1 次サーバーです。そのサーバー名は clientRerouteAlternateServerName リストの先頭に置く必要があります。

複数のサーバー名を指定する場合は、サーバー名をコンマ (,) またはスペースで区切ります。 clientRerouteAlternateServerName に指定される値の数は、clientRerouteAlternatePortNumber に指定される値の数と一致していなければなりません。

clientRerouteAlternateServerName は、 DB2 Database for Linux, UNIX, and Windows に対する IBM Data Server Driver for JDBC and SQLJ Type 2 接続、および IBM Data Server Driver for JDBC and SQLJ Type 4 接続に適用されます。

clientRerouteAlternatePortNumber
クライアント・リルート用の 1 つ以上のポート番号を指定します。このプロパティーのデータ・タイプは String です。

enableClientAffinitiesList=DB2BaseDataSource.YES (1) の場合、clientRerouteAlternatePortNumber に 1 次サーバーのポート番号と代替サーバーのポート番号が含まれている必要があります。 serverName および portNumber によって識別されるサーバーは、1 次サーバーです。そのポート番号は clientRerouteAlternatePortNumber リストの先頭に置く必要があります。

複数のポート番号を指定する場合は、ポート番号をコンマ (,) またはスペースで区切ります。 clientRerouteAlternatePortNumber に指定される値の数は、clientRerouteAlternateServerName に指定される値の数と一致していなければなりません。

clientRerouteAlternatePortNumber は、 DB2 Database for Linux, UNIX, and Windows に対する IBM Data Server Driver for JDBC and SQLJ Type 2 接続、および IBM Data Server Driver for JDBC and SQLJ Type 4 接続に適用されます。

clientRerouteServerListJNDIName
転送サーバー情報の JNDI リポジトリー内の DB2ClientRerouteServerList インスタンスへの JNDI 参照を識別します。 clientRerouteServerListJNDIName は、IBM Data Server Driver for JDBC and SQLJ Type 4 接続と、DataSource インターフェースにより設定された接続にのみ適用されます。

clientRerouteServerListJNDIName の値がヌルでない場合、clientRerouteServerListJNDIName は以下の機能を提供します。
- 転送サーバーに関する情報を、複数の JVM にわたって保持できます。
- データ・ソースへの最初の接続が失敗した場合に代替サーバー・ロケーションを提供します。

clientRerouteServerListJNDIContext
DB2ClientRerouteServerList インスタンスのバインディングおよび参照に使用される JNDI コンテキストを指定します。 clientRerouteServerListJNDIContext は、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続と、DataSource インターフェースにより設定された接続にのみ適用されます。

clientRerouteServerListJNDIContext を設定しない場合、 IBM Data Server Driver for JDBC and SQLJ はシステム・プロパティーまたは jndi.properties ファイルを使って初期コンテキストを作成します。

clientRerouteServerListJNDIContext は、以下のメソッドを使用して**のみ**設定できます。

```
public void setClientRerouteServerListJNDIContext(javax.naming.Context registry)
```

databaseName

データ・ソースの名前を指定します。この名前は、接続 URL のデータベース部分として使用されます。名前は、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続か IBM Data Server Driver for JDBC and SQLJ Type 2 接続のどちらが使用されるかによって異なります。

IBM Data Server Driver for JDBC and SQLJ Type 4 接続の場合:

- 接続先が DB2 for z/OS サーバーの場合、databaseName 値は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```

- 接続先が DB2 Database for Linux, UNIX, and Windows サーバーの場合、databaseName 値は、インストール中に定義されたデータベース名になります。
- 接続を IDS サーバーに対して行う場合、*database* は、データベース名です。この名前は大/小文字を区別しません。サーバーにより、名前は小文字に変換されます。
- 接続先が IBM Cloudscape サーバーの場合、 databaseName 値は、データベースを含むファイルの完全修飾名です。この名前は、二重引用符 (") で囲まなければなりません。以下に例を示します。

  ```
  "c:/databases/testdb"
  ```

このプロパティーが設定されていない場合、ローカル・サイトへの接続が行われます。

IBM Data Server Driver for JDBC and SQLJ Type 2 接続の場合:

- serverName 接続プロパティーの値がヌルの場合、 databaseName 値は、インストール中に定義されるデータベース名になります。 serverName プロパティーの値がヌルではない場合、databaseName 値はデータベース別名になります。

decimalSeparator

10 進、浮動小数点、または 10 進浮動小数点データ値の場合の入出力の小数点記号を指定します。このプロパティーのデータ・タイプは int です。

sendDataAsIs プロパティーの値が `true` の場合、decimalSeparator は出力値にのみ影響を与えます。

可能な値は以下のとおりです。

DB2BaseDataSource.DECIMAL_SEPARATOR_NOT_SET (0)
小数点記号としてピリオドが使用されます。これはデフォルトです。

DB2BaseDataSource.DECIMAL_SEPARATOR_PERIOD (1)
小数点記号としてピリオドが使用されます。

DB2BaseDataSource.DECIMAL_SEPARATOR_COMMA (2)
小数点記号としてコンマが使用されます。

DECIMAL_SEPARATOR_COMMA を設定すると、10 進、浮動小数点、または 10 進浮動小数点値に対する ResultSet.getString の結果では、区

切り記号としてコンマが使用されます。ただし、10 進、浮動小数点、または 10 進浮動小数点値を戻す ResultSet.getXXX メソッドで取得された値に対して toString メソッドを実行した場合、結果では小数点記号として小数点が使用されます。

decimalStringFormat

SDK for Java のバージョンが 1.5 以降の場合、DECIMAL または DECFLOAT 列から取得されるデータのストリング形式を指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

DB2BaseDataSource.DECIMAL_STRING_FORMAT_NOT_SET (0)

IBM Data Server Driver for JDBC and SQLJ は、java.math.BigDecimal.toString メソッドから戻される形式で 10 進値を戻します。これはデフォルトです。

例えば、値 0.0000000004 は 4E-10 として戻されます。

DB2BaseDataSource.DECIMAL_STRING_FORMAT_TO_STRING (1)

IBM Data Server Driver for JDBC and SQLJ は、java.math.BigDecimal.toString メソッドから戻される形式で 10 進値を戻します。

例えば、値 0.0000000004 は 4E-10 として戻されます。

DB2BaseDataSource.DECIMAL_STRING_FORMAT_TO_PLAIN_STRING (2)

IBM Data Server Driver for JDBC and SQLJ は、java.math.BigDecimal.toPlainString メソッドから戻される形式で 10 進値を戻します。

例えば、値 0.0000000004 は 0.0000000004 として戻されます。

このプロパティーは、以前のバージョンの SDK for Java には影響を与えません。これらのバージョンの場合、 IBM Data Server Driver for JDBC and SQLJ は、java.math.BigDecimal.toPlainString メソッドから戻される形式で 10 進値を戻します。

defaultIsolationLevel

新規接続用のデフォルトのトランザクション分離レベルを指定します。このプロパティーのデータ・タイプは int です。 defaultIsolationLevel が DataSource で設定されている場合、その DataSource から作成されるすべての接続には、defaultIsolationLevel によって指定されたデフォルトの分離レベルがあります。

DB2 データ・ソースの場合、デフォルトは `java.sql.Connection.TRANSACTION_READ_COMMITTED` です。

IBM Informix Dynamic Server (IDS) データベースの場合、デフォルトはデータ・ソースのタイプによって決まります。次の表はデフォルトを示しています。

表 45. IDS データベースのデフォルトの分離レベル

データ・ソースのタイプ	デフォルトの分離レベル
ロギングを行う ANSI 準拠データベース	java.sql.Connection.TRANSACTION_SERIALIZABLE
ロギングを行わないデータベース	java.sql.Connection.TRANSACTION_READ_UNCOMMITTED
ロギングを行う非 ANSI 準拠データベース	java.sql.Connection.TRANSACTION_READ_COMMITTED

deferPrepares

Connection.prepareStatement メソッドの呼び出しにより、データ・ソースに対して SQL ステートメントを直ちに準備するか、または PreparedStatement.execute メソッドが実行されるまでステートメントの準備を据え置くかを指定します。このプロパティーのデータ・タイプは boolean です。

deferPrepares は、 DB2 Database for Linux, UNIX, and Windows に対する IBM Data Server Driver for JDBC and SQLJ Type 2 接続、および IBM Data Server Driver for JDBC and SQLJ Type 4 接続についてサポートされます。

可能な値は以下のとおりです。

true データ・ソースに対するステートメントの準備は、PreparedStatement.execute メソッドが実行されるまで行われません。これはデフォルトです。

false データ・ソースに対するステートメントの準備は、Connection.prepareStatement メソッドが実行されると行われます。

準備操作を据え置くと、ネットワークの遅延を減らすことができます。ただし、準備操作を据え置く場合、入力データ・タイプが表列のタイプと一致していることを確認する必要があります。

説明

データ・ソースの説明。このプロパティーのデータ・タイプは String です。

downgradeHoldCursorsUnderXa

WITH HOLD が定義されたカーソルを XA 接続のもとでオープンできるかどうかを指定します。

downgradeHoldCursorsUnderXa は以下に適用されます。

- DB2 for z/OS サーバーに対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 Database for Linux, UNIX, and Windowsサーバーに対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続または IBM Data Server Driver for JDBC and SQLJ Type 2 接続

デフォルトは `false` です。つまり、WITH HOLD が定義されたカーソルを XA 接続のもとでオープンできません。そのカーソルをオープンしようとすると、例外がスローされます。

downgradeHoldCursorsUnderXa を `true` に設定した場合、WITH HOLD が定義されたカーソルを XA 接続のもとでオープンできます。ただし、カーソルには次のような制限があります。

- カーソルが XA 接続のもとでオープンされている場合、そのカーソルでは WITH HOLD 動作はありません。カーソルは XA End でクローズされます。
- ローカル・トランザクションで XA Start よりも前にオープンしているカーソルは、XA Start でクローズされます。

driverType

DataSource インターフェースでは、接続に使用するドライバーを判別します。このプロパティーのデータ・タイプは int です。有効な値は 2 または 4 です。 2 がデフォルトです。

enableClientAffinitiesList

カスケード・フェイルオーバーをサポートするためのクライアント・アフィニティーが IBM Data Server Driver for JDBC and SQLJで有効化されるかどうかを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)

IBM Data Server Driver for JDBC and SQLJ はカスケード・フェイルオーバー・サポート用のクライアント・アフィニティーを有効にします。これは、clientRerouteAlternateServerName および clientRerouteAlternatePortNumber プロパティーで指定されているサーバーだけが再試行されることを意味します。ドライバーは他のどのサーバーに対しても再接続を試行しません。

例えば、clientRerouteAlternateServerName に以下のストリングが含まれるとします。

```
host1,host2,host3
```

また、clientRerouteAlternatePortNumber に以下のストリングも含まれるとします。

```
port1,port2,port3
```

クライアント・アフィニティーを使用可能にすると、以下の順序で再試行されます。

1. host1:port1
2. host2:port2
3. host3:port3

DB2BaseDataSource.NO (2)

IBM Data Server Driver for JDBC and SQLJ はカスケード・フェイルオーバー・サポート用のクライアント・アフィニティーを無効にします。

DB2BaseDataSource.NOT_SET (0)

IBM Data Server Driver for JDBC and SQLJ はカスケード・フェイルオーバー・サポート用のクライアント・アフィニティーを無効にします。これはデフォルトです。

maxRetriesForClientReroute および retryIntervalForClientReroute プロパティーの影響は、enableClientAffinitiesList が有効になっているかどうかによって異なります。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

enableNamedParameterMarkers

名前付きパラメーター・マーカーのサポートを IBM Data Server Driver for JDBC and SQLJ で使用可能にするかどうかを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)

名前付きパラメーター・マーカーのサポートが IBM Data Server Driver for JDBC and SQLJ で使用可能になります。

DB2BaseDataSource.NO (2)

名前付きパラメーター・マーカーのサポートが IBM Data Server Driver for JDBC and SQLJ で使用不可になります。

ドライバーは名前付きパラメーター・マーカー付きの SQL ステートメントを変更せずにターゲット・データ・ソースに送ります。ステートメントが成功するか失敗するかは、例えば以下のような、多くの要因によって決まります。

- ターゲット・データ・ソースで名前付きパラメーター・マーカーがサポートされるかどうか
- deferPrepares プロパティー値が true または false のどちらであるか
- sendDataAsIs プロパティー値が true または false のどちらであるか

推奨: 名前付きパラメーター・マーカーを使用するアプリケーションでの予期されない動作を防ぐために、enableNamedParameterMarkers を YES に設定してください。

DB2BaseDataSource.NOT_SET (0)

この動作は、DB2BaseDataSource.NO (2) の場合の動作と同じになります。これはデフォルトです。

enableSeamlessFailover

IBM Data Server Driver for JDBC and SQLJ でクライアント・リルート用のシームレス・フェイルオーバーを使用するかどうかを指定します。このプロパティーのデータ・タイプは int です。

DB2 for z/OS への接続の場合、enableSysplexWLB を true に設定すると、enableSeamlessFailover は影響を与えません。 IBM Data Server Driver for JDBC and SQLJ は enableSeamlessFailover の設定値に関わりなくシームレス・フェイルオーバーを使用します。

enableSeamlessFailover に設定できる値は次のとおりです。

DB2BaseDataSource.YES (1)

IBM Data Server Driver for JDBC and SQLJはシームレス・フェイルオーバーを使用します。つまり、以下の条件が満たされる場合には、失敗した接続が正常に再確立された後、ドライバーはエラー・コード -4498 とともに SQLException をスローしません。

- 障害の発生時に、接続がトランザクション用には使用されていなかった。
- グローバル一時表などのグローバル・リソース、オープン状態の保留カーソル、または他のサーバーへのシームレス・フェイルオーバーを妨げる接続状態が残存しない。

シームレス・フェイルオーバーが行われる場合、新しいデータ・ソースへの接続が確立された後、ドライバーは元の接続の障害時に処理されていた SQL ステートメントを再発行します。

推奨: アプリケーションが保留カーソルを使用する場合、enableSeamlessFailover を DB2BaseDataSource.YES に設定する際には、

queryCloseImplicit プロパティーを DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NO (2) に設定してください。

DB2BaseDataSource.NO (2)
IBM Data Server Driver for JDBC and SQLJ はシームレス・フェイルオーバーを使用しません。

この設定が有効になっていると、サーバーがダウンした場合にドライバーは代替サーバーへのフェイルバックまたはフェイルオーバーを試みます。フェイルバックまたはフェイルオーバーが成功すると、ドライバーはエラー・コード -4498 の SQLException をスローします。これは、接続に失敗したものの正常に再確立されたことを示します。エラー・コード -4498 の SQLException は、接続の失敗が発生したときのトランザクションを再試行するようにアプリケーションに通知します。ドライバーが接続を再確立できない場合、エラー・コード -4499 の SQLException がスローされます。

DB2BaseDataSource.NOT_SET (0)
IBM Data Server Driver for JDBC and SQLJ はシームレス・フェイルオーバーを使用しません。これはデフォルトです。

fetchSize
Statement オブジェクトから生成される ResultSet オブジェクトのデフォルトのフェッチ・サイズを指定します。このプロパティーのデータ・タイプは int です。

fetchSize のデフォルトは Statement.setFetchSize メソッドでオーバーライドできます。 fetchSize プロパティーは、fetchSize 設定時に既に存在している Statement オブジェクトには影響を与えません。 IBM Data Server Driver for JDBC and SQLJ Type 4 接続、および DB2 Database for Linux, UNIX, and Windows データ・ソースへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合、fetchSize は両方向スクロール・カーソルにのみ影響を与えます。 DB2 for z/OS データ・ソースへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合、fetchSize は両方向スクロール・カーソルおよび前方スクロール・カーソルに影響を与えます。

fetchSize に指定できる値は次のとおりです。

0 または *positive-integer*
新しく作成される Statement オブジェクトのデフォルトの *fetchSize* 値。 fetchSize プロパティー値が無効な場合、 IBM Data Server Driver for JDBC and SQLJはデフォルトの *fetchSize* 値を 0 に設定します。

DB2BaseDataSource.FETCHSIZE_NOT_SET (-1)
Statement オブジェクトのデフォルトの *fetchSize* 値が 0 であることを示します。これがプロパティーのデフォルトです。

fetchSize プロパティーは、以下の点で queryDataSize プロパティーと異なります。

- fetchSize プロパティーは、 IBM Data Server Driver for JDBC and SQLJ でサポートされているすべてのデータ・ソースへの接続に適用されます。 queryDataSize プロパティーは、 DB2 for z/OS および DB2 Database for Linux, UNIX, and Windows への IBM Data Server Driver for JDBC and

SQLJ Type 4 接続と、 DB2 Database for Linux, UNIX, and Windows への IBM Data Server Driver for JDBC and SQLJ Type 2 接続にだけ適用されます。

- 両方向スクロール・カーソルの場合、fetchSize はデータ・ソースから返されるデータの量を制御します。データ・ソースからの単一の応答の内容は、常に最大で fetchSize 行までとなります。queryDataSize プロパティーは返されるデータの量の合計に影響しません。
- 前方スクロール・カーソルの場合、単一の応答でデータ・ソースから返されるデータの量は、queryDataSize によってのみ決まります。 fetchSize プロパティーは前方スクロール・カーソルに影響しません。

fullyMaterializeLobData

FETCH 操作のときにドライバーが LOB ロケーターを検索するかどうかを示します。このプロパティーのデータ・タイプは boolean です。

fullyMaterializeLobData の影響は、データ・ソースが連続ストリーミング (動的データ形式ともいう) をサポートしているかどうかによって異なります。

- データ・ソースが連続ストリーミングをサポートしていない場合:

  fullyMaterializeLobData の値が `true` の場合、行がフェッチされるときに LOB データは JDBC ドライバー内で完全にマテリアライズされます。この値が `false` の場合、LOB データはストリームされます。ドライバーは内部でロケーターを使用し、必要に応じて LOB データをチャンクごとに検索します。大量のデータを収容している LOB を検索する場合、この値を `false` に設定するように強くお勧めします。デフォルトは `true` です。
- データ・ソースが連続ストリーミングをサポートしている場合:

  progressiveStreaming プロパティーが `DB2BaseDataSource.YES` または `DB2BaseDataSource.NOT_SET` に設定されていると、JDBC ドライバーは fullyMaterializeLobData の値を無視します。

このプロパティーは、ストアード・プロシージャーのパラメーターや、両方向スクロール・カーソルを使用してフェッチされる LOB には影響しません。 LOB ストアード・プロシージャー・パラメーターは常に完全にマテリアライズされます。両方向スクロール・カーソルを使用してフェッチされる LOB は、連続ストリーミングが無効の場合は LOB ロケーターを使用します。

loginTimeout

データ・ソースへの接続に対する秒単位の最大待ち時間。 loginTimeout で指定された秒数が経過すると、ドライバーはデータ・ソースへの接続をクローズします。このプロパティーのデータ・タイプは int です。デフォルトは 0 です。値 0 は、タイムアウト値がデフォルトのシステム・タイムアウト値であることを意味します。このプロパティーは、 DB2 for z/OS上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続ではサポートされません。

logWriter

DataSource オブジェクトのすべてのロギングおよびトレース・メッセージが出力される文字出力ストリーム。このプロパティーのデータ・タイプは java.io.PrinterWriter です。デフォルト値はヌルです。この場合、DataSource のロギングもトレースも出力されません。

maxRetriesForClientReroute
自動クライアント・リルート中に、データ・ソースへの 1 次接続に失敗した場合の再試行回数を制限します。

このプロパティーのデータ・タイプは int です。

IBM Data Server Driver for JDBC and SQLJ が maxRetriesForClientReroute プロパティーを使用するのは、retryIntervalForClientReroute プロパティーも設定されている場合のみです。

enableClientAffinitiesList が DB2BaseDataSource.NO (2) に設定される場合、1 次サーバーおよび代替サーバーへの接続の試行は、1 回の再試行としてカウントされます。 enableClientAffinitiesList が DB2BaseDataSource.YES (1) に設定される場合、clientRerouteAlternateServerName および clientRerouteAlternatePortNumber 値によって指定されている各サーバーは、maxRetriesForClientReroute によって指定されている回数再試行されます。

maxRetriesForClientReroute のデフォルト値は、enableClientAffinitiesList が DB2BaseDataSource.NO (2) の場合は 0、enableClientAffinitiesList が DB2BaseDataSource.YES (1) の場合は 3 です。

password
接続の確立に使用するパスワード。このプロパティーのデータ・タイプは String です。 DataSource インターフェースを使用して接続を確立するときは、DataSource.getConnection メソッドを次の形式で呼び出すと、このプロパティー値をオーバーライドすることができます。

```
getConnection(user, password);
```

portNumber
DRDA サーバーが要求を listen するポート番号。このプロパティーのデータ・タイプは int です。

progressiveStreaming
データ・ソース上で連続ストリーミングがサポートされている場合に、JDBC ドライバーが連続ストリーミングを使用するかどうかを指定します。

DB2 for z/OS バージョン 9.1 以降では、LOB および XML オブジェクトの連続ストリーミングがサポートされます。 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および IBM Informix Dynamic Server (IDS) バージョン 11.50 以降では、LOB の連続ストリーミングがサポートされます。

連続ストリーミング (動的データ形式ともいう) を使用すると、LOB または XML オブジェクトのサイズに基づいて、LOB または XML のデータを戻すのに最も効率のよいモードがデータ・ソースで動的に判別されます。 streamBufferSize パラメーターの値は、データが戻されるときにマテリアライズ化されるかどうかを決定します。

progressiveStreaming のデータ・タイプは int です。有効な値は DB2BaseDataSource.YES (1) および DB2BaseDataSource.NO (2) です。 progressiveStreaming プロパティーが指定されない場合、progressiveStreaming 値は DB2BaseDataSource.NOT_SET (0) になります。

連続ストリーミングをサポートするデータ・ソースへの接続の場合に、progressiveStreaming の値が DB2BaseDataSource.YES または

DB2BaseDataSource.NOT_SET であると、JDBC ドライバーは連続ストリーミングを使用して LOB および XML データを戻します。

progressiveStreaming の値が DB2BaseDataSource.NO であるか、またはデータ・ソースが連続ストリーミングをサポートしない場合、JDBC ドライバーが LOB または XML データを戻す方法は、fullyMaterializeLobData プロパティーの値によって異なります。

queryCloseImplicit

すべての行をフェッチした後、すぐにカーソルをクローズするかどうかを指定します。 queryCloseImplicit が適用されるのは、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続 経由の DB2 for z/OS バージョン 8 以降への接続、および IBM Data Server Driver for JDBC and SQLJ Type 4 接続 または IBM Data Server Driver for JDBC and SQLJ Type 2 接続 経由の DB2 Database for Linux, UNIX, and Windows バージョン 9.7 以降への接続のみです。可能な値は以下のとおりです。

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES (1)

すべての行がフェッチされた後、直ちにカーソルをクローズします。

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES の値のほうが優れたパフォーマンスを提供できます。なぜなら、この設定の結果としてネットワーク・トラフィックが減るからです。

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NO (2)

すべての行がフェッチされた後、直ちにカーソルをクローズしません。

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_COMMIT (3)

以下のアクションを実行します。

- すべての行がフェッチされた後、暗黙的にカーソルをクローズします。
- アプリケーションが自動コミット・モードである場合、現在の作業単位のデータ・ソースにコミット要求を暗黙的に送ります。

重要: この値を設定した場合、他のリソースが影響を受ける可能性があります (これは明示的なコミット操作によって他のリソースが影響を受ける場合があるのと同じです)。例えば、他の非保留カーソルがクローズされ、LOB ロケーターがスコープ外になり、連続参照がリセットされて、両方向スクロール・カーソルの位置が失われます。

制約事項: QUERY_CLOSE_IMPLICIT_COMMIT の動作には、次のような制限があります。

- この動作は、アプリケーションによって出される SELECT ステートメントにのみ適用されます。 IBM Data Server Driver for JDBC and SQLJ によって生成される SELECT ステートメントには適用されません。
- QUERY_CLOSE_IMPLICIT_COMMIT が設定されている場合、アプリケーションが自動コミット・モードでなければ、ドライバーはデフォルトの動作 (QUERY_CLOSE_IMPLICIT_NOT_SET 動作) を使用します。 QUERY_CLOSE_IMPLICIT_COMMIT がデフォルト動作である場合、ドライバーは QUERY_CLOSE_IMPLICIT_YES 動作を使用します。

- QUERY_CLOSE_IMPLICIT_COMMIT が設定されている場合、データ・ソースで QUERY_CLOSE_IMPLICIT_COMMIT 動作がサポートされていなければ、ドライバーは QUERY_CLOSE_IMPLICIT_YES 動作を使用します。
- この動作はバッチ・ステートメントではサポートされません。
- 接続がローカル・トランザクションに含まれる場合にのみ、この動作は XA 接続でサポートされます。

DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NOT_SET (0)
これはデフォルトです。以下の表は、それぞれの種類のデータ・ソースに対する接続での動作を示しています。

データ・ソース	バージョン	データ共用環境	動作
DB2 for z/OS	バージョン 9 (APAR PK68746 を適用)	データ共用なし、またはデータ共用グループにおいてバージョン 8 メンバーとの共存モードなし	QUERY_CLOSE_IMPLICIT_COMMIT
DB2 for z/OS	バージョン 9 (APAR PK68746 未適用)	データ共用なし、またはデータ共用グループにおいてバージョン 8 メンバーとの共存モードなし	QUERY_CLOSE_IMPLICIT_YES
DB2 for z/OS	バージョン 9 (APAR PK68746 を適用)	データ共用グループにおいて、バージョン 8 メンバーとの共存モード	QUERY_CLOSE_IMPLICIT_COMMIT
DB2 for z/OS	バージョン 9 (APAR PK68746 未適用)	データ共用グループにおいて、バージョン 8 メンバーとの共存モード	QUERY_CLOSE_IMPLICIT_YES
DB2 for z/OS	バージョン 8 (APAR PK68746 適用または未適用)		QUERY_CLOSE_IMPLICIT_YES
DB2 Database for Linux, UNIX, and Windows	バージョン 9.7		QUERY_CLOSE_IMPLICIT_YES

`DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES (1)` および `DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_NO (2)`。デフォルトは `DB2BaseDataSource.QUERY_CLOSE_IMPLICIT_YES` です。

resultSetHoldability
コミット操作後もカーソルをオープンしたままにするかどうかを指定します。このプロパティーのデータ・タイプは int です。有効な値は以下のとおりです。

DB2BaseDataSource.HOLD_CURSORS_OVER_COMMIT (1)
コミット操作後にカーソルをオープンしたままにします。

この設定は、分散 (XA) トランザクションに含まれる接続には無効です。

DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT (2)
コミット操作後にカーソルをクローズします。

DB2BaseDataSource.NOT_SET (0)
これはデフォルト値です。動作は次のとおりです。

- 分散 (XA) トランザクションに含まれる接続の場合、カーソルはコミット操作後にクローズされます。
- 分散トランザクションに含まれない接続の場合、
  - すべてのバージョンの DB2 for z/OS、 DB2 Database for Linux, UNIX, and Windows、または DB2 for i サーバーへの接続、あるいは Cloudscape バージョン 8.1 以降のサーバーへの接続では、カーソルはコミット操作後にオープンされたままになります。
  - すべてのバージョンの IBM Informix Dynamic Server への接続、またはバージョン 8.1 より前のバージョンの Cloudscape への接続では、カーソルはコミット操作後にクローズされます。

retryIntervalForClientReroute
自動クライアント・リルートの場合、接続の再試行を行う間隔を秒単位の時間で指定します。

このプロパティーのデータ・タイプは int です。

IBM Data Server Driver for JDBC and SQLJ が retryIntervalForClientReroute プロパティーを使用するのは、maxRetriesForClientReroute プロパティーも設定されている場合のみです。

maxRetriesForClientReroute または retryIntervalForClientReroute が設定されていない場合、 IBM Data Server Driver for JDBC and SQLJ は再試行を 10 分間実行します。

enableClientAffinitiesList が DB2BaseDataSource.NO (2) に設定される場合、1 次サーバーおよび代替サーバーへの接続の試行は、1 回の再試行としてカウントされます。ドライバーは接続を再試行する前に、retryIntervalForClientReroute で指定されている秒数待機します。 enableClientAffinitiesList が DB2BaseDataSource.YES (1) に設定されている場合、clientRerouteAlternateServerName および clientRerouteAlternatePortNumber 値によって指定されている各サーバーは、retryIntervalForClientReroute によって指定されている秒数が経過した後で再試行されます。

retryIntervalForClientReroute のデフォルト値は 0 です。

securityMechanism
DRDA セキュリティー・メカニズムを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

CLEAR_TEXT_PASSWORD_SECURITY (3)
ユーザー ID およびパスワード

USER_ONLY_SECURITY (4)
ユーザー ID のみ

ENCRYPTED_PASSWORD_SECURITY (7)
ユーザー ID および暗号化されたパスワード

ENCRYPTED_USER_AND_PASSWORD_SECURITY (9)
暗号化されたユーザー ID およびパスワード

KERBEROS_SECURITY (11)
Kerberos。この値は IDS への接続には適用されません。

ENCRYPTED_USER_AND_DATA_SECURITY (12)
暗号化されたユーザー ID および暗号化されたセキュリティー・センシティブ・データ。この値は DB2 for z/OS への接続だけに適用されます。

ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY (13)
暗号化されたユーザー ID とパスワード、および暗号化されたセキュリティー・センシティブ・データ。この値は IDS への接続には適用されません。

PLUGIN_SECURITY (15)
プラグイン・セキュリティー。この値は DB2 Database for Linux, UNIX, and Windows への接続だけに適用されます。

ENCRYPTED_USER_ONLY_SECURITY (16)
暗号化されたユーザー ID。この値は IDS への接続には適用されません。

このプロパティーを指定すると、指定されたセキュリティー・メカニズムのみが使用されます。セキュリティー・メカニズムが接続でサポートされていない場合、例外が出されます。

securityMechanism のデフォルト値は、CLEAR_TEXT_PASSWORD_SECURITY です。サーバーが、CLEAR_TEXT_PASSWORD_SECURITY をサポートしない一方で、ENCRYPTED_USER_AND_PASSWORD_SECURITY をサポートするときは、 IBM Data Server Driver for JDBC and SQLJ ドライバーはセキュリティー・メカニズムを ENCRYPTED_USER_AND_PASSWORD_SECURITY に更新し、サーバーへの接続を試みます。リクエスターとサーバー間でこれ以外のセキュリティー・メカニズム・サポートの不一致がある場合には、エラーになります。

sendDataAsIs
IBM Data Server Driver for JDBC and SQLJ が入力パラメーター値をターゲット列のデータ・タイプに変換しないことを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは `false` です。

このプロパティーを使用する必要があるのは、アプリケーション内のデータ・タイプが、対応するデータベース表内のデータ・タイプに常に一致するようにしているアプリケーションの場合のみです。

serverName
データ・ソースのホスト名または TCP/IP アドレス。このプロパティーのデータ・タイプは String です。

sslConnection
IBM Data Server Driver for JDBC and SQLJ が SSL ソケットを使用してデータ・ソースに接続するかどうかを指定します。 sslConnection を `true` に設定すると、SSL ソケットが接続で使用されます。 sslConnection を `false` に設定すると、プレーン・ソケットが接続で使用されます。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

sslTrustStoreLocation

SSL 接続のサーバー証明書を含む、クライアント上の Java トラストストアの名前を指定します。

IBM Data Server Driver for JDBC and SQLJ がこのオプションを使用できるのは、sslConnection プロパティーが `true` に設定されている場合だけです。

sslTrustStore が設定されており、なおかつ sslConnection が `true` に設定されている場合、 IBM Data Server Driver for JDBC and SQLJ は javax.net.ssl.trustStore Java プロパティーの値ではなく、sslTrustStoreLocation 値を使用します。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

sslTrustStorePassword

SSL 接続のサーバー証明書を含む、クライアント上の Java トラストストアのパスワードを指定します。

IBM Data Server Driver for JDBC and SQLJ がこのオプションを使用できるのは、sslConnection プロパティーが `true` に設定されている場合だけです。

sslTrustStorePassword が設定されており、なおかつ sslConnection が `true` に設定されている場合、 IBM Data Server Driver for JDBC and SQLJ は javax.net.ssl.trustStorePassword Java プロパティーの値ではなく、sslTrustStorePassword 値を使用します。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

timestampFormat

TIMESTAMP 列に対する `ResultSet.getString` または `CallableStatement.getString` メソッドの結果が戻される形式を指定します。timestampFormat のデータ・タイプは int です。

timestampFormat に指定できる値は次のとおりです。

定数	整数値	形式
`com.ibm.db2.jcc.DB2BaseDataSource.ISO`	1	*yyyy-mm-dd-hh.mm.ss.nnnnnnnnn*[1]
`com.ibm.db2.jcc.DB2BaseDataSource.JDBC`	5	*yyyy-mm-dd hh:mm:ss.nnnnnnnnn*[1]

注:

1. タイム・スタンプの小数部分の桁数は、ソース表の TIMESTAMP(*p*) 列の精度によって異なります。 *p*<9 の場合、*p* 桁が戻されます。 *p*>=9 の場合、9 桁が戻され、残りの桁は切り捨てられます。

デフォルトは `com.ibm.db2.jcc.DB2BaseDataSource.JDBC` です。

timestampFormat は、出力の形式にのみ影響します。

timestampPrecisionReporting

TIMESTAMP 値について、Resultset.getString 呼び出しの結果から後続ゼロを切り捨てるかどうかを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

TIMESTAMP_JDBC_STANDARD (1)
TIMESTAMP 値について、Resultset.getString 呼び出しの結果から後続ゼロを切り捨てます。これはデフォルトです。

以下に例を示します。

- TIMESTAMP 値 2009-07-19-10.12.00.000000 は、検索の後、2009-07-19-10.12.00.0 に切り捨てられます。
- TIMESTAMP 値 2009-12-01-11.30.00.100000 は、検索の後、2009-12-01-11.30.00.1 に切り捨てられます。

TIMESTAMP_ZERO_PADDING (2)
TIMESTAMP 値について、Resultset.getString 呼び出しの結果から後続ゼロを切り捨てません。

traceDirectory
トレース情報の書き込み先のディレクトリーを指定します。このプロパティーのデータ・タイプは `String` です。 traceDirectory を指定すると、同一の DataSource 上にある複数の接続のトレース情報が、複数のファイルに書き込まれます。

traceDirectory を指定すると、接続のトレースは、traceFile_*origin_n* という名前のファイルへ書き込まれます。

n は、DataSource の *n* 番目の接続です。

origin は、使用中のログ書き込みプログラムの起点を示します。 *origin* に指定できる値は次のとおりです。

cpds DB2ConnectionPoolDataSource オブジェクトのログ書き込みプログラム。

driver DB2Driver オブジェクトのログ書き込みプログラム。

global DB2TraceManager オブジェクトのログ書き込みプログラム。

sds DB2SimpleDataSource オブジェクトのログ書き込みプログラム。

xads DB2XADataSource オブジェクトのログ書き込みプログラム。

traceFile プロパティーも指定した場合、traceDirectory 値は使用されません。

traceFile
IBM Data Server Driver for JDBC and SQLJ がトレース情報を書き込むファイルの名前を指定します。このプロパティーのデータ・タイプは `String` です。traceFile プロパティーは、ファイルに出力トレース・ストリームを送信する logWriter プロパティーの代わりに使用できます。

traceFileAppend
traceFile プロパティーによって指定されるファイルへ付加するか、あるいは上書きするかを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは `false` です。これは、traceFile プロパティーによって指定されるファイルが上書きされることを意味します。

traceLevel
何をトレースするかを指定します。このプロパティーのデータ・タイプは int です。

traceLevel プロパティーで、以下のトレースを 1 つ以上指定することができます。

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE (X'00')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS (X'01')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS (X'02')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS (X'04')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION (X'10')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS (X'20')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS (X'40')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA (X'80')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA (X'100')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS (X'200')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ (X'400')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (DB2 Database for Linux, UNIX, and Windows 用の IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ) (X'800')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS (X'2000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS (X'4000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS (X'8000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR (X'20000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS () (X'40000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL (X'FFFFFFFF')

複数のトレースを指定するには、以下の技法のいずれかを使用します。

- 2 つ以上のトレース値にビット単位 OR (|) 演算子を使用する。例えば、DRDA フローおよび接続呼び出しをトレースするには、traceLevel に以下の値を指定します。

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- ビット単位の補数 (~) 演算子とトレース値を使用して、特定のトレース以外のすべてを指定する。例えば、DRDA フロー以外のすべてをトレースする場合、traceLevel に以下の値を指定します。

  ```
  ~TRACE_DRDA_FLOWS
  ```

user

接続の確立に使用するユーザー ID。このプロパティーのデータ・タイプは String です。 DataSource インターフェースを使用して接続を確立するときは、DataSource.getConnection メソッドを次の形式で呼び出すと、このプロパティー値をオーバーライドすることができます。

```
getConnection(user, password);
```

xaNetworkOptimization

IBM Data Server Driver for JDBC and SQLJ Type 4 接続に関して XA ネットワーク最適化を使用可能にするかどうかを指定します。 1 つの Java プロセスから XA Start と XA End が発行され、別の Java プロセスから XA Prepare と XA Commit が発行されるような環境では、XA ネットワーク最適化を使用不可にする必要があるかもしれません。 XA ネットワーク最適化では、XA Prepare が XA End より先にデータ・ソースに到達する可能性があります。この場合、XAER_PROTO エラーが発生します。 XAER_PROTO エラーを防ぐには、XA ネットワーク最適化を使用不可にしてください。

デフォルトは true で、XA ネットワーク最適化が使用可能になります。xaNetworkOptimization が false の場合 (つまり XA ネットワーク最適化が使用不可の場合)、ドライバーは XA End 時にすべてのオープン・カーソルをクローズします。

xaNetworkOptimization は、DataSource オブジェクトで、あるいは getConnection 呼び出しの *url* パラメーターで設定することができます。接続の取得後に xaNetworkOptimization の値を変更することはできません。

com.ibm.db2.jcc.DB2ConnectionPoolDataSource.maxStatements
PooledConnection と関連付けられた内部ステートメント・キャッシュを制御します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

正の整数
PooledConnection について内部ステートメント・キャッシュを使用可能にし、 IBM Data Server Driver for JDBC and SQLJ がキャッシュ内でオープンしたままにするステートメントの数を指定します。

0 または負の整数
PooledConnection について内部ステートメント・キャッシングを使用不可にします。 0 がデフォルトです。

maxStatements は、PooledConnection オブジェクトが作成されるときにのみ、PooledConnection と関連付けられた内部ステートメント・キャッシュを制御します。 maxStatements は、既に存在している PooledConnection オブジェクトでのキャッシングには無効です。

maxStatements が適用されるのは、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

DB2 サーバーに共通の IBM Data Server Driver for JDBC and SQLJ プロパティー

IBM Data Server Driver for JDBC and SQLJ の一部のプロパティーは、 DB2 for z/OS および DB2 Database for Linux, UNIX, and Windows だけに適用されます。

特に注記されていない限り、すべてのプロパティーは com.ibm.db2.jcc.DB2BaseDataSource にあります。

それらのプロパティーは以下のとおりです。

clientAccountingInformation
接続の現在のクライアントに関するアカウンティング情報を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。このプロパティーのデータ・タイプは String です。 DB2 for z/OS サーバーの場合、最大長は 255 バイトです。 Java 空ストリング ("") はこの値として有効ですが、Java NULL 値は無効です。

このプロパティーは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

clientApplicationInformation
エンド・ユーザーのアプリケーションの、アプリケーションまたはトランザクションの名前を指定します。このプロパティーを使用して、クライアント・エン

ド・ユーザーの ID をアカウンティングおよびモニターのために提供できます。この値は、接続中に変更することもできます。このプロパティーのデータ・タイプは String です。 DB2 for z/OS サーバーの場合、最大長は 32 バイトです。 Java 空ストリング ("") はこの値として有効ですが、Java NULL 値は無効です。

このプロパティーは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

clientDebugInfo
CLIENT DEBUGINFO 接続属性の値を指定します。これは DB2 for z/OS サーバーに、接続を使用しているストアード・プロシージャーおよびユーザー定義関数がデバッグ・モードで実行していることを通知します。 CLIENT DEBUGINFO は、DB2 統合デバッガーにより使用されます。このプロパティーのデータ・タイプは String です。最大長は 254 バイトです。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

clientProgramId
エンド・ユーザーを識別するために使用できるクライアント・プログラム ID の値を指定します。このプロパティーのデータ・タイプは String で、長さは 80 バイトです。プログラム ID 値が 80 バイトより小さい場合、値にはブランクが埋め込まれます。

clientProgramName
クライアントの物理接続の継続期間中固定されるアプリケーション ID を指定します。このプロパティーの値は、 DB2 for z/OS サーバー上では相関 ID となります。データベース管理者はこのプロパティーを使用して、 DB2 for z/OS サーバー上の作業をクライアント・アプリケーションに関連付けることができます。このプロパティーのデータ・タイプは String です。最大長は 12 バイトです。この値を null にすると、 IBM Data Server Driver for JDBC and SQLJ によって db2jcc*thread-name* の値が提供されます。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

currentDegree
動的に準備される照会の実行の並列処理の度合いを指定します。このプロパティーのタイプは String です。 currentDegree 値は、データ・ソース上に CURRENT DEGREE 特殊レジスターを設定するために使用されます。currentDegree が設定されていない場合、データ・ソースに渡される値はありません。

currentFunctionPath
JDBC プログラム内にある SQL ステートメント内の非修飾の表データ・タイプ名および関数名を解決するために使用される、 SQL パスを指定します。このプロパティーのデータ・タイプは String です。 DB2 Database for Linux, UNIX, and Windows サーバーの場合の最大長は 254 バイトです。 DB2 for z/OS サーバーの場合の最大長は 2048 バイトです。値は、スキーマ名のコンマで区切られたリストになります。それらの名前は、通常 ID または区切り ID にできます。

currentMaintainedTableTypesForOptimization
動的 SQL 照会の処理をデータ・ソースが最適化するときに検討の対象にできるオブジェクトのタイプを特定する値を指定します。このレジスターには、表タイプを表すキーワードが含まれます。このプロパティーのデータ・タイプは String です。

currentMaintainedTableTypesForOptimization に指定できる値は次のとおりです。

ALL
すべてのマテリアライズ照会表が検討の対象になることを示します。

NONE
どのマテリアライズ照会表も検討の対象にならないことを示します。

SYSTEM
REFRESH DEFERRED されるシステム保守のマテリアライズ照会表のみが検討の対象となることを示します。

USER
REFRESH DEFERRED されるユーザー保守のマテリアライズ照会表のみが検討の対象となることを示します。

currentPackagePath
サーバー上のコレクションのコンマで区切られたリストを指定します。データベース・サーバーはこれらのコレクションから JDBC および SQLJ パッケージを検索します。

currentPackagePath および currentPackageSet プロパティーの優先順位規則は、CURRENT PACKAGESET および CURRENT PACKAGE PATH 特殊レジスターの優先順位規則に従います。

currentPackageSet
JDBC および SQLJ パッケージの検索に使用するコレクション ID を指定します。このプロパティーのデータ・タイプは String です。デフォルトは NULLID です。 currentPackageSet が設定されている場合、その値は jdbcCollection の値をオーバーライドします。

DB2Binder ユーティリティーを複数回実行することにより、データベース・サーバーで IBM Data Server Driver for JDBC and SQLJ の複数のインスタンスをインストールできます。 DB2binder ユーティリティーに含まれる -collection オプションを使用すると、各 IBM Data Server Driver for JDBC and SQLJ インスタンスのコレクション ID をインストーラーで指定できます。接続に対する IBM Data Server Driver for JDBC and SQLJ のインスタンスを選択するには、いずれかの IBM Data Server Driver for JDBC and SQLJ インスタンスのコレクション ID に一致する currentPackageSet 値を指定します。

currentPackagePath および currentPackageSet プロパティーの優先順位規則は、CURRENT PACKAGESET および CURRENT PACKAGE PATH 特殊レジスターの優先順位規則に従います。

currentRefreshAge
タイム・スタンプ期間の値を指定します。これは、マテリアライズ照会表を使用して照会の処理を最適化できるようにするために、REFRESH TABLE ステートメントをシステム保守の REFRESH DEFERRED マテリアライズ照会表で処理

してからの最大期間です。このプロパティーは、動的ステートメントのキャッシュの突き合わせに影響を与えます。このプロパティーのデータ・タイプは LONG です。

currentSchema
動的に準備される SQL ステートメントで非修飾データベース・オブジェクトを修飾するために使用されるデフォルトのスキーマ名を指定します。このプロパティーの値は、データベース・サーバーでの CURRENT SCHEMA 特殊レジスター内の値を設定します。スキーマ名は大文字小文字の区別があり、大文字で指定する必要があります。

cursorSensitivity
JDBC ResultSet の java.sql.ResultSet.TYPE_SCROLL_SENSITIVE 値を、基礎となるデータベース・カーソルの SENSITIVE DYNAMIC 属性、SENSITIVE STATIC 属性、または ASENSITIVE 属性にマップするかどうかを指定します。このプロパティーのデータ・タイプは int です。指定できる値は TYPE_SCROLL_SENSITIVE_STATIC (0)、TYPE_SCROLL_SENSITIVE_DYNAMIC (1)、または TYPE_SCROLL_ASENSITIVE (2) です。デフォルトは TYPE_SCROLL_SENSITIVE_STATIC です。

データ・ソースがセンシティブ動的両方向スクロール・カーソルをサポートしていない場合に、TYPE_SCROLL_SENSITIVE_DYNAMIC が要求されると、JDBC ドライバーは警告を累積し、感度を SENSITIVE STATIC にマップします。センシティブ静的カーソルをサポートしない DB2 for i データベース・サーバーの場合、java.sql.ResultSet.TYPE_SCROLL_SENSITIVE は常に SENSITIVE DYNAMIC にマップされます。

dateFormat
下記のことを指定します。

- DATE 列に対する PreparedStatement.setString メソッドの String 引数を指定する際の形式。
- DATE 列に対する ResultSet.getString または CallableStatement.getString メソッドの結果が返される形式。

dateFormat のデータ・タイプは int です。

dateFormat に指定できる値は次のとおりです。

定数	整数値	形式
com.ibm.db2.jcc.DB2BaseDataSource.ISO	1	*yyyy-mm-dd*
com.ibm.db2.jcc.DB2BaseDataSource.USA	2	*mm/dd/yyyy*
com.ibm.db2.jcc.DB2BaseDataSource.EUR	3	*dd.mm.yyyy*
com.ibm.db2.jcc.DB2BaseDataSource.JIS	4	*yyyy-mm-dd*

デフォルトは com.ibm.db2.jcc.DB2BaseDataSource.ISO です。

decimalRoundingMode
DB2 for z/OS バージョン 9 以降、または DB2 Database for Linux, UNIX, and Windows データベース・サーバー上での、10 進浮動小数点値の丸めモードを指定します。

可能な値は以下のとおりです。

DB2BaseDataSource.ROUND_DOWN (1)
0 の方向に丸めます (切り捨て)。破棄された数字は無視されます。

DB2BaseDataSource.ROUND_CEILING (2)
正の無限大方向に丸めます。破棄される数字がすべてゼロであるか、または符号が負の場合、破棄される数字の除去を除いて結果は未変更のままになります。それ以外の場合、結果の係数は 1 だけ増やされます。

DB2BaseDataSource.ROUND_HALF_EVEN (3)
値を最も近い値に丸めます。複数の値が等距離にある場合は、最後の数字が偶数になるように値を丸めます。破棄される数字が、左側の次の桁内の 1 の半分 (0.5) より大きい場合、結果の係数は 1 だけ増やされます。半分より小さい場合、結果の係数は調整されません (つまり、廃棄された数字は無視されます)。それ以外の場合、結果の係数の右端の数字が偶数であれば、その係数は変更されませんが、右端の数字が奇数であれば、その桁が偶数になるように 1 だけ増大 (切り上げ) します。

DB2BaseDataSource.ROUND_HALF_UP (4)
値を最も近い値に丸めます。複数の値が等距離にある場合は、0 の反対方向に値を丸めます。破棄される数字が、左側の次の桁内の 1 の半分 (0.5) 以上の場合、結果の係数は 1 だけ増やされます。そうでない場合、破棄された数字は無視されます。

DB2BaseDataSource.ROUND_FLOOR (6)
負の無限大方向に丸めます。破棄される数字がすべてゼロであるか、または符号が正の場合、破棄される数字の除去を除いて、結果は未変更のままになります。それ以外の場合、符号は負で、結果の係数は 1 だけ増やされます。

DB2BaseDataSource.ROUND_UNSET (-2147483647)
丸めモードは明示的に設定されませんでした。 IBM Data Server Driver for JDBC and SQLJ は、decimalRoundingMode を使用してデータ・ソース上で丸めモードを設定することはしません。

IBM Data Server Driver for JDBC and SQLJ はその丸めモードに以下の値を使用します。

- DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows データベース・サーバーの場合、10 進浮動小数点値の丸めモードは ROUND_HALF_EVEN です。

decimalRoundingMode が設定されている場合、decimalRoundingMode 値は、DB2 for z/OS データベース・サーバー上で CURRENT DECFLOAT ROUNDING MODE 特殊レジスターを設定するために使用されます。

enableRowsetSupport
データ・ソースが複数行 FETCH をサポートする場合、 IBM Data Server Driver for JDBC and SQLJ が前方スクロール・カーソルまたは両方向スクロール・カーソルに複数行 FETCH を使用するかどうかを指定します。このプロパティーのデータ・タイプは int です。

enableRowsetSupport が設定されている場合、その値は useRowsetCursor プロパティーの値をオーバーライドします。

可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)
以下を指定します。

- DB2 for z/OSへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続 では、データ・ソースが複数行 FETCH をサポートする場合、両方向スクロール・カーソルおよび前方スクロール・カーソルに複数行 FETCH が使用されます。
- IBM Data Server Driver for JDBC and SQLJ Type 4 接続、または DB2 Database for Linux, UNIX, and Windowsへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続では、データ・ソースが複数行 FETCH をサポートする場合、両方向スクロール・カーソルに複数行フェッチが使用されます。

DB2BaseDataSource.NO (2)
複数行フェッチが使用されないことを指定します。

DB2BaseDataSource.NOT_SET (0)
enableRowsetSupport プロパティーが設定されていない場合、以下を指定します。

- DB2 for z/OSへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続では、複数行フェッチは使用されません。
- IBM Data Server Driver for JDBC and SQLJ Type 4 接続、または DB2 Database for Linux, UNIX, and Windows への IBM Data Server Driver for JDBC and SQLJ Type 2 接続では、両方向スクロール・カーソルで複数行フェッチが使用されるかどうかは useRowsetCursor プロパティーによって決まります。

DB2 for z/OSへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続では、複数行フェッチに連続ストリーミングとの互換性がありません。そのため、FETCH 操作で連続ストリーミングが使用される場合、複数行 FETCH は使用されません。

encryptionAlgorithm
IBM Data Server Driver for JDBC and SQLJ が 56 ビットの DES (弱) 暗号化または 256 ビットの AES (強) 暗号化のどちらを使用するかを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

1 このドライバーは、56 ビット DES 暗号化を使用します。

2 データベース・サーバーが 256 ビット AES 暗号化をサポートしている場合、ドライバーはこの暗号化を使用します。256 ビット AES 暗号化は、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続専用です。

encryptionAlgorithm を指定できるのは、securityMechanism 値が `ENCRYPTED_PASSWORD_SECURITY` (7) または `ENCRYPTED_USER_AND_PASSWORD_SECURITY` (9) である場合のみです。

fullyMaterializeInputStreams
ストリームが、クライアントからデータ・ソースに送信される前に、完全にマテリアライズ化されるかどうかを示します。このプロパティーのデータ・タイプは boolean です。デフォルトは `false` です。

fullyMaterializeInputStreams の値が true の場合、JDBC ドライバーはストリームをサーバーに送信する前に完全にマテリアライズ化したことになります。

gssCredential
Kerberos セキュリティーを使用するデータ・ソースの場合に、他のプリンシパルから渡される委任証明書を指定します。このプロパティーのデータ・タイプは org.ietf.jgss.GSSCredential です。委任証明書は、例えばクライアントが WebSphere Application Server に接続し、次いでそれがデータ・ソースに接続するような、複数層の環境において使用されます。 GSSContext.getDelegCred メソッドを呼び出すことにより、このプロパティーの値をクライアントから取得します。 GSSContext は IBM Java Generic Security Service (GSS) API に含まれています。このプロパティーを設定する場合、 Mechanism および KerberosServerPrincipal プロパティーも設定しなければなりません。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

IBM Data Server Driver for JDBC and SQLJ での Kerberos セキュリティーの使用について詳しくは、『 IBM Data Server Driver for JDBC and SQLJ 使用時の Kerberos セキュリティーの使用』を参照してください。

kerberosServerPrincipal
Kerberos セキュリティーを使用するデータ・ソースの場合、 Kerberos 鍵配布センター (KDC) への登録時にデータ・ソースに使用される名前を指定します。このプロパティーのデータ・タイプは String です。

このプロパティーを適用できるのは IBM Data Server Driver for JDBC and SQLJ Type 4 接続だけです。

pdqProperties
IBM Data Server Driver for JDBC and SQLJ と pureQuery のクライアント最適化フィーチャーの間の対話を制御するプロパティーを指定します。

このプロパティーのデータ・タイプは String です。

pdqProperties プロパティーは、pureQuery のクライアント最適化フィーチャーを使用している場合**のみ**設定してください。 pdqProperties の有効な値について詳しくは、Integrated Data Management のインフォメーション・センターを参照してください。

readOnly
接続を読み取り専用にするかどうかを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは false です。

resultSetHoldabilityForCatalogQueries
DatabaseMetaData メソッドの代わりに実行される照会用のカーソルをコミット操作後もオープンしたままにするかどうかを指定します。このプロパティーのデータ・タイプは int です。

アプリケーションが DatabaseMetaData メソッドを実行すると、 IBM Data Server Driver for JDBC and SQLJ はターゲット・データ・ソースのカタログに対して照会を実行します。デフォルトでは、そのようなカーソルの保持可能性はアプリケーション・カーソルの保持可能性と同じです。カタログ照会に別の保持可能性を使用するには、resultSetHoldabilityForCatalogQueries プロパティーを使用します。可能な値は以下のとおりです。

DB2BaseDataSource.HOLD_CURSORS_OVER_COMMIT (1)
resultSetHoldability の設定とは無関係に、コミット操作後にカタログ照会用のカーソルをオープンしたままにします。

DB2BaseDataSource.CLOSE_CURSORS_AT_COMMIT (2)
resultSetHoldability の設定とは無関係に、コミット操作後にカタログ照会用のカーソルをクローズします。

DB2BaseDataSource.NOT_SET (0)
カタログ照会に resultSetHoldability 設定を使用します。これはデフォルト値です。

returnAlias
getTables などの表情報を戻す DatabaseMetaData メソッドについて、JDBC ドライバーが表の別名およびシノニムの行を戻すかどうかを指定します。 returnAlias のデータ・タイプは int です。可能な値は以下のとおりです。

0 表情報を戻す DatabaseMetaData メソッドからの出力内に、表の別名またはシノニムの行を戻しません。

1 別名またはシノニムを持つ表の場合、表情報を戻す DatabaseMetaData メソッドからの出力内に、表の行とともに、それらの表の別名およびシノニムの行を戻します。これはデフォルトです。

streamBufferSize
LOB または XML データをチャンク化するための、JDBC ドライバー・バッファーのサイズをバイト数で指定します。 JDBC ドライバーは、連続ストリーミングを使用するかどうかに関係なく、streamBufferSize 値を使用します。 streamBufferSize のデータ・タイプは int です。デフォルトは 1048576 です。

JDBC ドライバーが連続ストリーミングを使用する場合、LOB または XML データがマテリアライズ化されるのは、そのデータがバッファー内に収まり、このドライバーが fullyMaterializeLobData プロパティーを使用しない場合です。

DB2 for z/OS バージョン 9.1 以降では、LOB および XML オブジェクトの連続ストリーミングがサポートされます。 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および IBM Informix Dynamic Server (IDS) バージョン 11.50 以降では、LOB の連続ストリーミングがサポートされます。

supportsAsynchronousXARollback
IBM Data Server Driver for JDBC and SQLJ が非同期の XA ロールバック操作をサポートするかどうかを指定します。このプロパティーのデータ・タイプは int です。デフォルトは `DB2BaseDataSource.NO` (2) です。 BEA WebLogic Server アプリケーション・サーバーに対してアプリケーションを実行する場合、supportsAsynchronousXARollback を `DB2BaseDataSource.YES` (1) に設定します。

sysSchema
アプリケーションが DatabaseMetaData メソッドを呼び出す際に検索される、シャドー・カタログ表またはビューのスキーマを指定します。 sysSchema プロパティーは、以前は cliSchema という名称でした。

timeFormat
下記のことを指定します。

- TIME 列に対する PreparedStatement.setString メソッドの String 引数を指定する際の形式。
- TIME 列に対する ResultSet.getString または CallableStatement.getString メソッドの結果が戻される形式。

timeFormat のデータ・タイプは int です。

timeFormat に指定できる値は次のとおりです。

定数	整数値	形式
com.ibm.db2.jcc.DB2BaseDataSource.ISO	1	*hh:mm:ss*
com.ibm.db2.jcc.DB2BaseDataSource.USA	2	*hh:mm* am または *hh:mm* pm
com.ibm.db2.jcc.DB2BaseDataSource.EUR	3	*hh.mm.ss*
com.ibm.db2.jcc.DB2BaseDataSource.JIS	4	*hh:mm:ss*

デフォルトは com.ibm.db2.jcc.DB2BaseDataSource.ISO です。

useCachedCursor

PreparedStatement オブジェクトの基礎カーソルをキャッシュに入れて、PreparedStatement の後続の実行で再利用するかどうかを指定します。

useCachedCursor のデータ・タイプは boolean です。

useCachedCursor が true に設定されている場合、PreparedStatement オブジェクトのカーソルはキャッシュに入れられ、これによりパフォーマンスは向上します。 true がデフォルトです。

PreparedStatement オブジェクトが、それらの PreparedStatement オブジェクトの実行間に列のタイプまたは長さが変わる表にアクセスする場合は、useCachedCursor を false に設定します。

useJDBC4ColumnNameAndLabelSemantics

IBM Data Server Driver for JDBC and SQLJ が、ResultSetMetaData.getColumnName、ResultSetMetaData.getColumnLabel、および ResultSet.findColumn メソッド呼び出しで列ラベルを処理する方法を指定します。

可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)

IBM Data Server Driver for JDBC and SQLJ は、JDBC 4.0 仕様に準拠する以下の規則を使用して、ResultSetMetaData.getColumnName、ResultSetMetaData.getColumnLabel、および ResultSet.findColumn が戻す値を判別します。

- ResultSetMetaData.getColumnName によって戻される列名はデータベースからの名前です。
- ResultSetMetaData.getColumnLabel によって戻される列ラベルは SQL AS 節で指定されたラベルです。 SQL AS 節が指定されていない場合、ラベルは列の名前です。
- ResultSet.findColumn は、SQL AS 節で指定された列のラベルを入力としてとります。 SQL AS 節が指定されていない場合、ラベルは列名です。

- IBM Data Server Driver for JDBC and SQLJ は、SQL LABEL ON ステートメントによって割り当てられた列ラベルを使用しません。

これらの規則は、次のデータベース・システムへの接続の場合、 IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 以降に適用されます。

- DB2 for z/OS バージョン 8 以降
- DB2 Database for Linux, UNIX, and Windows バージョン 8.1 以降
- DB2 UDB for iSeries® V5R3 以降

以前のバージョンのドライバーまたはデータベース・システムの場合、useJDBC4ColumnNameAndLabelSemantics が DB2BaseDataSource.YES に設定されていても、DB2BaseDataSource.NO の useJDBC4ColumnNameAndLabelSemantics 値の規則が適用されます。

DB2BaseDataSource.NO (2)

IBM Data Server Driver for JDBC and SQLJ は以下の規則を使用して、ResultSetMetaData.getColumnName、ResultSetMetaData.getColumnLabel、および ResultSet.findColumn が戻す値を判別します。

データ・ソースが LABEL ON ステートメントをサポートしないか、またはソース列が LABEL ON ステートメントで定義されていない場合、以下の規則が適用されます。

- ResultSetMetaData.getColumnName によって戻される値は、SQL AS 節が指定されていない場合はデータベースからの名前です。SQL AS 節が指定されている場合、戻される値は列ラベルです。
- ResultSetMetaData.getColumnLabel によって戻される値は SQL AS 節で指定されたラベルです。SQL AS 節が指定されていない場合、戻される値は列の名前です。
- ResultSet.findColumn は列名を入力としてとります。

ソース列が LABEL ON ステートメントで定義されている場合、以下の規則が適用されます。

- ResultSetMetaData.getColumnName によって戻される値は、SQL AS 節が指定されていない場合はデータベースからの列名です。SQL AS 節が指定されている場合、戻される値は AS 節で指定された列ラベルです。
- ResultSetMetaData.getColumnLabel によって戻される値は LABEL ON ステートメントで指定されたラベルです。
- ResultSet.findColumn は列名を入力としてとります。

これらの規則は、 IBM Data Server Driver for JDBC and SQLJ のバージョン 3.50 より前の動作に準拠します。

DB2BaseDataSource.NOT_SET (0)

これがデフォルトの動作です。

IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 以前の場合、useJDBC4ColumnNameAndLabelSemantics のデフォルト動作は DB2BaseDataSource.NO の動作と同じです。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の場合:

- useJDBC4ColumnNameAndLabelSemantics のデフォルト動作は、次のデータベース・システムへの接続の場合、DB2BaseDataSource.YES の動作と同じです。
  - DB2 for z/OS バージョン 8 以降
  - DB2 Database for Linux, UNIX, and Windows バージョン 8.1 以降
  - DB2 UDB for iSeries V5R3 以降
- これらのデータベース・システムより前のバージョンへの接続の場合、useJDBC4ColumnNameAndLabelSemantics のデフォルト動作は DB2BaseDataSource.NO です。

com.ibm.db2.jcc.DB2ConnectionPoolDataSource.maxStatements

PooledConnection と関連付けられた内部ステートメント・キャッシュを制御します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

正の整数

PooledConnection について内部ステートメント・キャッシュを使用可能にし、 IBM Data Server Driver for JDBC and SQLJ がキャッシュ内でオープンしたままにするステートメントの数を指定します。

0 または負の整数

PooledConnection について内部ステートメント・キャッシングを使用不可にします。 0 がデフォルトです。

maxStatements は、PooledConnection オブジェクトが作成されるときにのみ、PooledConnection と関連付けられた内部ステートメント・キャッシュを制御します。 maxStatements は、既に存在している PooledConnection オブジェクトでのキャッシングには無効です。

maxStatements は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

DB2 for z/OSおよび IDS に共通の IBM Data Server Driver for JDBC and SQLJ プロパティー

IBM Data Server Driver for JDBC and SQLJ の一部のプロパティーは、 IBM Informix Dynamic Server (IDS) および DB2 for z/OS データベース・サーバーに適用されます。

IDS および DB2 for z/OS に適用されるプロパティーは以下のとおりです。

enableConnectionConcentrator

IBM Data Server Driver for JDBC and SQLJ の接続コンセントレーター機能を使用可能にするかどうかを示します。接続コンセントレーター機能を使用できるのは、 DB2 for z/OS サーバーへの接続の場合のみです。

enableConnectionConcentrator のデータ・タイプは boolean です。デフォルトは `false` です。ただし、enableSysplexWLB を `true` に設定した場合のデフォルトは `true` になります。

enableSysplexWLB

IBM Data Server Driver for JDBC and SQLJ の Sysplex ワークロード・バラン

シング機能を使用可能にするかどうかを示します。 enableSysplexWLB のデータ・タイプは boolean です。デフォルトは false です。

keepDynamic

既に準備済みの動的 SQL ステートメントを再利用できるように、コミット・ポイント後もその SQL ステートメントをデータ・ソースが動的ステートメント・キャッシュ内に維持し続けるかどうかを指定します。このプロパティーのデータ・タイプは int です。有効な値は DB2BaseDataSource.YES (1) および DB2BaseDataSource.NO (2) です。

keepDynamic プロパティーが指定されない場合、keepDynamic 値は DB2BaseDataSource.NOT_SET (0) になります。接続先が DB2 for z/OS サーバーの場合、プロパティーが設定されていなければ接続に対する動的ステートメントのキャッシングは行われません。接続先が IDS データ・ソースの場合、プロパティーが設定されていなければ接続に対する動的ステートメントのキャッシングが行われます。

keepDynamic は、DB2Binder -keepdynamic オプションと一緒に使用します。指定された keepDynamic プロパティーの値は、DB2Binder の実行時に指定された -keepdynamic 値と一致していなければなりません。

DB2 for z/OS データベース・サーバーの場合、動的ステートメントのキャッシングを行えるのは、データ・ソース上で EDM 動的ステートメント・キャッシュが使用可能になっている場合のみです。動的ステートメント・キャッシュを使用可能にするには、CACHEDYN サブシステム・パラメーターを DB2BaseDataSource.YES に設定する必要があります。

maxTransportObjects

関連 DataSource オブジェクトとのすべての接続で使用できるトランスポート・オブジェクトの最大数を指定します。 IBM Data Server Driver for JDBC and SQLJ はトランスポート・オブジェクトおよびグローバル・トランスポート・オブジェクト・プールを使用して、接続コンセントレーターおよび Sysplex ワークロード・バランシングをサポートします。データ・ソースへの各物理接続ごとに、トランスポート・オブジェクトが 1 つあります。

このプロパティーのデータ・タイプは int です。

enableConnectionConcentrator または enableSysplexWLB プロパティーを設定せず、接続コンセントレーターまたは Sysplex ワークロード・バランシングの使用が有効になっていない場合、maxTransportObjects 値は無視されます。

maxTransportObjects 値に達していない場合に、グローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクトが使用可能でないと、新規のトランスポート・オブジェクトがプールで作成されます。
maxTransportObjects 値に達した場合、アプリケーションは、db2.jcc.maxTransportObjectWaitTime 構成プロパティーに指定されている期間待機します。その期間が経過した後も使用可能なトランスポート・オブジェクトがプール内にない場合、プールから SQLException がスローされます。

maxTransportObjects は db2.jcc.maxTransportObjects 構成プロパティーをオーバーライド**しません**。 maxTransportObjects は、他の DataSource オブジェクトからの接続に対しては何の効果もありません。 maxTransportObjects 値のほうが db2.jcc.maxTransportObjects 値より大きくても、maxTransportObjects によって db2.jcc.maxTransportObjects 値が増加するわけではありません。

maxTransportObjects のデフォルト値は -1 です。これは、ドライバーの db2.jcc.maxTransportObjects 値によってのみ、DataSource のトランスポート・オブジェクト数が制限されることを意味します。

retrieveMessagesFromServerOnGetMessage
JDBC SQLException.getMessage または SQLWarning.getMessage 呼び出しで、エラー・メッセージ・テキストを取得する DB2 for z/OS ストアード・プロシージャーを IBM Data Server Driver for JDBC and SQLJ に呼び出させるかどうかを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは `false` です。これは、メッセージ・テキスト全体がクライアントに戻されるわけではないことを意味します。

例えば、retrieveMessagesFromServerOnGetMessage を `true` に設定すると、存在しない表 ADMF001.NO_TABLE に対する SQL 操作が試みられた後、次のようなメッセージが SQLException.getMessage によって戻されます。

```
ADMF001.NO_TABLE IS AN UNDEFINED NAME. SQLCODE=-204,
SQLSTATE=42704, DRIVER=3.50.54
```

retrieveMessagesFromServerOnGetMessage を `false` に設定すると、次のようなメッセージが戻されます。

```
DB2 SQL Error: SQLCODE=-204, SQLSTATE=42704, DRIVER=3.50.54
```

このプロパティーを `true` に設定する代わりに、アプリケーションで IBM Data Server Driver for JDBC and SQLJ 限定の `DB2Sqlca.getMessage` メソッドを使用することもできます。いずれの技法でも、ストアード・プロシージャーが呼び出されて作業単位が開始されます。

IDS および DB2 Database for Linux, UNIX, and Windows に共通の IBM Data Server Driver for JDBC and SQLJ のプロパティー

IBM Data Server Driver for JDBC and SQLJ の一部のプロパティーは、 IBM Informix Dynamic Server (IDS) および DB2 Database for Linux, UNIX, and Windows データベース・サーバーに適用されます。

IDS および DB2 Database for Linux, UNIX, and Windows に適用されるプロパティーは以下のとおりです。

currentLockTimeout
ロックを即時に取得できないときに、 DB2 Database for Linux, UNIX, and Windows サーバーがロックを待機するかどうかを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

integer 整数の*秒数* 待機します。 *integer* は、-1 から 32767 までです。

`LOCK_TIMEOUT_NO_WAIT`
ロックを待機しません。これはデフォルトです。

`LOCK_TIMEOUT_WAIT_INDEFINITELY`
無期限にロックを待機します。

`LOCK_TIMEOUT_NOT_SET`
データ・ソースにデフォルトを使用します。

queryDataSize

各フェッチ操作でデータ・ソースから戻される、照会データの量を制御するために使用されるヒントをバイト単位で指定します。この値は、データの検索に必要なデータ・ソースへの往復の数を制御することで、アプリケーションを最適化するために使用できます。

queryDataSize にさらに大きな値を使用すると、ネットワーク・トラフィックは少なくなり、結果としてパフォーマンスを向上させることができます。例えば、結果セットのサイズが 50 KB であり、queryDataSize の値が 32768 (32 KB) の場合、結果セットを検索するにはデータベース・サーバーへの 2 回の往復が必要です。しかし、queryDataSize が 61440 (60 KB) に設定されていれば、結果セットを検索するために必要なデータ・ソースへの往復は 1 回だけです。

以下の表では、各データ・ソースの queryDataSize の最小値、最大値、およびデフォルト値をリストしています。

表 46. queryDataSize の最小値、最大値、およびデフォルト値

データ・ソース	最小 queryDataSize 値	最大 queryDataSize 値	デフォルトの queryDataSize 値
DB2 Database for Linux, UNIX, and Windows	4096	65535	32767
IDS	4096	10485760	32767
DB2 for i	4096	65535	32767
DB2 for z/OS	適用外	適用外	照会データ・サイズは常に 32767 です。

DB2 Database for Linux, UNIX, and Windows の IBM Data Server Driver for JDBC and SQLJ プロパティー

IBM Data Server Driver for JDBC and SQLJ の一部のプロパティーは、 DB2 Database for Linux, UNIX, and Windows サーバーだけに適用されます。

それらのプロパティーは以下のとおりです。

connectNode

アプリケーションの接続先のターゲット・データベース・パーティション・サーバーを指定します。このプロパティーのデータ・タイプは int です。値は 0 から 999 の範囲から選択できます。デフォルトは、ポート 0 で定義されたデータベース・パーティション・サーバーです。connectNode は DB2 Database for Linux, UNIX, and Windows サーバーへの IBM Data Server Driver for JDBC and SQLJ Type 4 接続にのみ適用されます。

concurrentAccessResolution

データ・ソースが Currently Commited アクセスをサポートし、アプリケーションの分離レベルがカーソル固定 (CS) または読み取り固定 (RS) の場合に、読み取りトランザクションが、書き込みトランザクションによって非互換でロックされている行のコミット済みで一貫性のあるイメージにアクセスできることを、IBM Data Server Driver for JDBC and SQLJ が要求するかどうかを指定します。このオプションは DB2 CONCURRENTACCESSRESOLUTION バインド・オプションと同じ効果を及ぼします。可能な値は以下のとおりです。

DB2BaseDataSource.-
CONCURRENTACCESS_USE_CURRENTLY_COMMITTED (1)
IBM Data Server Driver for JDBC and SQLJ は以下の事柄を要求します。

- データの更新時または削除時に、読み取りトランザクションは currently committed データにアクセスする。
- 読み取りトランザクションは、挿入されようとしている行をスキップする。

DB2BaseDataSource.CONCURRENTACCESS_WAIT_FOR_OUTCOME (2)
IBM Data Server Driver for JDBC and SQLJ は以下の事柄を要求します。

- 更新または削除されようとしているデータが検出された場合、読み取りトランザクションはコミットまたはロールバック操作を待機する。
- 読み取りトランザクションは、挿入されようとしている行をスキップしない。

DB2BaseDataSource.CONCURRENTACCESS_NOT_SET (0)
ロック競合が発生した場合の読み取りトランザクションに関するデータ・サーバーのデフォルト動作を使用可能にします。これはデフォルト値です。

currentExplainMode
CURRENT EXPLAIN MODE 特殊レジスターの値を指定します。 CURRENT EXPLAIN MODE 特殊レジスターを使用して、EXPLAIN 機能を有効/無効にすることができます。このプロパティーのデータ・タイプは String です。最大長は 254 バイトです。このプロパティーは、 DB2 Database for Linux, UNIX, and Windows などの、CURRENT EXPLAIN MODE 特殊レジスターをサポートするデータ・ソースへの接続にのみ適用されます。

currentExplainSnapshot
CURRENT EXPLAIN SNAPSHOT 特殊レジスターの値を指定します。 CURRENT EXPLAIN SNAPSHOT 特殊レジスターを使用して、Explain スナップショット機能を有効/無効にすることができます。このプロパティーのデータ・タイプは String です。最大長は 8 バイトです。このプロパティーは、 DB2 Database for Linux, UNIX, and Windows などの、CURRENT EXPLAIN SNAPSHOT 特殊レジスターをサポートするデータ・ソースへの接続にのみ適用されます。

currentQueryOptimization
データベース・マネージャーが動的 SQL ステートメントのバインド時に実行する照会最適化のクラスを制御する値を指定します。このプロパティーのデータ・タイプは int です。 currentQueryOptimization に指定できる値は次のとおりです。

0 最小限の最適化を実行してアクセス・プランを生成することを指定します。このクラスは、的確に索引付けされた表への単純な動的 SQL アクセスの場合に最適です。

1 DB2 Database for Linux, UNIX, and Windows バージョン 1 とほぼ同等の最適化を実行してアクセス・プランを生成することを指定します。

2 最適化のレベルを DB2 Database for Linux, UNIX, and Windows バージョン 1 のレベルより高く指定しますが、特に非常に複雑な照会の場合、最適化コストはレベル 3 以上よりかなり低く指定されます。

3 中程度の最適化を実行してアクセス・プランを生成することを指定します。

5 かなりの量の最適化を実行してアクセス・プランを生成することを指定します。複雑な動的 SQL 照会の場合、アクセス・プランの選択に費やす時間を制限するためにヒューリスティック・ルールが使用されます。可能なかぎり、照会は、基礎となる基本表ではなくマテリアライズ照会表を使用します。

7 かなりの量の最適化を実行してアクセス・プランを生成することを指定します。この値は 5 に似ていますが、ヒューリスティック・ルールは使用しません。

9 最大限の最適化を実行してアクセス・プランを生成することを指定します。これによって、評価対象とすることができるアクセス・プランの数が大幅に増加します。このクラスを使用する必要があるのは、大きな表を使用する非常に複雑でしかも非常に長期間にわたって実行される照会用として、より優れたアクセス・プランを生成できるかどうかを判断する場合です。 Explain およびパフォーマンス測定を使用して、より優れたプランが生成されたことを検証することができます。

optimizationProfile
SQL の最適化中に使用される最適化プロファイルを指定します。このプロパティーのデータ・タイプは String です。 optimizationProfile 値は OPTIMIZATION PROFILE 特殊レジスターの設定に使用されます。デフォルトは NULL です。

optimizationProfile は DB2 Database for Linux, UNIX, and Windows サーバーだけに適用されます。

optimizationProfileToFlush
最適化プロファイル・キャッシュから除去される最適化プロファイルの名前を指定します。このプロパティーのデータ・タイプは String です。デフォルトは NULL です。

plugin
クライアント・サイドの JDBC セキュリティー・プラグインの名前。このプロパティーは `Object` タイプを持ち、JDBC セキュリティー・プラグイン・メソッドの新規インスタンスを収容します。

pluginName
サーバー・サイドのセキュリティー・プラグイン・モジュールの名前。

retryWithAlternativeSecurityMechanism
プロパティー securityMechanism で指定されたセキュリティー・メカニズムがデータ・ソースによってサポートされていない場合に、 IBM Data Server Driver for JDBC and SQLJ が代わりのセキュリティー・メカニズムを使用して接続を再試行するかどうかを指定します。このプロパティーのデータ・タイプは int です。可能な値は以下のとおりです。

com.ibm.db2.jcc.DB2BaseDataSource.YES (1)
代わりのセキュリティー・メカニズムを使用して接続を再試行します。

IBM Data Server Driver for JDBC and SQLJ は警告コード +4222 を出し、使用できる中で最もセキュアなセキュリティー・メカニズムを使用して接続を再試行します。

com.ibm.db2.jcc.DB2BaseDataSource.NO (2) または com.ibm.db2.jcc.DB2BaseDataSource.NOT_SET (0)
代わりのセキュリティー・メカニズムを使用して接続を再試行しません。

retryWithAlternativeSecurityMechanism は、 DB2 Database for Linux, UNIX, and Windows に対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続の接続のみに適用されます。

statementConcentrator
IBM Data Server Driver for JDBC and SQLJ で、データ・ソースのステートメント・コンセントレーター機能を使用するかどうかを指定します。ステートメント・コンセントレーターは、(リテラル値を除いて) 動的ステートメント・キャッシュ内のステートメントと同じである場合にステートメントの準備を迂回する機能です。ステートメント・コンセントレーター機能は、リテラルを含み、パラメーター・マーカーを含まない SQL ステートメントにのみ適用されます。可能な値は以下のとおりです。

DB2BaseDataSource.STATEMENT_CONCENTRATOR_OFF (1)
IBM Data Server Driver for JDBC and SQLJ はデータ・ソースのステートメント・コンセントレーター機能を使用しません。

DB2BaseDataSource.STATEMENT_CONCENTRATOR_WITH_LITERALS (2)
IBM Data Server Driver for JDBC and SQLJ はデータ・ソースのステートメント・コンセントレーター機能を使用します。

DB2BaseDataSource.STATEMENT_CONCENTRATOR_NOT_SET (0)
データ・ソースは、ステートメント・コンセントレーター機能を使用するかどうかを決定します。これはデフォルト値です。

ステートメント・コンセントレーター機能をサポートする DB2 Database for Linux, UNIX, and Windows データ・ソースの場合、STMT_CONC 構成パラメーターが ON に設定されていれば、この機能が使用されます。それ以外の場合は、ステートメント・コンセントレーター機能は使用されません。

useTransactionRedirect
パフォーマンス向上のために DB2 システムが SQL ステートメントを異なるデータベース・パーティションに送信するかどうかを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは `false` です。

このプロパティーは、以下の条件下でのみ適用されます。

- 接続先が Database Partitioning Feature (DPF) を使用する DB2 Database for Linux, UNIX, and Windows サーバーである。
- パーティション・キーがトランザクション全体で一定のままである。

useTransactionRedirect が `true` の場合、 IBM Data Server Driver for JDBC and SQLJ は接続要求を、トランザクション内の最初の送信可能ステートメントのタ

ーゲット・データを含む DPF ノードに送信します。次に DB2 Database for Linux, UNIX, and Windows は、必要に応じて SQL ステートメントを別のパーティションに送信します。

DB2 for z/OS 用の IBM Data Server Driver for JDBC and SQLJ のプロパティー

IBM Data Server Driver for JDBC and SQLJ の一部のプロパティーは、 DB2 for z/OS サーバーだけに適用されます。

それらのプロパティーは以下のとおりです。

accountingInterval

DB2 アカウント・レコードをコミット・ポイントで作成するか、データ・ソースへの物理接続の終了時に作成するかを指定します。このプロパティーのデータ・タイプは String です。

accountingInterval の値が ″COMMIT″ であり、開かれた保留カーソルがない場合、DB2 はアプリケーションが作業をコミットするたびに、アカウント・レコードを書き込みます。accountingInterval の値が ″COMMIT″ であり、保留カーソルが開いている間にアプリケーションがコミット操作を実行した場合、アカウンティング・インターバルは、そのコミット・ポイントから始まり、次の有効なアカウンティング・インターバルの終点までに及ぶものになります。
accountingInterval の値が ″COMMIT″ でない場合、アカウント・レコードはデータ・ソースへの物理接続の終了時に作成されます。

この accountingInterval プロパティーにより、基礎となる RRSAF サインオン呼び出しの *accounting-interval* パラメーターが設定されます。サブシステム・パラメーター ACCUMACC の値が NO でない場合、ACCUMACC の値は accountingInterval 設定をオーバーライドします。

accountingInterval は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。 accountingInterval は、CICS または IMS のもとでの接続や、Java ストアード・プロシージャーでの接続に対しては適用されません。

accountingInterval プロパティーは、db2.jcc.accountingInterval 構成プロパティーをオーバーライドします。

charOutputSize

`Types.CHAR` として登録される INOUT または OUT ストアード・プロシージャー・パラメーターに使用するバイトの最大数を指定します。 charOutputSize は、 DB2 for z/OSデータベース・サーバーへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続にのみ適用されます。

実行時にはストアード・プロシージャー INOUT および OUT パラメーターの DESCRIBE 情報を使用できないため、デフォルトでは、 IBM Data Server Driver for JDBC and SQLJ は INOUT または OUT 文字パラメーターの最大長をそれぞれ 32767 に設定します。ストアード・プロシージャーに多数の `Types.CHAR` パラメーターが含まれている場合、この最大値の設定によって、必要量を大きく超えるストレージが割り振られる可能性があります。

ストレージをより効率的に使用するには、それぞれの Types.CHAR INOUT または OUT パラメーターに対して、予想される最大長を charOutputSize に設定してください。

charOutputSize は、Types.VARCHAR または Types.LONGVARCHAR として登録された INOUT または OUT パラメーターには影響を与えません。 Types.VARCHAR および Types.LONGVARCHAR パラメーターの場合、ドライバーはデフォルトの長さ 32767 を使用します。

charOutputSize の値を選択するとき、文字変換中の拡張の可能性を考慮に入れる必要があります。 IBM Data Server Driver for JDBC and SQLJ は、出力パラメーター値に使用されるサーバー・サイドの CCSID に関する情報を持たないため、ドライバーはストアード・プロシージャーの出力データを UTF-8 Unicode 形式で要求します。 charOutputSize 値は、パラメーター値が UTF-8 Unicode に変換された後に必要とされる最大バイト数でなければなりません。 UTF-8 Unicode 文字は、最大で 3 バイトを必要とします。(3 バイトの UTF-8 文字の例としてユーロ記号があります。) 出力データに関する情報がない場合に charOutputSize を十分に大きな値にするには、最も長い CHAR パラメーターに定義されている長さの 3 倍に charOutputSize を設定してください。

clientUser
接続のための現行クライアントのユーザー名を指定します。この情報は、クライアントのアカウンティングを目的としています。 JDBC 接続ユーザー名とは異なり、この値は接続中に変更することもできます。 DB2 for z/OS サーバーの場合、最大長は 16 バイトです。

このプロパティーは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

clientWorkstation
接続対象の現行クライアントのワークステーション名を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。このプロパティーのデータ・タイプは String です。 DB2 for z/OS サーバーの場合、最大長は 18 バイトです。 Java 空ストリング (″″) はこの値として有効ですが、Java NULL 値は無効です。

このプロパティーは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

currentSQLID
下記のことを指定します。

- 動的に作成された CREATE、GRANT、および REVOKE SQL ステートメントに対する許可検査で使用される許可 ID。
- 動的に発行された CREATE ステートメントによって作成される、表スペース、データベース、ストレージ・グループ、またはシノニムの所有者。
- 動的 SQL ステートメントで指定される、すべての表、ビュー、別名、および索引名の暗黙修飾子。

currentSQLID は、 DB2 for z/OS サーバーの CURRENT SQLID 特殊レジスター内に値を設定します。 currentSQLID プロパティーが設定されていない場合、デフォルトのスキーマ名は、CURRENT SQLID 特殊レジスター内の値です。

jdbcCollection

実行時に IBM Data Server Driver for JDBC and SQLJ のインスタンスによって使用される、パッケージのコレクション ID を指定します。 jdbcCollection のデータ・タイプは String です。デフォルトは NULLID です。

このプロパティーは、DB2Binder -collection オプションとともに使用されます。DB2Binder ユーティリティーは、jdbcCollection 値と一致する -collection 値を使用して、サーバーで前もって IBM Data Server Driver for JDBC and SQLJ パッケージをバインドしていなければなりません。

jdbcCollection の設定によって、SQLJ アプリケーションで使用されるコレクションが判別されることはありません。 SQLJ に関しては、SQLJ カスタマイザーの -collection オプションでコレクションが判別されます。

jdbcCollection は、 DB2 for z/OS上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対しては適用されません。

maxRowsetSize

IBM Data Server Driver for JDBC and SQLJ がカーソルの複数行 FETCH を使用する際に各ステートメントの行セットのバッファリングで使用されるバイトの最大数を指定します。このプロパティーのデータ・タイプは int です。デフォルトは 32767 です。

maxRowsetSize は DB2 for z/OS上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

reportLongTypes

DatabaseMetaData メソッドが LONG VARCHAR および LONG VARGRAPHIC 列のデータ・タイプを LONG データ・タイプとしてレポートするかどうかを指定します。このプロパティーのデータ・タイプは short です。可能な値は以下のとおりです。

com.ibm.db2.jcc.DB2BaseDataSource.NO (2) または
com.ibm.db2.jcc.DB2BaseDataSource.NOT_SET (0)

LONG VARCHAR または LONG VARGRAPHIC 列に関する情報を戻す DatabaseMetaData メソッドが、結果セットの DATA_TYPE 列には java.sql.Types.VARCHAR を、TYPE_NAME 列には VARCHAR または VARGRAPHIC を戻すことを指定します。これは、 DB2 for z/OS バージョン 9 以降でのデフォルトです。

com.ibm.db2.jcc.DB2BaseDataSource.YES (1)

LONG VARCHAR または LONG VARGRAPHIC 列に関する情報を戻す DatabaseMetaData メソッドが、結果セットの DATA_TYPE 列には java.sql.Types.LONGVARCHAR を、TYPE_NAME 列には LONG VARCHAR または LONG VARGRAPHIC を戻すことを指定します。

sendCharInputsUTF8

IBM Data Server Driver for JDBC and SQLJ が文字入力データを DB2 for z/OS データベース・サーバーの CCSID に変換するか、またはデータベース・サーバーによる変換のために UTF-8 エンコードでデータを送信するかを指定します。sendCharInputsUTF8 は、 DB2 for z/OS データベース・サーバーに対する IBM Data Server Driver for JDBC and SQLJ Type 2 接続にのみ適用されます。このプロパティーのデータ・タイプは int です。このプロパティーがドライバー・

レベル (db2.jcc.sendCharInputsUTF8) でも設定されている場合、この値はドライバー・レベルの値をオーバーライドします。

可能な値は以下のとおりです。

com.ibm.db2.jcc.DB2BaseDataSource.NO (2)
データが DB2 for z/OS データベース・サーバーに送信される前に、IBM Data Server Driver for JDBC and SQLJ が文字入力データをターゲット・エンコードに変換することを指定します。
com.ibm.db2.jcc.DB2BaseDataSource.NO がデフォルトです。

com.ibm.db2.jcc.DB2BaseDataSource.YES (1)
IBM Data Server Driver for JDBC and SQLJ が文字入力データを UTF-8 エンコードで DB2 for z/OS データベース・サーバーに送信することを指定します。データベース・サーバーはデータを UTF-8 エンコードからターゲット CCSID に変換します。

com.ibm.db2.jcc.DB2BaseDataSource.YES は、SDK for Java によるターゲット CCSID への変換によって文字変換問題が発生する場合にのみ指定します。 IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用して Unicode 改行文字 (U+000A) を CCSID 37 である表列に挿入し、そのデータを z/OS 以外のクライアントから取り出すときに、最も一般的な問題が発生します。文字を列に挿入するときに SDK for Java が変換を行う場合、改行文字は EBCDIC 改行文字 X'15' に変換されます。しかし、取り出し時には、z/OS 以外のオペレーティング・システム上の一部の SDK for Java は X'15' 文字を、改行文字 (U+000A) ではなく Unicode 改行文字 (U+0085) に変換します。改行文字により、一部の XML パーサーが予期しない動作を起こします。
sendCharInputsUTF8 を com.ibm.db2.jcc.DB2BaseDataSource.YES に設定する場合、 DB2 for z/OS データベース・サーバーは列への挿入時に U+000A 文字を EBCDIC 改行文字 X'25' に変換します。そのため、文字は常に改行文字として取り出されます。

データベース・サーバー上でデータをターゲット CCSID に変換すると、 IBM Data Server Driver for JDBC and SQLJ が、ドライバーによる変換の場合よりも多くのメモリーを使用することがあります。ドライバーは文字データの変換用のメモリーを、ソース・エンコードからデータベース・サーバーに送信されるデータのエンコードに割り振ります。ドライバーが表列に送信される文字データに割り振るスペースの量は、データの最大長に基づいています。 UTF-8 データは各文字に最大で 3 バイトを必要とします。そのため、ドライバーが UTF-8 データをデータベース・サーバーに送信する場合、ドライバーは入力データ内の最大文字数の 3 倍を割り振る必要があります。ドライバーが変換を行う場合、ターゲット CCSID が 1 バイト CCSID であれば、ドライバーは入力データ内の最大文字数のみ割り振る必要があります。

sqljEnableClassLoaderSpecificProfiles
IBM Data Server Driver for JDBC and SQLJ が、複数の J2EE アプリケーション (.ear) ファイルで同じ Java 名を持つ SQLJ プロファイルの使用およびロードを許可するかどうかを指定します。このプロパティーのデータ・タイプは boolean です。デフォルトは `false` です。 sqljEnableClassLoaderSpecificProfiles

は DataSource プロパティーです。このプロパティーは主として WebSphere Application Server での使用を意図しています。

ssid

DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用して接続が確立される先の、ローカル DB2 for z/OS サブシステムの名前を指定します。このプロパティーのデータ・タイプは String です。

ssid プロパティーは、db2.jcc.ssid 構成プロパティーをオーバーライドします。

ssid は、ローカル・サブシステムのサブシステム名またはグループ接続名にすることができます。

単一のローカル・サブシステム名を指定すると、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用する接続の場合に、単一の LPAR 上の複数のサブシステムにローカル・サブシステムとしてアクセスすることができます。

グループ接続名を指定すると、データ共用グループ・メンバーに障害が起きた場合にフェイルオーバー処理を行えます。アプリケーションの接続先の DB2 サブシステムに障害が起こると、接続は終了します。ただし、新しい接続がそのグループ接続名を使用するときに、 DB2 for z/OS はグループ接続処理を使用して、接続先になるアクティブな DB2 サブシステムを見つけます。

ssid は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。

useRowsetCursor

データ・ソースが複数行フェッチをサポートする場合、 IBM Data Server Driver for JDBC and SQLJ が常に両方向スクロール・カーソルに複数行 FETCH を使用するかどうかを指定します。このプロパティーのデータ・タイプは boolean です。

このプロパティーは IBM Data Server Driver for JDBC and SQLJ Type 4 接続、または DB2 for z/OS への IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。 enableRowsetSupport プロパティーが設定されていない場合、useRowsetCursor のデフォルトは `true` です。enableRowsetSupport プロパティーが設定されている場合、useRowsetCursor プロパティーは使用されません。

位置指定の更新または削除操作を実行するために JDBC 1 技法を使用するアプリケーションは、useRowSetCursor を `false` に設定する必要があります。これらのアプリケーションは、 IBM Data Server Driver for JDBC and SQLJ が複数行 FETCH を使用している場合には正しく作動しません。

IDS の IBM Data Server Driver for JDBC and SQLJプロパティー

IBM Data Server Driver for JDBC and SQLJ の一部のプロパティーは、 IBM Informix Dynamic Server (IDS) データベースにのみ適用されます。これらのプロパティーは IDS 環境変数に相当します。

以下の情報で大文字で示されているプロパティーは、大文字で指定する必要があります。そのようなプロパティーの getXXX および setXXX メソッドは、get または set が先頭に付く大文字のプロパティー名によって形成されます。以下に例を示します。

```
boolean dbDate = DB2BaseDateSource.getDBDATE();
```

IDS 固有のプロパティーは以下のとおりです。

DBANSIWARN

SQL ステートメントで ANSI 規格の構文が使用されていない場合、 IBM Data Server Driver for JDBC and SQLJ が IDS データベースに、SQLWarning をアプリケーションに戻すよう指示するかどうかを指定します。このプロパティーのデータ・タイプは boolean です。可能な値は以下のとおりです。

false または 0

SQL ステートメントで ANSI 規格の構文が使用されていない場合、SQLWarning をアプリケーションに戻すように IDS データベースに指示する値をこのデータベースに送信しません。これはデフォルトです。

true または 1

SQL ステートメントで ANSI 規格の構文が使用されていない場合、SQLWarning をアプリケーションに戻すように IDS データベースに指示する値をこのデータベースに送信します。

IBM Data Server Driver for JDBC and SQLJ プロパティー DBANSIWARN を使用して DBANSIWARN IDS プロパティーを設定することはできますが、IBM Data Server Driver for JDBC and SQLJ プロパティー DBANSIWARN を使用して DBANSIWARN IDS プロパティーをリセットすることはできません。

DBDATE

DATE 値のエンド・ユーザー形式を指定します。このプロパティーのデータ・タイプは String です。指定できる値については、「*IBM Informix Guide to SQL: Reference*」の DBDATE 環境変数の説明を参照してください。

デフォルト値は ″Y4MD-″ です。

DBPATH

データベースを含んでいるデータベース・サーバーを識別する値のコロン区切りリストを指定します。このプロパティーのデータ・タイプは String です。各値は以下のようにすることができます。

- 絶対パス名
- 相対パス名
- IDS データベース・サーバーのサーバー名
- サーバー名および絶対パス名

デフォルトは ″.″ です。

DBSPACETEMP

一時表が置かれている既存の dbspace のコンマ区切りリストまたはコロン区切りリストを指定します。このプロパティーのデータ・タイプは String です。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 DBSPACETEMP 環境変数の値が使用されます。

DBTEMP
一時ファイルおよび一時表が置かれている既存のディレクトリーの絶対パス名を指定します。このプロパティーのデータ・タイプは String です。デフォルトは ″/tmp″ です。

DBUPSPACE
UPDATE STATISTICS ステートメントが複数の列分布を同時に構成するときに使用できるシステム・ディスク・スペースの最大量およびメモリーの最大量を K バイト (KB) 単位で指定します。このプロパティーのデータ・タイプは String です。

DBUPSPACE の形式は *″maximum-disk-space:maximum-memory″* です。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 DBUPSPACE 環境変数の値が使用されます。

DB_LOCALE
ロケール依存データの処理にデータベース・サーバーが使用するデータベース・ロケールを指定します。このプロパティーのデータ・タイプは String です。有効な値は、DB_LOCALE 環境変数の有効な値と同じです。デフォルト値は NULL です。

DELIMIDENT
区切り付き SQL ID をアプリケーションで使用できるかどうかを指定します。このプロパティーのデータ・タイプは boolean です。可能な値は以下のとおりです。

false アプリケーションに区切り付き SQL ID を含めることはできません。二重引用符 (″) または単一引用符 (') でリテラル・ストリングを区切ります。これはデフォルトです。

true アプリケーションに区切り付き SQL ID を含めることができます。区切り付き SQL ID は二重引用符 (″) で囲む必要があります。単一引用符 (') でリテラル・ストリングを区切ります。

IFX_DIRECTIVES
オプティマイザーが照会内からの照会最適化ディレクティブを許可するかどうかを指定します。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″1″ または ″ON″
最適化ディレクティブは受け入れられます。

″0″ または ″OFF″
最適化ディレクティブは受け入れられません。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 IFX_DIRECTIVES 環境変数の値が使用されます。

IFX_EXTDIRECTIVES
オプティマイザーが、sysdirectives システム・カタログ表からの外部照会最適化ディレクティブを、既存のアプリケーションでの照会に適用できるかどうかを指定します。可能な値は以下のとおりです。

″1″ または ″ON″
外部照会最適化ディレクティブは受け入れられます。

″**0**″ または ″**OFF**″
外部照会最適化は受け入れられません。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 IFX_EXTDIRECTIVES 環境変数の値が使用されます。

IFX_UPDDESC
UPDATE ステートメントの DESCRIBE が許可されるかどうかを指定します。このプロパティーのデータ・タイプは String です。

NULL 以外の値は、UPDATE ステートメントの DESCRIBE が許可されることを示します。デフォルトは ″1″ です。

IFX_XASTDCOMPLIANCE_XAEND
グローバル・トランザクションが明示的ロールバックの後にのみ解放されるか、またはどのロールバックの後にでも解放されるかを指定します。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″**0**″ グローバル・トランザクションは明示的ロールバックの後にのみ解放されます。この動作は X/Open XA 標準に準拠します。

″**1**″ グローバル・トランザクションはいずれかのロールバックの後に解放されます。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 IFX_XASTDCOMPLIANCE_XAEND 環境変数の値が使用されます。

INFORMIXOPCACHE
クライアント・アプリケーションのステージング・エリア BLOB 領域に対してメモリー・キャッシュのサイズを K バイト (KB) 単位で指定します。このプロパティーのデータ・タイプは String です。値 ″0″ は、キャッシュが使用されないことを示します。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 INFORMIXOPCACHE 環境変数の値が使用されます。

INFORMIXSTACKSIZE
データベース・サーバーがクライアント・セッションの 1 次スレッドに使用するスタック・サイズを K バイト (KB) 単位で指定します。このプロパティーのデータ・タイプは String です。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 INFORMIXSTACKSIZE 環境変数の値が使用されます。

NODEFDAC
ANSI 準拠でないデータベースで、新規表が現行セッション中に作成されるときに、データベース・サーバーがデフォルトの表特権
(SELECT、INSERT、UPDATE、および DELETE) が PUBLIC に付与されないようにするかどうかを指定します。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″**yes**″ ANSI 準拠でないデータベースで、新規表が現行セッション中に作成されるときに、データベース・サーバーはデフォルトの表特権が PUBLIC に付与されないようにします。

″**no**″ ANSI 準拠でないデータベースで、新規表が現行セッション中に作成さ

れるときに、データベース・サーバーはデフォルトの表特権が PUBLIC に付与されないようにしません。これはデフォルトです。

OPTCOMPIND
順序対の表に対して結合操作を実行するための優先メソッドを指定します。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″**0**″ オプティマイザーは、可能であればソート・マージ結合またはハッシュ結合よりもネスト・ループ結合を選択します。

″**1**″ 分離レベルが反復可能読み取りである場合、オプティマイザーは可能であればソート・マージ結合またはハッシュ結合よりもネスト・ループ結合を選択します。分離レベルが反復可能読み取りでない場合、オプティマイザーはコストに基づいて結合方法を選択します。

″**2**″ トランザクション分離モードに関係なく、オプティマイザーはコストに基づいて結合方法を選択します。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 OPTCOMPIND 環境変数の値が使用されます。

OPTOFC
optimize-OPEN-FETCH-CLOSE 機能を使用可能にするかどうかを指定します。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″**0**″ アプリケーションのすべてのスレッドについて optimize-OPEN-FETCH-CLOSE 機能を使用不可にします。

″**1**″ アプリケーションのすべてのスレッド内のすべてのカーソルについて optimize-OPEN-FETCH-CLOSE 機能を使用可能にします。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 OPTOFCD 環境変数の値が使用されます。

PDQPRIORITY
データベース・サーバーが使用する並列処理の度合いを指定します。
PDQPRIORITY 値は、データベース・サーバーがリソース (メモリー、プロセッサー、およびディスク読み取りを含む) を割り振る方法に影響を与えます。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″**HIGH**″
データベース・サーバーがリソースをすべてのユーザー間で割り振る場合、できる限り多くのリソースが照会に割り当てられます。

″**LOW**″ **または** ″**1**″
データベース・サーバーはフラグメント化表から値を並列でフェッチします。

″**OFF**″ **または** ″**0**″
並列処理は使用不可にされています。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 PDQPRIORITY 環境変数の値が使用されます。

PSORT_DBTEMP
データベース・サーバーがソート操作に使用される一時ファイルを作成するディレクトリーの絶対パス名を指定します。このプロパティーのデータ・タイプは String です。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 PSORT_DBTEMP 環境変数の値が使用されます。

PSORT_NPROCS
データベース・サーバーが照会をソートするために使用できるスレッドの最大数を指定します。このプロパティーのデータ・タイプは String です。 PSORT_NPROCS の最大値は ″10″ です。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 PSORT_NPROCS 環境変数の値が使用されます。

STMT_CACHE
shared-statement キャッシュを使用可能にするかどうかを指定します。このプロパティーのデータ・タイプは String です。可能な値は以下のとおりです。

″**0**″ shared-statement キャッシュは使用不可にされています。

″**1**″ 512 KB の shared-statement キャッシュが使用可能にされます。

このプロパティーが設定されていない場合、サーバーに送信される値はありません。 STMT_CACHE 環境変数の値が使用されます。

dumpPool
サマリー統計に加えて、書き込まれるグローバル・トランスポート・プール・イベントの統計のタイプを指定します。グローバル・トランスポート・プールは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

dumpPool のデータ・タイプは int. dumpPoolStatisticsOnSchedule です。どの統計が書き込まれるよりも前に、 dumpPoolStatisticsOnScheduleFile も統計の書き込み用に設定する必要があります。

db2.jcc.dumpPool プロパティーには、以下のタイプの統計を 1 つ以上指定することができます。

- DUMP_REMOVE_OBJECT (16 進: X'01'、10 進: 1)
- DUMP_GET_OBJECT (16 進: X'02'、10 進: 2)
- DUMP_WAIT_OBJECT (16 進: X'04'、10 進: 4)
- DUMP_SET_AVAILABLE_OBJECT (16 進: X'08'、10 進: 8)
- DUMP_CREATE_OBJECT (16 進: X'10'、10 進: 16)
- DUMP_SYSPLEX_MSG (16 進: X'20'、10 進: 32)
- DUMP_POOL_ERROR (16 進: X'80'、10 進: 128)

複数のイベントのタイプをトレースするには、トレースしたいイベントのタイプの値を加算します。例えば、DUMP_GET_OBJECT イベントと DUMP_CREATE_OBJECT イベントをトレースするとします。これらの値に相当する数値は 2 と 16 なので、dumpPool 値には 18 を指定します。

デフォルトは 0 です。これは、グローバル・トランスポート・プールのサマリー統計だけが書き込まれることを意味します。

このプロパティーには setXXX または getXXX メソッドがありません。

dumpPoolStatisticsOnSchedule
グローバル・トランスポート・プールの統計が、dumpPoolStatisticsOnScheduleFile で指定されたファイルに書き込まれる頻度を秒単位で指定します。グローバル・トランスポート・オブジェクト・プールは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

デフォルトは -1 です。 -1 は、グローバル・トランスポート・プールの統計が書き込まれないことを意味します。

このプロパティーには setXXX または getXXX メソッドがありません。

dumpPoolStatisticsOnScheduleFile
グローバル・トランスポート・プールの統計を書き込むファイルの名前を指定します。グローバル・トランスポート・プールは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

dumpPoolStatisticsOnScheduleFile を指定しないと、グローバル・トランスポート・プールの統計は書き込まれません。

このプロパティーには setXXX または getXXX メソッドがありません。

maxTransportObjectIdleTime
未使用のトランスポート・オブジェクトが、グローバル・トランスポート・オブジェクト・プール内に置かれてからこのプールから削除されるまでの期間を秒数で指定します。トランスポート・オブジェクトは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

maxTransportObjectIdleTime のデフォルト値は 60 です。
maxTransportObjectIdleTime を 0 より小さい値に設定すると、未使用のトランスポート・オブジェクトは即時にプールから削除されます。このアクションは、重大な性能低下の原因になる可能性があるので、お勧め**しません**。

このプロパティーには setXXX または getXXX メソッドがありません。

maxTransportObjectWaitTime
maxTransportObjects 値に達した場合に、アプリケーションがトランスポート・オブジェクトを待機する最大の時間を秒数で指定します。トランスポート・オブジェクトは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。 アプリケーションが、maxTransportObjectWaitTime 値よりも長い期間待機していると、グローバル・トランスポート・オブジェクト・プールは SQLException をスローします。

maxTransportObjectWaitTime のデフォルト値は -1 です。負の値は、アプリケーションはいつまでも待機することを意味します。

このプロパティーには setXXX または getXXX メソッドがありません。

minTransportObjects
接続コンセントレーターと Sysplex ワークロード・バランシング用のグローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクト数の下限を指定します。 JVM の作成時には、プール内にはトランスポート・オブジェクトはありません。トランスポート・オブジェクトは、必要に応じてプールに追加されます。minTransportObjects 値に達した後は、JVM が存続するかぎり、グローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクトの数が minTransportObjects 値より小さくなることはありません。

minTransportObjects のデフォルト値は 0 です。0 以下のすべての値は、グローバル・トランスポート・オブジェクト・プールは空になってもかまわないことを意味します。

このプロパティーには setXXX または getXXX メソッドがありません。

IBM Data Server Driver for JDBC and SQLJ 構成プロパティー

IBM Data Server Driver for JDBC and SQLJ 構成プロパティーにはドライバー全体にわたる有効範囲があります。

以下の表は、構成プロパティーおよび対応する Connection または DataSource プロパティー (存在する場合) を要約しています。

表 47. 構成プロパティーおよび対応する Connection および DataSource プロパティーの要約

構成プロパティー名	Connection または DataSource プロパティー名: com.ibm.db2.jcc.DB2BaseDataSource。 ...	注
db2.jcc.accountingInterval	accountingInterval	1 (339 ページ), 4 (339 ページ)
db2.jcc.allowSqljDuplicateStaticQueries		4 (339 ページ)
db2.jcc.charOutputSize	charOutputSize	1 (339 ページ), 4 (339 ページ)
db2.jcc.currentSchema	currentSchema	1 (339 ページ), 4 (339 ページ), 6 (339 ページ)
db2.jcc.override.currentSchema	currentSchema	2 (339 ページ), 4 (339 ページ), 6 (339 ページ)
db2.jcc.currentSQLID	currentSQLID	1 (339 ページ), 4 (339 ページ)
db2.jcc.override.currentSQLID	currentSQLID	2 (339 ページ), 4 (339 ページ)
db2.jcc.decimalRoundingMode	decimalRoundingMode	1 (339 ページ), 4 (339 ページ), 6 (339 ページ)
db2.jcc.override.decimalRoundingMode	decimalRoundingMode	2 (339 ページ), 4 (339 ページ), 6 (339 ページ)
db2.jcc.defaultSQLState		4 (339 ページ)
db2.jcc.disableSQLJProfileCaching		4 (339 ページ)
db2.jcc.dumpPool	dumpPool	1 (339 ページ), 3 (339 ページ), 4 (339 ページ), 5 (339 ページ)
db2.jcc.dumpPoolStatisticsOnSchedule	dumpPoolStatisticsOnSchedule	1 (339 ページ), 3 (339 ページ), 4 (339 ページ), 5 (339 ページ)
db2.jcc.dumpPoolStatisticsOnScheduleFile	dumpPoolStatisticsOnScheduleFile	1 (339 ページ), 3 (339 ページ), 4 (339 ページ), 5 (339 ページ)
db2.jcc.jmxEnabled		4 (339 ページ), 5 (339 ページ), 6 (339 ページ)
db2.jcc.lobOutputSize		4 (339 ページ)
db2.jcc.maxTransportObjectIdleTime	maxTransportObjectIdleTime	1 (339 ページ), 4 (339 ページ), 5 (339 ページ)
db2.jcc.maxTransportObjectWaitTime	maxTransportObjectWaitTime	1 (339 ページ), 4 (339 ページ), 5 (339 ページ)

表 47. 構成プロパティーおよび対応する *Connection* および *DataSource* プロパティーの要約 (続き)

構成プロパティー名	Connection または DataSource プロパティー名: com.ibm.db2.jcc.DB2BaseDataSource。 ...	注
db2.jcc.maxTransportObjects	maxTransportObjects	1, 4, 5
db2.jcc.minTransportObjects	minTransportObjects	1, 4, 5
db2.jcc.pkList	pkList	1, 4
db2.jcc.planName	planName	1, 4
db2.jcc.progressiveStreaming	progressiveStreaming	1, 4, 5, 6
db2.jcc.override.progressiveStreaming	progressiveStreaming	2, 4, 5, 6
db2.jcc.rollbackOnShutdown		4
db2.jcc.sendCharInputsUTF8	sendCharInputsUTF8	4
db2.jcc.sqljUncustomizedWarningOrException		4, 6
db2.jcc.ssid	ssid	1, 4
db2.jcc.traceDirectory	traceDirectory	1, 4, 5, 6
db2.jcc.override.traceDirectory	traceDirectory	2, 4, 5, 6
db2.jcc.traceFile	traceFile	1, 4, 5, 6
db2.jcc.override.traceFile	traceFile	2, 4, 5, 6
db2.jcc.traceFileAppend	traceFileAppend	1, 4, 5, 6
db2.jcc.override.traceFileAppend	traceFileAppend	2, 4, 5, 6
db2.jcc.traceLevel	traceLevel	1, 4, 5, 6
db2.jcc.override.traceLevel	traceLevel	2, 4, 5, 6
db2.jcc.tracePolling		4, 5, 6
db2.jcc.tracePollingInterval		4, 5, 6
db2.jcc.t2zosTraceFile		4
db2.jcc.t2zosTraceBufferSize		4
db2.jcc.t2zosTraceWrap		4
db2.jcc.useCcsid420ShapedConverter		4

注:

1. Connection または DataSource プロパティーの設定は、構成プロパティーの設定をオーバーライドします。構成プロパティーは、Connection または DataSource プロパティーのデフォルト値を提供します。
2. 構成プロパティーの設定は、Connection または DataSource プロパティーをオーバーライドします。
3. 対応する Connection または DataSource プロパティーは、 IBM Informix Dynamic Server に対してのみ定義されます。
4. 構成プロパティーは DB2 for z/OS に対して適用されます。
5. 構成プロパティーは IBM Informix Dynamic Server に対して適用されます。
6. 構成プロパティーは DB2 Database for Linux, UNIX, and Windows に対して適用されます。

構成プロパティーの意味は、以下のとおりです。

db2.jcc.accountingInterval

DB2 アカウント・レコードをコミット・ポイントで作成するか、データ・ソースへの物理接続の終了時に作成するかを指定します。 db2.jcc.accountingInterval の値が COMMIT である場合、 DB2 アカウント・レコードはコミット・ポイントで作成されます。例:

```
db2.jcc.accountingInterval=COMMIT
```

それ以外の場合、アカウント・レコードはデータ・ソースへの物理接続の終了時に作成されます。

db2.jcc.accountingInterval は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 に対してのみ適用されます。
db2.jcc.accountingInterval は、CICS または IMS のもとでの接続や、Java ストアード・プロシージャーでの接続に対しては適用されません。

Connection または DataSource オブジェクトの accountingInterval プロパティーを設定することによって、db2.jcc.accountingInterval をオーバーライドすることができます。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.allowSqljDuplicateStaticQueries
SQLJ アプリケーションの単一 SELECT ステートメントの複数のオープン・イテレーターが IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の下で許可されるかどうかを指定します。

このサポートを使用可能にするには、db2.jcc.allowSqljDuplicateStaticQueries を YES または true に設定します。

db2.jcc.charOutputSize
Types.CHAR として登録されているストアード・プロシージャーの INOUT または OUT パラメーターに使用する最大バイト数を指定します。

実行時にはストアード・プロシージャー INOUT および OUT パラメーターの DESCRIBE 情報を使用できないため、デフォルトでは、 IBM Data Server Driver for JDBC and SQLJ は INOUT または OUT 文字パラメーターの最大長をそれぞれ 32767 に設定します。ストアード・プロシージャーに多数の Types.CHAR パラメーターが含まれている場合、この最大値の設定によって、必要量を大きく超えるストレージが割り振られる可能性があります。

ストレージをより効率的に使用するには、どの Types.CHAR INOUT または OUT パラメーターについても、db2.jcc.charOutputSize を予期される最大長に設定してください。

db2.jcc.charOutputSize は、Types.VARCHAR または Types.LONGVARCHAR として登録されている INOUT または OUT パラメーターには影響しません。
Types.VARCHAR および Types.LONGVARCHAR パラメーターの場合、ドライバーはデフォルトの長さ 32767 を使用します。

db2.jcc.charOutputSize 用に選択する値は、文字変換時に拡張される可能性があるということを考慮に入れる必要があります。 IBM Data Server Driver for JDBC and SQLJ は、出力パラメーター値に使用されるサーバー・サイドの CCSID に関する情報を持たないため、ドライバーはストアード・プロシージャーの出力データを UTF-8 Unicode 形式で要求します。 db2.jcc.charOutputSize の値は、パラメーター値を UTF-8 Unicode に変換した後に必要になる最大バイト数でなければなりません。 UTF-8 Unicode 文字は、最大で 3 バイトを必要とします。(3 バイトの UTF-8 文字の例としてユーロ記号があります。) 出力データに関する情報がない場合に db2.jcc.charOutputSize の値を十分な大きさにするには、db2.jcc.charOutputSize を、最大 CHAR パラメーターに定義されている 3 倍の長さに設定してください。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.currentSchema または db2.jcc.override.currentSchema
動的に準備される SQL ステートメントで非修飾データベース・オブジェクトを

修飾するために使用されるデフォルトのスキーマ名を指定します。このプロパティーのこの値は、データベース・サーバーでの CURRENT SCHEMA 特殊レジスター内の値を設定します。スキーマ名は大文字小文字の区別があり、大文字で指定する必要があります。

この構成プロパティーは、 DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows にのみ適用されます。

db2.jcc.currentSQLID または db2.jcc.override.currentSQLID
下記のことを指定します。

- 動的に作成された CREATE、GRANT、および REVOKE SQL ステートメントに対する許可検査で使用される許可 ID。
- 動的に発行された CREATE ステートメントによって作成される、表スペース、データベース、ストレージ・グループ、またはシノニムの所有者。
- 動的 SQL ステートメントで指定される、すべての表、ビュー、別名、および索引名の暗黙修飾子。

currentSQLID は、 DB2 for z/OS サーバーの CURRENT SQLID 特殊レジスター内に値を設定します。 currentSQLID プロパティーが設定されていない場合、デフォルトのスキーマ名は、CURRENT SQLID 特殊レジスター内の値です。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.decimalRoundingMode または db2.jcc.override.decimalRoundingMode
DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows データベース・サーバー上での 10 進値または 10 進浮動小数点値の丸めモード、および 10 進データ・タイプをサポートするその他のすべてのデータ・ソース上での 10 進値の丸めモードを指定します。

可能な値は以下のとおりです。

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_DOWN (1)
0 の方向に丸めます (切り捨て)。破棄された数字は無視されます。

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_CEILING (2)
正の無限大方向に丸めます。破棄される数字がすべてゼロであるか、または符号が負の場合、破棄される数字の除去を除いて結果は未変更のままになります。それ以外の場合、結果の係数は 1 だけ増やされます。

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_HALF_EVEN (3)
値を最も近い値に丸めます。複数の値が等距離にある場合は、最後の数字が偶数になるように値を丸めます。破棄される数字が、左側の次の桁内の 1 の半分 (0.5) より大きい場合、結果の係数は 1 だけ増やされます。半分より小さい場合、結果の係数は調整されません (つまり、廃棄された数字は無視されます)。それ以外の場合、結果の係数の右端の数字が偶数であれば、その係数は変更されませんが、右端の数字が奇数であれば、その桁が偶数になるように 1 だけ増大 (切り上げ) します。

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_HALF_UP (4)
値を最も近い値に丸めます。複数の値が等距離にある場合は、0 の反対方向に値を丸めます。破棄される数字が、左側の次の桁内の 1 の半分 (0.5) 以上の場合、結果の係数は 1 だけ増やされます。そうでない場合、破棄された数字は無視されます。

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_FLOOR (6)
負の無限大方向に丸めます。破棄される数字がすべてゼロであるか、または符号が正の場合、破棄される数字の除去を除いて、結果は未変更のままになります。それ以外の場合、符号は負で、結果の係数は 1 だけ増やされます。

com.ibm.db2.jcc.DB2BaseDataSource.ROUND_UNSET (-2147483647)
丸めモードは明示的に設定されませんでした。 IBM Data Server Driver for JDBC and SQLJ は、decimalRoundingMode を使用してデータ・ソース上で丸めモードを設定することはしません。

IBM Data Server Driver for JDBC and SQLJ はその丸めモードに以下の値を使用します。

- データ・ソースが DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows である場合、丸めモードは 10 進値または 10 進浮動小数点値については ROUND_HALF_EVEN です。
- その他のデータ・ソースの場合、丸めモードは 10 進値については ROUND_DOWN です。

この構成プロパティーは、 DB2 for z/OS バージョン 9 以降、または DB2 Database for Linux, UNIX, and Windows バージョン 9.1 以降にのみ適用されます。

db2.jcc.defaultSQLState
SQLException または SQLWarning オブジェクトの SQLSTATE 値がヌルの場合に、 IBM Data Server Driver for JDBC and SQLJがクライアントに返す SQLSTATE 値を指定します。この構成プロパティーは以下の方法で指定できます。

db2.jcc.defaultSQLState
db2.jcc.defaultSQLState を値なしで指定すると、 IBM Data Server Driver for JDBC and SQLJ は 'FFFFF' を返します。

db2.jcc.defaultSQLState=*xxxxx*
xxxxx は、SQLSTATE 値がヌルの場合に IBM Data Server Driver for JDBC and SQLJ が返す値です。*xxxxx* が 5 バイトより長い場合、ドライバーは値を切り捨てて、5 バイトにします。*xxxxx* が 5 バイトより短い場合、ドライバーは *xxxxx* の右方にブランクを埋め込みます。

db2.jcc.defaultSQLState を指定しない場合、 IBM Data Server Driver for JDBC and SQLJ はヌルの SQLSTATE 値を返します。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.disableSQLJProfileCaching
アプリケーションを実行している JVM がリセットされるときに、直列化されたプロファイルがキャッシュされるかどうかを指定します。
db2.jcc.disableSQLJProfileCaching はリセット可能な JVM で実行されているアプリケーション (CICS、IMS、または Java ストアード・プロシージャー環境で実行されているアプリケーション) にのみ適用され、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 を使用します。可能な値は以下のとおりです。

YES JVM がリセットされるときに、直列化された SQLJ プロファイルの新

規バージョンがロードされるので、その直列化された SQLJ プロファイルは、JVM がリセットされる度にキャッシュされる訳ではありません。アプリケーションが開発中のときにこのオプションを使用すると、アプリケーションおよび直列化されたプロファイルの新規バージョンが頻繁に作成されます。

NO JVM がリセットされるときに、直列化された SQLJ プロファイルをキャッシュします。 NO がデフォルトです。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.dumpPool

サマリー統計に加えて、書き込まれるグローバル・トランスポート・プール・イベントの統計のタイプを指定します。グローバル・トランスポート・プールは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

どの統計が書き込まれるよりも前に、db2.jcc.dumpPoolStatisticsOnSchedule および db2.jcc.dumpPoolStatisticsOnScheduleFile も統計の書き込み用に設定する必要があります。

db2.jcc.dumpPool プロパティーには、以下のタイプの統計を 1 つ以上指定することができます。

- DUMP_REMOVE_OBJECT (16 進: X'01'、10 進: 1)
- DUMP_GET_OBJECT (16 進: X'02'、10 進: 2)
- DUMP_WAIT_OBJECT (16 進: X'04'、10 進: 4)
- DUMP_SET_AVAILABLE_OBJECT (16 進: X'08'、10 進: 8)
- DUMP_CREATE_OBJECT (16 進: X'10'、10 進: 16)
- DUMP_SYSPLEX_MSG (16 進: X'20'、10 進: 32)
- DUMP_POOL_ERROR (16 進: X'80'、10 進: 128)

複数のイベントのタイプをトレースするには、トレースしたいイベントのタイプの値を加算します。例えば、DUMP_GET_OBJECT イベントと DUMP_CREATE_OBJECT イベントをトレースするとします。これらの値に相当する数値は 2 と 16 なので、db2.jcc.dumpPool 値には 18 を指定します。

デフォルトは 0 です。これは、グローバル・トランスポート・プールのサマリー統計だけが書き込まれることを意味します。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.dumpPoolStatisticsOnSchedule

グローバル・トランスポート・プールの統計が、db2.jcc.dumpPoolStatisticsOnScheduleFile で指定されたファイルに書き込まれる頻度を秒単位で指定します。グローバル・トランスポート・オブジェクト・プールは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

デフォルトは -1 です。 -1 は、グローバル・トランスポート・プールの統計が書き込まれないことを意味します。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.dumpPoolStatisticsOnScheduleFile
グローバル・トランスポート・プールの統計を書き込むファイルの名前を指定します。グローバル・トランスポート・プールは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

db2.jcc.dumpPoolStatisticsOnScheduleFile 構成プロパティーを指定しないと、グローバル・トランスポート・プールの統計は書き込まれません。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.jmxEnabled
Java Management Extensions (JMX) が IBM Data Server Driver for JDBC and SQLJ インスタンスについて使用可能にされるかどうかを指定します。アプリケーションがリモート・トレース・コントローラーを使用できるようにするには、その前に JMX を使用可能にする必要があります。

可能な値は以下のとおりです。

true または yes
JMX が使用可能にされることを示します。

その他の値
JMX が使用不可にされることを示します。これはデフォルトです。

db2.jcc.lobOutputSize
ドライバーが出力 LOB のサイズを判別できないときに、 IBM Data Server Driver for JDBC and SQLJ が出力 LOB 値に割り振る必要のあるストレージのバイト数を指定します。この状態は、LOB ストアード・プロシージャーの出力パラメーターに関して発生します。db2.jcc.lobOutputSize は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 に対してのみ適用されます。

db2.jcc.lobOutputSize のデフォルト値は 1048576 です。ストレージの制限があるシステムおよび LOB がより小さいシステムの場合、db2.jcc.lobOutputSize 値をより低い数値に設定します。

例えば、出力 LOB サイズが最大で 64000 であることがわかっている場合、db2.jcc.lobOutputSize は 64000 に設定します。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.maxTransportObjectIdleTime
未使用のトランスポート・オブジェクトが、グローバル・トランスポート・オブジェクト・プール内に置かれてからこのプールから削除されるまでの期間を秒数で指定します。トランスポート・オブジェクトは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。

db2.jcc.maxTransportObjectIdleTime のデフォルト値は 60 です。
db2.jcc.maxTransportObjectIdleTime を 0 より小さい値に設定すると、未使用のトランスポート・オブジェクトは即時にプールから削除されます。このアクションは、重大な性能低下の原因になる可能性があるので、お勧め**しません**。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.maxTransportObjects
接続コンセントレーターと Sysplex ワークロード・バランシング用のグローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクト数の上限を指定します。プール内のトランスポート・オブジェクトの数が db2.jcc.maxTransportObjects 値に達すると、db2.jcc.maxTransportObjectIdleTime 値を超えた期間使用されなかったトランスポート・オブジェクトは、プールから削除されます。

db2.jcc.maxTransportObjects のデフォルト値は -1 です。0 以下の値は、グローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクト数には制限がないことを意味します。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.maxTransportObjectWaitTime
db2.jcc.maxTransportObjects 値に達した場合に、アプリケーションがトランスポート・オブジェクトを待機する最大の時間を秒数で指定します。トランスポート・オブジェクトは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されます。 アプリケーションが、db2.jcc.maxTransportObjectWaitTime 値よりも長い期間待機していると、グローバル・トランスポート・オブジェクト・プールは SQLException をスローします。

db2.jcc.maxTransportObjectWaitTime のデフォルト値は -1 です。負の値は、アプリケーションはいつまでも待機することを意味します。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.minTransportObjects
接続コンセントレーターと Sysplex ワークロード・バランシング用のグローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクト数の下限を指定します。 JVM の作成時には、プール内にはトランスポート・オブジェクトはありません。トランスポート・オブジェクトは、必要に応じてプールに追加されます。db2.jcc.minTransportObjects 値に達した後は、JVM が存続するかぎり、グローバル・トランスポート・オブジェクト・プール内のトランスポート・オブジェクトの数が db2.jcc.minTransportObjects 値より小さくなることはありません。

db2.jcc.minTransportObjects のデフォルト値は 0 です。0 以下のすべての値は、グローバル・トランスポート・オブジェクト・プールは空になってもかまわないことを意味します。

この構成プロパティーは、 DB2 for z/OS または IBM Informix Dynamic Server にのみ適用されます。

db2.jcc.pkList
データ・ソースへの JDBC または SQLJ 接続が確立されるときに、基礎となる RRSAF CREATE THREAD 呼び出しで使用されるパッケージ・リストを指定します。 SQLJ プログラムまたは JDBC ドライバーでプランをバインドしない場合、このプロパティーを指定します。このプロパティーを指定する場合、**db2.jcc.planName は指定しないでください。**

db2.jcc.pkList は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。 db2.jcc.pkList は、CICS または IMS 下で実行されるアプリケーション、または Java ストアード・プロシージャーに対して実行されるアプリケーションには適用されません。JDBC ドライバーは、これらの場合に db2.jcc.pkList 設定を無視します。

推奨: db2.jcc.planName ではなく、db2.jcc.pkList を使用してください。

パッケージ・リストのフォーマットは以下のとおりです。

db2.jcc.pkList のデフォルト値は NULLID.* です。

com.ibm.db2.jcc.DB2Binder を実行するときに、-collection パラメーターを指定する場合、 com.ibm.db2.jcc.DB2Binder を実行するときに IBM Data Server Driver for JDBC and SQLJ パッケージ用に指定するコレクション ID も、db2.jcc.pkList プロパティーのパッケージ・リストに含まれている必要があります。

Connection または DataSource オブジェクトの pkList プロパティーを設定することによって、db2.jcc.pkList をオーバーライドすることができます。

以下の例は、コレクション JDBCCID にパッケージがある IBM Data Server Driver for JDBC and SQLJ インスタンスのパッケージ・リストを指定します。このドライバー・インスタンスの下で準備された SQLJ アプリケーションは、コレクション SQLJCID1、SQLJCID2、または SQLJCID3 にバインドされます。

```
db2.jcc.pkList=JDBCCID.*,SQLJCID1.*,SQLJCID2.*,SQLJCID3.*
```

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.planName

データ・ソースへの JDBC または SQLJ 接続が確立されるときに、基礎となる RRSAF CREATE THREAD 呼び出しで使用される DB2 for z/OS プラン名を指定します。 SQLJ プログラムおよび JDBC ドライバーでパッケージをバインドする場合、このプロパティーを指定します。このプロパティーを指定する場合、**db2.jcc.pkList は指定しないでください。**

db2.jcc.planName は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続に対してのみ適用されます。 db2.jcc.planName は、CICS または IMS 下で実行されるアプリケーション、または Java ストアード・プロシージャーに対して実行されるアプリケーションには適用されません。 JDBC ドライバーは、これらの場合に db2.jcc.planName 設定を無視します。

このプロパティーまたは db2.jcc.pkList プロパティーを指定しない場合、 IBM Data Server Driver for JDBC and SQLJ は db2.jcc.pkList のデフォルト値の NULLID.* を使用します。

db2.jcc.planName を指定する場合、com.ibm.db2.jcc.DB2Binder を実行するときに作成するパッケージを、名前がこのプロパティーの値であるプランにバインドする必要があります。すべての SQLJ パッケージも、名前がこのプロパティーの値であるプランにバインドする必要があります。

Connection または DataSource オブジェクトの planName プロパティーを設定することによって、db2.jcc.planName をオーバーライドすることができます。

以下の例は、 IBM Data Server Driver for JDBC and SQLJ の JDBC パッケージおよび SQLJ パッケージの MYPLAN のプラン名を指定します。

```
db2.jcc.planName=MYPLAN
```

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.progressiveStreaming または db2.jcc.override.progressiveStreaming
データ・ソース上で連続ストリーミングがサポートされている場合に、JDBC ドライバーが連続ストリーミングを使用するかどうかを指定します。

連続ストリーミング (動的データ形式ともいう) を使用すると、LOB または XML オブジェクトのサイズに基づいて、LOB または XML のデータを戻すのに最も効率のよいモードがデータ・ソースで動的に判別されます。

有効な値は以下のとおりです。

1 データ・ソースがサポートしている場合は連続ストリーミングを使用します。

2 連続ストリーミングは使用しません。

db2.jcc.rollbackOnShutdown
DB2 for z/OS が、JVM シャットダウン・フックの処理中に作業単位内にある JDBC 接続に対してロールバック操作を強制し、それ以外の操作も使用不可にするかどうかを指定します。

db2.jcc.rollbackOnShutdown は、 IBM Data Server Driver for JDBC and SQLJ Type 2 接続だけに適用されます。

db2.jcc.rollbackOnShutdown は、CICS、IMS、ストアード・プロシージャー、または WebSphere Application Server 環境には適用されません。

可能な値は以下のとおりです。

yes または true
IBM Data Server Driver for JDBC and SQLJ が DB2 for z/OS に、JVM シャットダウン・フックの処理中に作業単位内にある JDBC 接続に対してロールバック操作を強制し、それ以外の操作も使用不可にするように指示します。

その他の値
IBM Data Server Driver for JDBC and SQLJ は、JVM シャットダウン・フックの処理中にロールバック処理に関してアクションをとりません。これはデフォルトです。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.sendCharInputsUTF8
IBM Data Server Driver for JDBC and SQLJ が文字入力データを DB2 for z/OS データベース・サーバーの CCSID に変換するか、またはデータベース・サーバーによる変換のために UTF-8 エンコードでデータを送信するかを指定します。
db2.jcc.sendCharInputsUTF8 は、 DB2 for z/OS データベース・サーバーに対す

る IBM Data Server Driver for JDBC and SQLJ Type 2 接続だけに適用されます。このプロパティーが接続レベルでも設定されている場合、接続レベルの設定がこの値をオーバーライドします。

可能な値は以下のとおりです。

no、false、または 2

データが DB2 for z/OS データベース・サーバーに送信される前に、IBM Data Server Driver for JDBC and SQLJ が文字入力データをターゲット・エンコードに変換することを指定します。これはデフォルトです。

yes、true、または 1

IBM Data Server Driver for JDBC and SQLJ が文字入力データを UTF-8 エンコードで DB2 for z/OS データベース・サーバーに送信することを指定します。データ・ソースはデータを UTF-8 エンコードからターゲット CCSID に変換します。

yes、true、または 1 は、SDK for Java によるターゲット CCSID への変換によって文字変換問題が発生する場合にのみ指定します。 IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用して Unicode 改行文字 (U+000A) を CCSID 37 である表列に挿入し、そのデータを z/OS 以外のクライアントから取り出すときに、最も一般的な問題が発生します。文字を列に挿入するときに SDK for Java が変換を行う場合、改行文字は EBCDIC 改行文字 X'15' に変換されます。しかし、取り出し時には、z/OS 以外のオペレーティング・システム上の一部の SDK for Java は X'15' 文字を、改行文字 (U+000A) ではなく Unicode 改行文字 (U+0085) に変換します。改行文字により、一部の XML パーサーが予期しない動作を起こします。 db2.jcc.sendCharInputsUTF8 を yes に設定する場合、 DB2 for z/OS データベース・サーバーは列への挿入時に U+000A 文字を EBCDIC 改行文字 X'25' に変換します。そのため、文字は常に改行文字として取り出されます。

データ・ソース上でデータをターゲット CCSID に変換すると、 IBM Data Server Driver for JDBC and SQLJ が、ドライバーによる変換の場合よりも多くのメモリーを使用することがあります。ドライバーは文字データの変換用のメモリーを、ソース・エンコードからデータ・ソースに送信されるデータのエンコードに割り振ります。ドライバーが表列に送信される文字データに割り振るスペースの量は、データの最大長に基づいています。 UTF-8 データは各文字に最大で 3 バイトを必要とします。そのため、ドライバーが UTF-8 データをデータ・ソースに送信する場合、ドライバーは入力データ内の最大文字数の 3 倍を割り振る必要があります。ドライバーが変換を行う場合、ターゲット CCSID が 1 バイト CCSID であれば、ドライバーは入力データ内の最大文字数のみ割り振る必要があります。

例えば、db2.jcc.sendCharInputsUTF8 の以下のいずれかの設定により、データをデータ・ソースに送信する前に、 IBM Data Server Driver for JDBC and SQLJ は入力文字ストリングをターゲット・エンコードではなく UTF-8 に変換します。

```
db2.jcc.sendCharInputsUTF8=yes
db2.jcc.sendCharInputsUTF8=true
db2.jcc.sendCharInputsUTF8=1
```

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.sqljUncustomizedWarningOrException
カスタマイズされていない SQLJ アプリケーションが実行されるときに IBM Data Server Driver for JDBC and SQLJ が行うアクションを指定します。 db2.jcc.sqljUncustomizedWarningOrException は、次のような値をとることができます。

0 カスタマイズされていない SQLJ アプリケーションが実行されるとき、IBM Data Server Driver for JDBC and SQLJ は Warning または Exception をスローしません。 これはデフォルトです。

1 カスタマイズされていない SQLJ アプリケーションが実行されるとき、IBM Data Server Driver for JDBC and SQLJ は Warning をスローします。

2 カスタマイズされていない SQLJ アプリケーションが実行されるとき、IBM Data Server Driver for JDBC and SQLJ は Exception をスローします。

この構成プロパティーは、 DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows にのみ適用されます。

db2.jcc.traceDirectory または db2.jcc.override.traceDirectory
Java ドライバー・コードの IBM Data Server Driver for JDBC and SQLJ トレースを使用可能にし、トレース情報を書き込むディレクトリーを指定します。 db2.jcc.override.traceDirectory を指定すると、同一の DataSource 上にある複数の接続のトレース情報が、複数のファイルに書き込まれます。

db2.jcc.override.traceDirectory を指定すると、接続は *file-name_origin_n* という名前のファイルにトレースされます。

- *n* は、DataSource の *n* 番目の接続です。
- db2.jcc.traceFileName と db2.jcc.override.traceFileName のどちらも指定されない場合、 *file-name* は traceFile になります。 db2.jcc.traceFileName または db2.jcc.override.traceFileName も指定されている場合、*file-name* は db2.jcc.traceFileName または db2.jcc.override.traceFileName の値になります。
- *origin* は、使用中のログ書き込みプログラムの起点を示します。 *origin* に指定できる値は次のとおりです。

  cpds DB2ConnectionPoolDataSource オブジェクトのログ書き込みプログラム。

  driver DB2Driver オブジェクトのログ書き込みプログラム。

  global DB2TraceManager オブジェクトのログ書き込みプログラム。

  sds DB2SimpleDataSource オブジェクトのログ書き込みプログラム。

  xads DB2XADataSource オブジェクトのログ書き込みプログラム。

db2.jcc.override.traceDirectory プロパティーは、`Connection` または DataSource オブジェクトの traceDirectory プロパティーをオーバーライドします。

例えば、db2.jcc.override.traceDirectory の設定を以下のように指定することによって、 IBM Data Server Driver for JDBC and SQLJ の Java コードを /SYSTEM/tmp という名前のディレクトリーにあるファイルにトレースすることができるようになります。

```
db2.jcc.override.traceDirectory=/SYSTEM/tmp
```

トレースのプロパティーは、 IBM ソフトウェア・サポートの指示の下でのみ設定してください。

db2.jcc.traceLevel または db2.jcc.override.traceLevel
何をトレースするかを指定します。

db2.jcc.override.traceLevel プロパティーは、Connection または DataSource オブジェクトの traceLevel プロパティーをオーバーライドします。

10 進値を指定して、1 つ以上のトレース・レベルを指定します。このトレース・レベルは、Connection または DataSource オブジェクトの traceLevel プロパティーに定義されているトレース・レベルと同じです。

複数のトレース・レベルを指定する場合には、値に対して OR (|) 演算を行い、db2.jcc.traceLevel または db2.jcc.override.traceLevel の指定に結果を 10 進数で指定します。

例えば、TRACE_DRDA_FLOWS および TRACE_CONNECTIONS を db2.jcc.override.traceLevel に指定するとします。 TRACE_DRDA_FLOWS の値は 16 進値 X'40' です。TRACE_CONNECTION_CALLS の値は 16 進値 X'01' です。両方のトレースを指定するには、ビット単位 OR 演算をその 2 つの値に対して行います。結果は X'41' になります。これと等価の 10 進数は 65 です。それで、次のように指定します。

```
db2.jcc.override.traceLevel=65
```

db2.jcc.ssid
DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続を使用してアプリケーションが接続する DB2 for z/OS サブシステムを指定します。

db2.jcc.ssid の値は、ローカル DB2 サブシステムの名前またはグループ接続名にすることができます。

例:

```
db2.jcc.ssid=DB2A
```

ssid Connection および DataSource プロパティーは db2.jcc.ssid をオーバーライドします。

グループ接続名を指定する場合、アプリケーションの接続先の DB2 サブシステムに障害が起こると、接続は終了します。ただし、新しい接続がそのグループ接続名を使用するときに、 DB2 for z/OS はグループ接続処理を使用して、接続先になるアクティブな DB2 サブシステムを見つけます。

db2.jcc.ssid プロパティーを指定しない場合、 IBM Data Server Driver for JDBC and SQLJ は DSNHDECP のデータのみのロード・モジュールからの SSID 値を使用します。 DB2 for z/OS をインストールする場合、DSNHDECP モジュールは *prefix*.SDSNEXIT データ・セットおよび *prefix*.SDSNLOAD データ・セッ

トで作成されます。他の DSNHDECP ロード・モジュールは、選択されたアプリケーションの他のデータ・セットで作成される場合があります。

IBM Data Server Driver for JDBC and SQLJ は、SSID 値を読み取る前に、DSNHDECP モジュールをロードしておく必要があります。 z/OS は以下の場所にあるデータ・セットで、以下の順で DSNHDECP モジュールを検索します。

1. ジョブ・パック域 (JPA)
2. TASKLIB
3. STEPLIB または JOBLIB
4. LPA
5. リンク・リストにあるライブラリー

ご使用のシステムに DSNHDECP モジュールの複数のコピーがある場合、z/OS が IBM Data Server Driver for JDBC and SQLJ に対して適切なコピーを含むデータ・セットを最初に検索することを確認する必要があります。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.traceFile または db2.jcc.override.traceFile

Java ドライバー・コードの IBM Data Server Driver for JDBC and SQLJ トレースを使用可能にし、トレース・ファイル名の基になる名前を指定します。

db2.jcc.override.traceFile プロパティーの値には、完全修飾 z/OS UNIX システム・サービス・ファイル名を指定します。

db2.jcc.override.traceFile プロパティーは、`Connection` または DataSource オブジェクトの traceFile プロパティーをオーバーライドします。

例えば、db2.jcc.override.traceFile の設定を以下のように指定することによって、IBM Data Server Driver for JDBC and SQLJ の Java コードを /SYSTEM/tmp/jdbctrace という名前のファイルにトレースすることができるようになります。

```
db2.jcc.override.traceFile=/SYSTEM/tmp/jdbctrace
```

トレースのプロパティーは、 IBM ソフトウェア・サポートの指示の下でのみ設定してください。

db2.jcc.traceFileAppend または db2.jcc.override.traceFileAppend

db2.jcc.override.traceFile プロパティーによって指定されるファイルへ付加するか、あるいは上書きするかを指定します。 有効な値は `true` または `false` です。デフォルトは `false` です。これは、traceFile プロパティーによって指定されるファイルが上書きされることを意味します。

db2.jcc.override.traceFileAppend プロパティーは、`Connection` または DataSource オブジェクトの traceFileAppend プロパティーをオーバーライドします。

例えば、db2.jcc.override.traceFileAppend の設定を以下のように指定することによって、トレース・データは既存のトレース・ファイルに追加されるようになります。

```
db2.jcc.override.traceFileAppend=true
```

トレースのプロパティーは、 IBM ソフトウェア・サポートの指示の下でのみ設定してください。

db2.jcc.tracePolling

トレース・ディレクティブの変更を検出するために IBM Data Server Driver for

JDBC and SQLJ がグローバル構成ファイルをポーリングし、新しいトレース・ディレクティブに合わせてトレース動作を変更するかどうかを指定します。構成プロパティー・ファイルが変更された後、ドライバーは、次のポーリング間隔の初めにトレース動作を変更します。可能な値は true または false です。デフォルトは false です。トレース・ポーリングを有効にするには、ドライバーがロードされて初期化される*前に* db2.jcc.tracePolling プロパティーを有効にする必要があります。

db2.jcc.tracePolling は、以下のグローバル構成プロパティーをポーリングします。

- db2.jcc.override.traceLevel
- db2.jcc.override.traceFile
- db2.jcc.override.traceDirectory
- db2.jcc.override.traceFileAppend

db2.jcc.tracePollingInterval

トレース・ディレクティブの変更を検出するために IBM Data Server Driver for JDBC and SQLJ グローバル構成ファイルをポーリングする間隔 (秒) を指定します。このプロパティー値は正の整数です。デフォルトは 60 です。指定したトレース・ポーリング間隔が使用されるには、ドライバーがロードされて初期化される*前に* db2.jcc.tracePollingInterval プロパティーを設定する必要があります。ドライバーがロードされ初期化された後に db2.jcc.tracePollingInterval を変更しても、効果はありません。

db2.jcc.t2zosTraceFile

IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の C/C++ ネイティブ・ドライバー・コードの IBM Data Server Driver for JDBC and SQLJ トレースを使用可能にし、トレース・ファイルの名の基になる名前を指定します。このプロパティーは、C/C++ ネイティブ・ドライバー・コードのトレース・データの収集に必要です。

db2.jcct.t2zosTraceFile プロパティーの値には、完全修飾 z/OS UNIX システム・サービス・ファイル名を指定します。

例えば、db2.jcct.t2zosTraceFile の設定を以下のように指定することによって、IBM Data Server Driver for JDBC and SQLJ の C/C++ ネイティブ・コードを /SYSTEM/tmp/jdbctraceNative という名前のファイルにトレースすることができるようになります。

```
db2.jcc.t2zosTraceFile=/SYSTEM/tmp/jdbctraceNative
```

トレースのプロパティーは、 IBM ソフトウェア・サポートの指示の下でのみ設定してください。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.t2zosTraceBufferSize

C/C++ ネイティブ・ドライバー・コードによって実行される処理をトレースするのに使用される仮想ストレージ内のトレース・バッファーのサイズをキロバイト単位で指定します。この値は、収集できる C/C++ ネイティブ・ドライバーのトレース情報の最大量でもあります。

64 (64 KB) から 4096 (4096 KB) までの整数を指定します。デフォルトは 256 (256 KB) です。

JDBC ドライバーは、トレース・バッファーのサイズを次の表に示すようにして決定します。

指定された値 (*n*)	トレース・バッファーのサイズ (KB)
<64	64
64<=*n*<128	64
128<=*n*<256	128
256<=*n*<512	256
512<=*n*<1024	512
1024<=*n*<2048	1024
2048<=*n*<4096	2048
n>=4096	4096

db2.jcc.t2zosTraceBufferSize は、db2.jcc.t2zosTraceFile プロパティーが設定されている場合にのみ使用されます。

推奨: パフォーマンスへの影響を避けるために、1024 以下の値を指定してください。

例えば、トレース・バッファーのサイズを 1024 KB に設定するには、以下の設定を使用します。

```
db2.jcc.t2zosTraceBufferSize=1024
```

トレースのプロパティーは、 IBM ソフトウェア・サポートの指示の下でのみ設定してください。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.t2zosTraceWrap

SQLJ トレースの折り返しを使用可能または使用不可にします。
db2.jcc.t2zosTraceWrap は以下の値をとることができます。

1 トレースを折り返します。

0 トレースを折り返しません。

デフォルトは 1 です。このパラメーターはオプションです。例:

```
DB2SQLJ_TRACE_WRAP=0
```

db2.jcc.t2zosTraceWrap は、 IBM ソフトウェア・サポートの指示の下でのみ設定してください。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

db2.jcc.useCcsid420ShapedConverter

EBCDIC CCSID 420 であるアラビア語文字データが Cp420S エンコードにマップするかどうかを指定します。

db2.jcc.useCcsid420ShapedConverter は、 DB2 for z/OS データベース・サーバーへの接続にのみ適用されます。

db2.jcc.useCcsid420ShapedConverter の値が `true` の場合、CCSID 420 は Cp420S エンコードにマップします。db2.jcc.useCcsid420ShapedConverter の値が `false` の場合、CCSID 420 は Cp420 エンコードにマップします。デフォルトは `false` です。

この構成プロパティーは、 DB2 for z/OS にのみ適用されます。

JDBC API でのドライバーのサポート

DB2 および IBM Informix Dynamic Server (IDS) データベース・システムによってサポートされる JDBC ドライバーは、JDBC メソッドについて異なるレベルのサポートを持っています。

次の表は JDBC インターフェースをリストし、それらがどのドライバーによってサポートされるかを示しています。ドライバーおよびサポートされるプラットフォームは、以下のとおりです。

表 48. DB2 および IDS データベース・システム用の JDBC ドライバー

JDBC ドライバー名	関連するデータ・ソース
IBM Data Server Driver for JDBC and SQLJ	DB2 Database for Linux, UNIX, and Windows、 DB2 for z/OS、または IBM Informix Dynamic Server (IDS)
Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (非推奨)	DB2 Database for Linux, UNIX, and Windows
IBM Informix JDBC ドライバー (IDS JDBC ドライバー)	IDS

メソッドの形式が JDBC 2.0 および JDBC 3.0 の場合、 IBM Data Server Driver for JDBC and SQLJ はすべての形式をサポートします。Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーは JDBC 2.0 形式のみサポートします。

表 49. Array メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ1 のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
free[2]	はい	いいえ	いいえ
getArray	はい	いいえ	はい
getBaseType	はい	いいえ	はい
getBaseTypeName	はい	いいえ	はい
getResultSet	はい	いいえ	はい

注:

1. IBM Data Server Driver for JDBC and SQLJ では、 DB2 Database for Linux, UNIX, and WindowsArray メソッドがサポートされます。
2. これは JDBC 4.0 のメソッドです。

表 50. *BatchUpdateException* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	はい	はい
getUpdateCounts	はい	はい	はい

表 51. *Blob* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
free[1]	はい	いいえ	いいえ
getBinaryStream	はい[2]	はい	はい
getBytes	はい	はい	はい
length	はい	はい	はい
position	はい	はい	はい
setBinaryStream[3]	はい	いいえ	いいえ
setBytes[3]	はい	いいえ	いいえ
truncate[3]	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 のメソッドです。
2. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   `getBinaryStream(long pos, long length)`
3. IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 より前のバージョンでは、Blob が IN または INOUT パラメーターとしてストアード・プロシージャーに渡される場合にはこれらのメソッドを使用できず、ストアード・プロシージャー内の Blob に対してメソッドが使用されます。

表 52. *CallableStatement* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.sql.Statement から継承されたメソッド	はい	はい	はい
java.sql.PreparedStatement から継承されたメソッド	はい[1]	はい	はい
getArray	いいえ	いいえ	いいえ
getBigDecimal	はい[3]	はい	はい
getBlob	はい[3]	はい	はい
getBoolean	はい[3]	はい	はい
getByte	はい[3]	はい	はい
getBytes	はい[3]	はい	はい

表 52. *CallableStatement* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getClob	はい[3]	はい	はい
getDate	はい[3、 4]	はい[4]	はい
getDouble	はい[3]	はい	はい
getFloat	はい[3]	はい	はい
getInt	はい[3]	はい	はい
getLong	はい[3]	はい	はい
getObject	はい[3、 5]	はい[5]	はい
getRef	いいえ	いいえ	いいえ
getRowId[2]	はい	いいえ	いいえ
getShort	はい[3]	はい	はい
getString	はい[3]	はい	はい
getTime	はい[3、 4]	はい[4]	はい
getTimestamp	はい[3、 4]	はい[4]	はい
getURL	はい	いいえ	いいえ
registerOutParameter	はい[6]	はい[6]	はい[6]
setAsciiStream	はい[7]	いいえ	はい
setBigDecimal	はい[7]	いいえ	はい
setBinaryStream	はい[7]	いいえ	はい
setBoolean	はい[7]	いいえ	はい
setByte	はい[7]	いいえ	はい
setBytes	はい[7]	いいえ	はい
setCharacterStream	はい[7]	いいえ	はい
setDate	はい[7]	いいえ	はい
setDouble	はい[7]	いいえ	はい
setFloat	はい[7]	いいえ	はい
setInt	はい[7]	いいえ	はい
setLong	はい[7]	いいえ	はい
setNull	はい[7、 8]	いいえ	はい
setObject	はい[7、]	いいえ	はい
setShort	はい[7]	いいえ	はい
setString	はい[7]	いいえ	はい
setTime	はい[7]	いいえ	はい
setTimestamp	はい[7]	いいえ	はい
setURL	はい	いいえ	いいえ
wasNull	はい	はい	はい

表 52. *CallableStatement* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート

注:

1. 継承された getParameterMetaData メソッドは、データ・ソースが DB2 for z/OS の場合にはサポートされません。
2. これは JDBC 4.0 のメソッドです。
3. 以下の形式の CallableStatement.getXXX メソッドは、データ・ソースが DB2 for z/OS の場合にはサポートされません。

   `get`*`XXX`*`(String` *`parameterName`*`)`
4. データベース・サーバーは日時値の時間帯調整を行いません。 JDBC ドライバーは、java.util.Calendar パラメーターを含む getDate、getTime、または getTimestamp メソッドの形式が指定された場合、サーバーから値を取り出した後でローカル時間帯の値を調整します。
5. 次の形式の getObject メソッドはサポートされません。

   `getObject(int` *`parameterIndex`*`, java.util.Map` *`map`*`)`
6. 次の形式の registerOutParameter メソッドはサポートされません。

   `registerOutParameter(int` *`parameterIndex`*`, int` *`jdbcType`*`, String` *`typeName`*`)`
7. 以下の形式の CallableStatement.setXXX メソッドは、データ・ソースが DB2 for z/OS の場合にはサポートされません。

   `set`*`XXX`*`(String` *`parameterName`*`,...)`
8. 次の形式の setNull はサポートされません。

   `setNull(int` *`parameterIndex`*`, int` *`jdbcType`*`, String` *`typeName`*`)`

表 53. *Clob* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
free[1]	はい	いいえ	いいえ
getAsciiStream	はい	はい	はい
getCharacterStream	はい2 (358 ページ)	はい	はい
getSubString	はい	はい	はい
length	はい	はい	はい
position	はい	はい	はい
setAsciiStream[3]	はい	いいえ	はい
setCharacterStream[3]	はい	いいえ	はい
setString[3]	はい	いいえ	はい
truncate[3]	はい	いいえ	はい

表 53. Clob メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート

注:

1. これは JDBC 4.0 のメソッドです。
2. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   `getCharacterStream(long pos, long length)`
3. IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 より前のバージョンでは、Clob が IN または INOUT パラメーターとしてストアード・プロシージャーに渡される場合にはこれらのメソッドを使用できず、ストアード・プロシージャー内の Clob に対してメソッドが使用されます。

表 54. Connection メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
clearWarnings	はい	はい	はい
close	はい	はい	はい
commit	はい	はい	はい
createStatement	はい	はい[2]	はい
createBlob[1]	はい	いいえ	いいえ
createClob[1]	はい	いいえ	いいえ
getAutoCommit	はい	はい	はい
getCatalog	はい	はい	はい
getClientInfo[3]	はい	いいえ	いいえ
getHoldability	はい	いいえ	いいえ
getMetaData	はい	はい	はい
getTransactionIsolation	はい	はい	はい
getTypeMap	いいえ	いいえ	はい
getWarnings	はい	はい	はい
isClosed	はい	はい	はい
isReadOnly	はい	はい	はい
isValid[3,4]	はい	いいえ	いいえ
nativeSQL	はい	はい	はい
prepareCall	はい[5]	はい	はい
prepareStatement	はい	はい[2]	はい
releaseSavepoint	はい	いいえ	いいえ
rollback	はい	はい[2]	はい
setAutoCommit	はい	はい	はい
setCatalog	はい	はい	いいえ
setClientInfo[3]	はい	いいえ	いいえ

表 54. *Connection* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
setReadOnly	はい[6]	はい	いいえ
setSavepoint	はい	いいえ	いいえ
setTransactionIsolation	はい	はい	はい
setTypeMap	いいえ	いいえ	はい

注:

1. これは JDBC 4.0 のメソッドです。
2. Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーは、このメソッドの JDBC 3.0 形式をサポートしません。
3. これは JDBC 4.0 のメソッドです。
4. IBM Data Server Driver for JDBC and SQLJ Type 4 接続 の場合、*timeout* パラメーターの値が 0 より小さい場合、SQLException がスローされます。 IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合、*timeout* パラメーターの値が 0 でない場合、SQLException がスローされます。
5. CALL ステートメント内のストアード・プロシージャーが DB2 for z/OS 上にある場合、CALL ステートメントのパラメーターを式にすることはできません。
6. ドライバーは設定を使用しません。 IBM Data Server Driver for JDBC and SQLJ では、`Connection` または DataSource オブジェクトの readOnly プロパティーにより、接続を読み取り専用として設定することができます。

表 55. *ConnectionEvent* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.util.EventObject から継承されたメソッド	はい	はい	はい
getSQLException	はい	はい	はい

表 56. *ConnectionEventListener* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
connectionClosed	はい	はい	はい
connectionErrorOccurred	はい	はい	はい

表 57. *ConnectionPoolDataSource* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getLoginTimeout	はい	はい	はい

表 57. *ConnectionPoolDataSource* メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getLogWriter	はい	はい	はい
getPooledConnection	はい	はい	はい
setLoginTimeout	はい[1]	はい	はい
setLogWriter	はい	はい	はい

注:

1. このメソッドは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続ではサポートされません。

表 58. *DatabaseMetaData* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
allProceduresAreCallable	はい	はい	はい
allTablesAreSelectable	はい[1]	はい	はい[1]
dataDefinitionCausesTransactionCommit	はい	はい	はい
dataDefinitionIgnoredInTransactions	はい	はい	はい
deletesAreDetected	はい	はい	はい
doesMaxRowSizeIncludeBlobs	はい	はい	はい
getAttributes	はい[2]	いいえ	いいえ
getBestRowIdentifier	はい	はい	はい
getCatalogs	はい	はい	はい
getCatalogSeparator	はい	はい	はい
getCatalogTerm	はい	はい	はい
getClientInfoProperties[6]	はい	いいえ	いいえ
getColumnPrivileges	はい	はい	はい
getColumns	はい[7]	はい[10]	はい[10]
getConnection	はい	はい	はい
getCrossReference	はい	はい	はい
getDatabaseMajorVersion	はい	いいえ	いいえ
getDatabaseMinorVersion	はい	いいえ	いいえ
getDatabaseProductName	はい	はい	はい
getDatabaseProductVersion	はい	はい	はい
getDefaultTransactionIsolation	はい	はい	はい
getDriverMajorVersion	はい	はい	はい
getDriverMinorVersion	はい	はい	はい
getDriverName	はい[8]	はい	はい

表 58. *DatabaseMetaData* メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getDriverVersion	はい	はい	はい
getExportedKeys	はい	はい	はい
getFunctionColumns[6]	はい	いいえ	いいえ
getFunctions[6]	はい	いいえ	いいえ
getExtraNameCharacters	はい	はい	はい
getIdentifierQuoteString	はい	はい	はい
getImportedKeys	はい	はい	はい
getIndexInfo	はい	はい	はい
getJDBCMajorVersion	はい	いいえ	いいえ
getJDBCMinorVersion	はい	いいえ	いいえ
getMaxBinaryLiteralLength	はい	はい	はい
getMaxCatalogNameLength	はい	はい	はい
getMaxCharLiteralLength	はい	はい	はい
getMaxColumnNameLength	はい	はい	はい
getMaxColumnsInGroupBy	はい	はい	はい
getMaxColumnsInIndex	はい	はい	はい
getMaxColumnsInOrderBy	はい	はい	はい
getMaxColumnsInSelect	はい	はい	はい
getMaxColumnsInTable	はい	はい	はい
getMaxConnections	はい	はい	はい
getMaxCursorNameLength	はい	はい	はい
getMaxIndexLength	はい	はい	はい
getMaxProcedureNameLength	はい	はい	はい
getMaxRowSize	はい	はい	はい
getMaxSchemaNameLength	はい	はい	はい
getMaxStatementLength	はい	はい	はい
getMaxStatements	はい	はい	はい
getMaxTableNameLength	はい	はい	はい
getMaxTablesInSelect	はい	はい	はい
getMaxUserNameLength	はい	はい	はい
getNumericFunctions	はい	はい	はい
getPrimaryKeys	はい	はい	はい
getProcedureColumns	はい7 (365 ページ)	はい	はい
getProcedures	はい7 (365 ページ)	はい	はい

表 58. *DatabaseMetaData* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getProcedureTerm	はい	はい	はい
getResultSetHoldability	はい	いいえ	いいえ
getRowIdLifetime[6]	はい	いいえ	いいえ
getSchemas	はい9 (365 ページ)	はい[10]	はい[10]
getSchemaTerm	はい	はい	はい
getSearchStringEscape	はい	はい	はい
getSQLKeywords	はい	はい	はい
getSQLStateType	はい	いいえ	いいえ
getStringFunctions	はい	はい	はい
getSuperTables	はい[2]	いいえ	いいえ
getSuperTypes	はい[2]	いいえ	いいえ
getSystemFunctions	はい	はい	はい
getTablePrivileges	はい	はい	はい
getTables	はい	はい[10]	はい[10]
getTableTypes	はい	はい	はい
getTimeDateFunctions	はい	はい	はい
getTypeInfo	はい	はい	はい
getUDTs	いいえ	はい[11]	はい[11]
getURL	はい	はい	はい
getUserName	はい	はい	はい
getVersionColumns	はい	はい	はい
insertsAreDetected	はい	はい	はい
isCatalogAtStart	はい	はい	はい
isReadOnly	はい	はい	はい
locatorsUpdateCopy	はい[3]	はい	はい[3]
nullPlusNonNullIsNull	はい	はい	はい
nullsAreSortedAtEnd	はい[4]	はい	はい[4]
nullsAreSortedAtStart	はい	はい	はい
nullsAreSortedHigh	はい[5]	はい	はい[5]
nullsAreSortedLow	はい[1]	はい	はい[1]
othersDeletesAreVisible	はい	はい	はい
othersInsertsAreVisible	はい	はい	はい
othersUpdatesAreVisible	はい	はい	はい
ownDeletesAreVisible	はい	はい	はい
ownInsertsAreVisible	はい	はい	はい

表 58. *DatabaseMetaData* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
ownUpdatesAreVisible	はい	はい	はい
storesLowerCaseIdentifiers	はい[1]	はい	はい[1]
storesLowerCaseQuotedIdentifiers	はい[4]	はい	はい[4]
storesMixedCaseIdentifiers	はい	はい	はい
storesMixedCaseQuotedIdentifiers	はい	はい	はい
storesUpperCaseIdentifiers	はい[5]	はい	はい[5]
storesUpperCaseQuotedIdentifiers	はい	はい	はい
supportsAlterTableWithAddColumn	はい	はい	はい
supportsAlterTableWithDropColumn	はい[1]	はい	はい[1]
supportsANSI92EntryLevelSQL	はい	はい	はい
supportsANSI92FullSQL	はい	はい	はい
supportsANSI92IntermediateSQL	はい	はい	はい
supportsBatchUpdates	はい	はい	はい
supportsCatalogsInDataManipulation	はい[1]	はい	はい[1]
supportsCatalogsInIndexDefinitions	はい	はい	はい
supportsCatalogsInPrivilegeDefinitions	はい	はい	はい
supportsCatalogsInProcedureCalls	はい[1]	はい	はい[1]
supportsCatalogsInTableDefinitions	はい	はい	はい
SupportsColumnAliasing	はい	はい	はい
supportsConvert	はい	はい	はい
supportsCoreSQLGrammar	はい	はい	はい
supportsCorrelatedSubqueries	はい	はい	はい
supportsDataDefinitionAndDataManipulationTransactions	はい	はい	はい
supportsDataManipulationTransactionsOnly	はい	はい	はい
supportsDifferentTableCorrelationNames	はい[4]	はい	はい[4]
supportsExpressionsInOrderBy	はい	はい	はい
supportsExtendedSQLGrammar	はい	はい	はい
supportsFullOuterJoins	はい[3]	はい	はい[3]
supportsGetGeneratedKeys	はい	いいえ	いいえ
supportsGroupBy	はい	はい	はい
supportsGroupByBeyondSelect	はい	はい	はい
supportsGroupByUnrelated	はい	はい	はい
supportsIntegrityEnhancementFacility	はい	はい	はい
supportsLikeEscapeClause	はい	はい	はい
supportsLimitedOuterJoins	はい	はい	はい
supportsMinimumSQLGrammar	はい	はい	はい

表 58. *DatabaseMetaData* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
supportsMixedCaseIdentifiers	はい	はい	はい
supportsMixedCaseQuotedIdentifiers	はい[3]	はい	はい[3]
supportsMultipleOpenResults	はい[5]	いいえ	はい[5]
supportsMultipleResultSets	はい[5]	はい	はい[5]
supportsMultipleTransactions	はい	はい	はい
supportsNamedParameters	はい	いいえ	いいえ
supportsNonNullableColumns	はい	はい	はい
supportsOpenCursorsAcrossCommit	はい[3]	はい	はい[3]
supportsOpenCursorsAcrossRollback	はい	はい	はい
supportsOpenStatementsAcrossCommit	はい[3]	はい	はい[3]
supportsOpenStatementsAcrossRollback	はい[3]	はい	はい[3]
supportsOrderByUnrelated	はい	はい	はい
supportsOuterJoins	はい	はい	はい
supportsPositionedDelete	はい	はい	はい
supportsPositionedUpdate	はい	はい	はい
supportsResultSetConcurrency	はい	はい	はい
supportsResultSetHoldability	はい	いいえ	いいえ
supportsResultSetType	はい	はい	はい
supportsSavepoints	はい	いいえ	はい
supportsSchemasInDataManipulation	はい	はい	はい
supportsSchemasInIndexDefinitions	はい	はい	はい
supportsSchemasInPrivilegeDefinitions	はい	はい	はい
supportsSchemasInProcedureCalls	はい	はい	はい
supportsSchemasInTableDefinitions	はい	はい	はい
supportsSelectForUpdate	はい	はい	はい
supportsStoredProcedures	はい	はい	はい
supportsSubqueriesInComparisons	はい	はい	はい
supportsSubqueriesInExists	はい	はい	はい
supportsSubqueriesInIns	はい	はい	はい
supportsSubqueriesInQuantifieds	はい	はい	はい
supportsSuperTables	はい	いいえ	いいえ
supportsSuperTypes	はい	いいえ	いいえ
supportsTableCorrelationNames	はい	はい	はい
supportsTransactionIsolationLevel	はい	はい	はい
supportsTransactions	はい	はい	はい
supportsUnion	はい	はい	はい

表 58. *DatabaseMetaData* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
supportsUnionAll	はい	はい	はい
updatesAreDetected	はい	はい	はい
usesLocalFilePerTable	はい	はい	はい
usesLocalFiles	はい	はい	はい

注:

1. DB2 データ・ソースはこのメソッドに `false` を戻します。 IDS データ・ソースは `true` を戻します。
2. このメソッドは、 DB2 Database for Linux, UNIX, and Windows および IDS への接続の場合のみサポートされます。
3. IBM Data Server Driver for JDBC and SQLJ では、 DB2 データ・ソースおよび IDS データ・ソースはこのメソッドに `true` を戻します。 IDS JDBC ドライバーでは、IDS データ・ソースは `false` を戻します。
4. IBM Data Server Driver for JDBC and SQLJ では、 DB2 データ・ソースおよび IDS データ・ソースはこのメソッドに `false` を戻します。 IDS JDBC ドライバーでは、IDS データ・ソースは `true` を戻します。
5. DB2 データ・ソースはこのメソッドに `true` を戻します。 IDS データ・ソースは `false` を戻します。
6. これは JDBC 4.0 のメソッドです。
7. このメソッドは、JDBC 4.0 仕様で記述された追加列を戻します。
8. IBM Data Server Driver for JDBC and SQLJ の JDBC 3.0 以前のインプリメンテーションでは、「IBM DB2 JDBC Universal Driver Architecture」を戻します。

   IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは、「 IBM Data Server Driver for JDBC and SQLJ」を戻します。
9. このメソッドの JDBC 4.0 以前の形式がサポートされます。
10. Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーは、このメソッドの JDBC 3.0 形式をサポートしません。
11. メソッドは実行されますが、空の ResultSet を戻します。

表 59. *DataSource* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getConnection	はい	はい	はい
getLoginTimeout	はい	はい[1]	はい
getLogWriter	はい	はい	はい
setLoginTimeout	はい[2]	はい[1]	はい
setLogWriter	はい	はい	はい

表 59. *DataSource* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート

注:

1. DB2 JDBC Type 2 ドライバーはこの設定を使用しません。
2. このメソッドは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続ではサポートされません。

表 60. *DataTruncation* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Throwable から継承されたメソッド	はい	はい	はい
java.sql.SQLException から継承されたメソッド	はい	はい	はい
java.sql.SQLWarning から継承されたメソッド	はい	はい	はい
getDataSize	はい	はい	はい
getIndex	はい	はい	はい
getParameter	はい	はい	はい
getRead	はい	はい	はい
getTransferSize	はい	はい	はい

表 61. *Driver* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
acceptsURL	はい	はい	はい
connect	はい	はい	はい
getMajorVersion	はい	はい	はい
getMinorVersion	はい	はい	はい
getPropertyInfo	はい	はい	はい
jdbcCompliant	はい	はい	はい

表 62. *DriverManager* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
deregisterDriver	はい	はい	はい

表 62. *DriverManager* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getConnection	はい	はい	はい
getDriver	はい	はい	はい
getDrivers	はい	はい	はい
getLoginTimeout	はい	はい[1]	はい[1]
getLogStream	はい	はい	はい
getLogWriter	はい	はい	はい
println	はい	はい	はい
registerDriver	はい	はい	はい
setLoginTimeout	はい[2]	はい[1]	はい[1]
setLogStream	はい	はい	はい
setLogWriter	はい	はい	はい

注:

1. DB2 JDBC Type 2 ドライバーはこの設定を使用しません。
2. このメソッドは、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続ではサポートされません。

表 63. *ParameterMetaData* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getParameterClassName	いいえ	いいえ	いいえ
getParameterCount	はい	いいえ	いいえ
getParameterMode	はい	いいえ	いいえ
getParameterType	はい	いいえ	いいえ
getParameterTypeName	はい	いいえ	いいえ
getPrecision	はい	いいえ	いいえ
getScale	はい	いいえ	いいえ
isNullable	はい	いいえ	いいえ
isSigned	はい	いいえ	いいえ

表 64. *PooledConnection* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
addConnectionEventListener	はい	はい	はい
addStatementEventListener[1]	はい	いいえ	いいえ
close	はい	はい	はい

表 64. PooledConnection メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getConnection	はい	はい	はい
removeConnectionEventListener	はい	はい	はい
removeStatementEventListener[1]	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 のメソッドです。

表 65. PreparedStatement メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.sql.Statement から継承されたメソッド	はい	はい	はい
addBatch	はい	はい	はい
clearParameters	はい	はい	はい
execute	はい	はい	はい
executeQuery	はい	はい	はい
executeUpdate	はい	はい	はい
getMetaData	はい	はい	はい
getParameterMetaData	はい	はい	はい
setArray	いいえ	いいえ	いいえ
setAsciiStream	はい[1、 2]	はい	はい
setBigDecimal	はい	はい	はい
setBinaryStream	はい[1、 3]	はい	はい
setBlob	はい[4]	はい	はい
setBoolean	はい	はい	はい
setByte	はい	はい	はい
setBytes	はい	はい	はい
setCharacterStream	はい[1、 5]	はい	はい
setClob	はい[6]	はい	はい
setDate	はい[8]	はい[8]	はい[8]
setDouble	はい	はい	はい
setFloat	はい	はい	はい
setInt	はい	はい	はい
setLong	はい	はい	はい
setNull	はい[9]	はい[9]	はい[9]
setObject	はい	はい	はい
setRef	いいえ	いいえ	いいえ

表 65. *PreparedStatement* メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
setRowId[7]	はい	いいえ	いいえ
setShort	はい	はい	はい
setString	はい[10]	はい[10]	はい[10]
setTime	はい[8]	はい[8]	はい[8]
setTimestamp	はい[8]	はい[8]	はい[8]
setUnicodeStream	はい	はい	はい
setURL	はい	はい	はい

注:

1. *length* パラメーターの値が -1 の場合、InputStream または Reader からのすべてのデータが読み取られてデータ・ソースに送信されます。
2. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   setAsciiStream(int parameterIndex, InputStream x, long length)
   setAsciiStream(int parameterIndex, InputStream x)
   ```
3. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   setBinaryStream(int parameterIndex, InputStream x, long length)
   setBinaryStream(int parameterIndex, InputStream x)
   ```
4. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   setBlob(int parameterIndex, InputStream inputStream, long length)
   ```
5. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   setCharacterStream(int parameterIndex, Reader reader, long length)
   setCharacterStream(int parameterIndex, Reader reader)
   ```
6. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   setClob(int parameterIndex, Reader reader, long length)
   ```
7. これは JDBC 4.0 のメソッドです。
8. データベース・サーバーは日時値の時間帯調整を行いません。 JDBC ドライバーは、java.util.Calendar パラメーターを含む setDate、setTime、または setTimestamp メソッドの形式が指定された場合、値をサーバーに送信する前にローカル時間帯の値を調整します。
9. 次の形式の setNull はサポートされません。

   ```
   setNull(int parameterIndex, int jdbcType, String typeName)
   ```
10. 列に FOR BIT DATA 属性がある場合またはデータ・タイプが BLOB である場合、setString はサポートされません。

表 66. *Ref* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
get BaseTypeName	いいえ	いいえ	いいえ

表 67. *ResultSet* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
absolute	はい	はい	はい
afterLast	はい	はい	はい
beforeFirst	はい	はい	はい
cancelRowUpdates	はい	いいえ	いいえ
clearWarnings	はい	はい	はい
close	はい	はい	はい
deleteRow	はい	いいえ	いいえ
findColumn	はい	はい	はい
first	はい	はい	はい
getArray	いいえ	いいえ	いいえ
getAsciiStream	はい	はい	はい
getBigDecimal	はい	はい	はい
getBinaryStream	はい[1]	はい	はい
getBlob	はい	はい	はい
getBoolean	はい	はい	はい
getByte	はい	はい	はい
getBytes	はい	はい	はい
getCharacterStream	はい	はい	はい
getClob	はい	はい	はい
getConcurrency	はい	はい	はい
getCursorName	はい	はい	はい
getDate	はい[3]	はい[3]	はい[3]
getDouble	はい	はい	はい
getFetchDirection	はい	はい	はい
getFetchSize	はい	はい	はい
getFloat	はい	はい	はい
getInt	はい	はい	はい
getLong	はい	はい	はい
getMetaData	はい	はい	はい
getObject	はい[4]	はい[4]	はい[4]
getRef	いいえ	いいえ	いいえ
getRow	はい	はい	はい
getRowId[10]	はい	いいえ	いいえ
getShort	はい	はい	はい
getStatement	はい	はい	はい
getString	はい	はい	はい
getTime	はい[3]	はい[3]	はい[3]

表 67. *ResultSet* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getTimestamp	はい[3]	はい[3]	はい[3]
getType	はい	はい	はい
getUnicodeStream	はい	はい	はい
getURL	はい	はい	はい
getWarnings	はい	はい	はい
insertRow	はい	いいえ	いいえ
isAfterLast	はい	はい	はい
isBeforeFirst	はい	はい	はい
isFirst	はい	はい	はい
isLast	はい	はい	はい
last	はい	はい	はい
moveToCurrentRow	はい	いいえ	いいえ
moveToInsertRow	はい	いいえ	いいえ
next	はい	はい	はい
previous	はい	はい	はい
refreshRow	はい	いいえ	いいえ
relative	はい	はい	はい
rowDeleted	はい	いいえ	いいえ
rowInserted	はい	いいえ	いいえ
rowUpdated	はい	いいえ	いいえ
setFetchDirection	はい	はい	はい
setFetchSize	はい	はい	はい
updateArray	いいえ	いいえ	いいえ
updateAsciiStream	はい[5]	いいえ	いいえ
updateBigDecimal	はい	いいえ	いいえ
updateBinaryStream	はい[6]	いいえ	いいえ
updateBlob	はい[7]	いいえ	いいえ
updateBoolean	はい	いいえ	いいえ
updateByte	はい	いいえ	いいえ
updateBytes	はい	いいえ	いいえ
updateCharacterStream	はい[8]	いいえ	いいえ
updateClob	はい[9]	いいえ	いいえ
updateDate	はい	いいえ	いいえ
updateDouble	はい	いいえ	いいえ
updateFloat	はい	いいえ	いいえ
updateInt	はい	いいえ	いいえ
updateLong	はい	いいえ	いいえ

表 67. *ResultSet* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
updateNull	はい	いいえ	いいえ
updateObject	はい	いいえ	いいえ
updateRef	いいえ	いいえ	いいえ
updateRow	はい	いいえ	いいえ
updateRowId[10]	はい	いいえ	いいえ
updateShort	はい	いいえ	いいえ
updateString	はい	いいえ	いいえ
updateTime	はい	いいえ	いいえ
updateTimestamp	はい	いいえ	いいえ
wasNull	はい	はい	はい

表 67. *ResultSet* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート

注:

1. getBinaryStream は CLOB 列にサポートされません。
2. getMetaData は、返されたスキーマ名が 8 文字未満の場合、スキーマ名に埋め込みを行って、8 文字にします。
3. データベース・サーバーは日時値の時間帯調整を行いません。 JDBC ドライバーは、java.util.Calendar パラメーターを含む getDate、getTime、または getTimestamp メソッドの形式が指定された場合、サーバーから値を取り出した後でローカル時間帯の値を調整します。
4. 次の形式の getObject メソッドはサポートされません。

   ```
   getObject(int parameterIndex, java.util.Map  map)
   ```

5. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   updateAsciiStream(int columnIndex, InputStream x)
   updateAsciiStream(String columnLabel, InputStream x)
   updateAsciiStream(int columnIndex, InputStream x, long length)
   updateAsciiStream(String columnLabel, InputStream x, long length)
   ```

6. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   updateBinaryStream(int columnIndex, InputStream x)
   updateBinaryStream(String columnLabel, InputStream x)
   updateBinaryStream(int columnIndex, InputStream x, long length)
   updateBinaryStream(String columnLabel, InputStream x, long length)
   ```

7. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   updateBlob(int columnIndex, InputStream x)
   updateBlob(String columnLabel, InputStream x)
   updateBlob(int columnIndex, InputStream x, long length)
   updateBlob(String columnLabel, InputStream x, long length)
   ```

8. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   updateCharacterStream(int columnIndex, Reader reader)
   updateCharacterStream(String columnLabel, Reader reader)
   updateCharacterStream(int columnIndex, Reader reader, long length)
   updateCharacterStream(String columnLabel, Reader reader, long length)
   ```

9. このメソッドのサポートされる形式には、以下の JDBC 4.0 形式が含まれます。

   ```
   updateClob(int columnIndex, Reader reader)
   updateClob(String columnLabel, Reader reader)
   updateClob(int columnIndex, Reader reader, long length)
   updateClob(String columnLabel, Reader reader, long length)
   ```

10. これは JDBC 4.0 のメソッドです。

表 68. *ResultSetMetaData* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getCatalogName	はい	はい	はい
getColumnClassName	いいえ	はい	はい
getColumnCount	はい	はい	はい
getColumnDisplaySize	はい	はい	はい

表 68. *ResultSetMetaData* メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getColumnLabel	はい	はい	はい
getColumnName	はい	はい	はい
getColumnType	はい	はい	はい
getColumnTypeName	はい	はい	はい
getPrecision	はい	はい	はい
getScale	はい	はい	はい
getSchemaName	はい	はい	はい
getTableName	はい[1]	はい	はい
isAutoIncrement	はい	はい	はい
isCaseSensitive	はい	はい	はい
isCurrency	はい	はい	はい
isDefinitelyWritable	はい	はい	はい
isNullable	はい	はい	はい
isReadOnly	はい	はい	はい
isSearchable	はい	はい	はい
isSigned	はい	はい	はい
isWritable	はい	はい	はい

注:

1. IDS データ・ソースの場合、getTableName は値を戻しません。
2. getSchemaName は、返されたスキーマ名が 8 文字未満の場合、スキーマ名に埋め込みを行って、8 文字にします。

表 69. *RowId* メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート[2]	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
equals	はい	いいえ	いいえ
getBytes	はい	いいえ	いいえ
hashCode	いいえ	いいえ	いいえ
toString	はい	いいえ	いいえ

注:

1. これらのメソッドは JDBC 4.0 メソッドです。
2. これらのメソッドは、 DB2 for z/OS、DB2 for i、および IDS データ・ソースへの接続にサポートされます。

表 70. SQLClientInfoException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ
getFailedProperties	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 71. SQLData メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getSQLTypeName	いいえ	いいえ	いいえ
readSQL	いいえ	いいえ	いいえ
writeSQL	いいえ	いいえ	いいえ

表 72. SQLDataException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 73. SQLException メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	はい	はい
getSQLState	はい	はい	はい

表 73. SQLException メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getErrorCode	はい	はい	はい
getNextException	はい	はい	はい
setNextException	はい	はい	はい

表 74. SQLFeatureNotSupported メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 75. SQLInput メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
readArray	いいえ	いいえ	いいえ
readAsciiStream	いいえ	いいえ	いいえ
readBigDecimal	いいえ	いいえ	いいえ
readBinaryStream	いいえ	いいえ	いいえ
readBlob	いいえ	いいえ	いいえ
readBoolean	いいえ	いいえ	いいえ
readByte	いいえ	いいえ	いいえ
readBytes	いいえ	いいえ	いいえ
readCharacterStream	いいえ	いいえ	いいえ
readClob	いいえ	いいえ	いいえ
readDate	いいえ	いいえ	いいえ
readDouble	いいえ	いいえ	いいえ
readFloat	いいえ	いいえ	いいえ
readInt	いいえ	いいえ	いいえ
readLong	いいえ	いいえ	いいえ
readObject	いいえ	いいえ	いいえ

表 75. *SQLInput* メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
readRef	いいえ	いいえ	いいえ
readShort	いいえ	いいえ	いいえ
readString	いいえ	いいえ	いいえ
readTime	いいえ	いいえ	いいえ
readTimestamp	いいえ	いいえ	いいえ
wasNull	いいえ	いいえ	いいえ

表 76. *SQLIntegrityConstraintViolationException* メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 77. *SQLInvalidAuthorizationSpecException* メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 78. SQLNonTransientConnectionException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 79. SQLNonTransientException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 80. SQLOutput メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
writeArray	いいえ	いいえ	いいえ
writeAsciiStream	いいえ	いいえ	いいえ
writeBigDecimal	いいえ	いいえ	いいえ
writeBinaryStream	いいえ	いいえ	いいえ
writeBlob	いいえ	いいえ	いいえ
writeBoolean	いいえ	いいえ	いいえ
writeByte	いいえ	いいえ	いいえ
writeBytes	いいえ	いいえ	いいえ
writeCharacterStream	いいえ	いいえ	いいえ
writeClob	いいえ	いいえ	いいえ
writeDate	いいえ	いいえ	いいえ
writeDouble	いいえ	いいえ	いいえ

表 80. *SQLOutput* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
writeFloat	いいえ	いいえ	いいえ
writeInt	いいえ	いいえ	いいえ
writeLong	いいえ	いいえ	いいえ
writeObject	いいえ	いいえ	いいえ
writeRef	いいえ	いいえ	いいえ
writeShort	いいえ	いいえ	いいえ
writeString	いいえ	いいえ	いいえ
writeStruct	いいえ	いいえ	いいえ
writeTime	いいえ	いいえ	いいえ
writeTimestamp	いいえ	いいえ	いいえ

表 81. *SQLRecoverableException* メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 82. *SQLSyntaxErrorException* メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 83. SQLTimeoutException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 84. SQLTransientConnectionException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 85. SQLTransientException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 86. SQLTransientRollbackException メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
java.lang.Exception から継承されたメソッド	はい	いいえ	いいえ
java.lang.Throwable から継承されたメソッド	はい	いいえ	いいえ
java.lang.Object から継承されたメソッド	はい	いいえ	いいえ

注:

1. これは JDBC 4.0 クラスです。

表 87. SQLXML メソッドのサポート[1]

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
free	はい	いいえ	いいえ
getBinaryStream	はい	いいえ	いいえ
getCharacterStream	はい	いいえ	いいえ
getSource	はい	いいえ	いいえ
getString	はい	いいえ	いいえ
setBinaryStream	はい	いいえ	いいえ
setCharacterStream	はい	いいえ	いいえ
setResult	はい	いいえ	いいえ
setString	はい	いいえ	いいえ

注:

1. これらは JDBC 4.0 メソッドです。これらのメソッドは、 IBM Informix Dynamic Server サーバーとの接続ではサポートされていません。

表 88. Statement メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
addBatch	はい	はい	はい
cancel	はい[1]	はい[2]	はい
clearBatch	はい	はい	はい
clearWarnings	はい	はい	はい
close	はい	はい	はい
execute	はい	はい[3]	はい[3]
executeBatch	はい	はい	はい

表 88. Statement メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
executeQuery	はい	はい	はい
executeUpdate	はい	はい[3]	はい[3]
getConnection	はい	はい	はい
getFetchDirection	はい	はい	はい
getFetchSize	はい	はい	はい
getGeneratedKeys	はい	いいえ	いいえ
getMaxFieldSize	はい	はい	はい
getMaxRows	はい	はい	はい
getMoreResults	はい	はい[3]	はい[3]
getQueryTimeout	はい[2]	はい	はい
getResultSet	はい	はい	はい
getResultSetConcurrency	はい	はい	はい
getResultSetHoldability	はい	いいえ	いいえ
getResultSetType	はい	はい	はい
getUpdateCount[4]	はい	はい	はい
getWarnings	はい	はい	はい
isClosed[7]	はい	いいえ	いいえ
isPoolable[7]	はい	いいえ	いいえ
setCursorName	はい	はい	はい
setEscapeProcessing	はい	はい	はい
setFetchDirection	はい	はい	はい
setFetchSize	はい	はい	はい
setMaxFieldSize	はい	はい	はい
setMaxRows	はい	はい	はい
setPoolable[7]	はい	いいえ	いいえ
setQueryTimeout	はい[5、6]	はい	はい

表 88. *Statement* メソッドのサポート *(続き)*

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート

注:

1. IBM Data Server Driver for JDBC and SQLJ の場合、Statement.cancel() は以下の環境でのみサポートされます。
   - Linux、UNIX、または Windows クライアントから DB2 Database for Linux, UNIX, and Windows サーバー、バージョン 8 以降へのタイプ 2 またはタイプ 4 接続
   - Linux、UNIX、または Windows クライアントから DB2 for z/OS サーバー、バージョン 9 以降へのタイプ 2 またはタイプ 4 接続
   - z/OS クライアントから DB2 Database for Linux, UNIX, and Windows サーバー、バージョン 8 以降へのタイプ 4 接続
   - z/OS クライアントから DB2 for z/OS サーバー、バージョン 9 以降へのタイプ 4 接続
2. Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーの場合、Statement.cancel() は以下の環境でのみサポートされます。
   - DB2 Database for Linux, UNIX, and Windows サーバー、バージョン 8 以降への接続
   - DB2 for z/OS サーバー、バージョン 9 以降への接続
3. Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーは、このメソッドの JDBC 3.0 形式をサポートしません。
4. ストアード・プロシージャー ResultSet ではサポートされていません。
5. DB2 for i および DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続では、このメソッドは *seconds* の値が 0 の場合にのみサポートされます。
6. IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の場合、以下のメソッドに対して Statement.setQueryTimeout がサポートされます。
   - Statement.execute
   - Statement.executeUpdate
   - Statement.executeQuery

   Statement.setQueryTimeout は、Statement.executeBatch メソッドに対してはサポートされません。
7. これは JDBC 4.0 のメソッドです。

表 89. *Struct* メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getSQLTypeName	いいえ	いいえ	いいえ
getAttributes	いいえ	いいえ	いいえ

表 90. *Wrapper* メソッドのサポート

JDBC メソッド[1]	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
isWrapperFor	はい	いいえ	いいえ

表 90. Wrapper メソッドのサポート (続き)

JDBC メソッド[1]	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
unWrap	はい	いいえ	いいえ

注:

1. これらは JDBC 4.0 メソッドです。

表 91. javax.sql.XAConnection メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート[1]	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
javax.sql.PooledConnection から継承されたメソッド	はい	はい	はい
getXAResource	はい	はい	はい

注:

1. これらのメソッドは、 DB2 Database for Linux, UNIX, and Windows サーバーへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続、または IBM Data Server Driver for JDBC and SQLJ Type 4 接続でサポートされます。

表 92. XADataSource メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
getLoginTimeout	はい	はい	はい
getLogWriter	はい	はい	はい
getXAConnection	はい	はい	はい
setLoginTimeout	はい	はい	はい
setLogWriter	はい	はい	はい

表 93. javax.transaction.xa.XAResource メソッドのサポート

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
commit	はい[1]	はい	はい
end	はい[1]	はい	はい
forget	はい[1]	はい	はい
getTransactionTimeout	はい[2]	はい	はい
isSameRM	はい[1]	はい	はい
prepare	はい[1]	はい	はい
recover	はい[1]	はい	はい

表 93. *javax.transaction.xa.XAResource* メソッドのサポート (続き)

JDBC メソッド	IBM Data Server Driver for JDBC and SQLJ のサポート	Linux 、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバーのサポート	IDS JDBC ドライバーのサポート
rollback	はい[1]	はい	はい
setTransactionTimeout	はい[2]	はい	はい
start	はい[1]	はい	はい

注:

1. このメソッドは、 DB2 Database for Linux, UNIX, and Windows サーバーへの IBM Data Server Driver for JDBC and SQLJ Type 2 接続、または IBM Data Server Driver for JDBC and SQLJ Type 4 接続 でサポートされます。
2. このメソッドは、 DB2 Database for Linux, UNIX, and Windows バージョン 9.1 以降に対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続 でサポートされています。

SQLJ ステートメント参照情報

SQLJ ステートメントは、トランザクション制御および SQL ステートメント実行に使用されます。

SQLJ 節

SQLJ プログラム内の SQL ステートメントは SQLJ 節です。

構文

使用上の注意

SQLJ 節内のキーワードは、それらのキーワードが実行可能節内の SQL ステートメントの一部でない限り、大文字と小文字を区別します。

SQLJ ホスト式

ホスト式とは、SQLJ アプリケーション・プログラムの SQLJ 節によって参照される Java 変数または式です。

構文

説明

: 後に続く変数または式がホスト式であることを示します。コロンは変数または式の直前に付ける必要があります。

IN|OUT|INOUT
ストアード・プロシージャー呼び出しでパラメーターとして使用されるホスト式の場合、パラメーターがストアード・プロシージャーにデータを提供するか (IN)、ストアード・プロシージャーからデータを取り出すか (OUT)、またはその両方を行うか (INOUT) を識別します。デフォルトは、IN です。

simple-variable
Java 非修飾 ID を指定します。

complex-expression
単一値を生成する Java 式を指定します。

使用上の注意

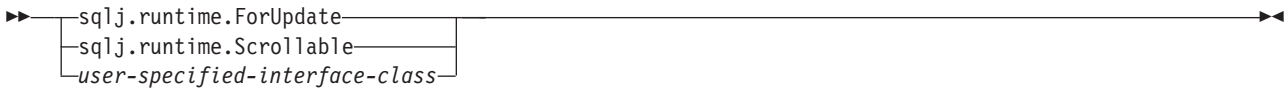
- 複合式は括弧で囲む必要があります。
- ANSI/ISO 規則は、ホスト式を静的 SQL ステートメントに配置できる場所で適用されます。

SQLJ インプリメント節

インプリメント節は、Java インターフェースの 1 つ以上のクラスから派生したものです。

構文

interface-element:



説明

interface-element
ユーザー定義 Java インターフェース、SQLJ インターフェース sqlj.runtime.ForUpdate、または SQLJ インターフェース sqlj.runtime.Scrollable を指定します。

位置指定 UPDATE または位置指定 DELETE 操作用にイテレーターを宣言する際に、sqlj.runtime.ForUpdate をインプリメントする必要があります。 SQLJ での位置指定 UPDATE または位置指定 DELETE 操作の実行について詳しくは、『SQLJ アプリケーションでの位置指定 UPDATE および DELETE 操作の実行』を参照してください。

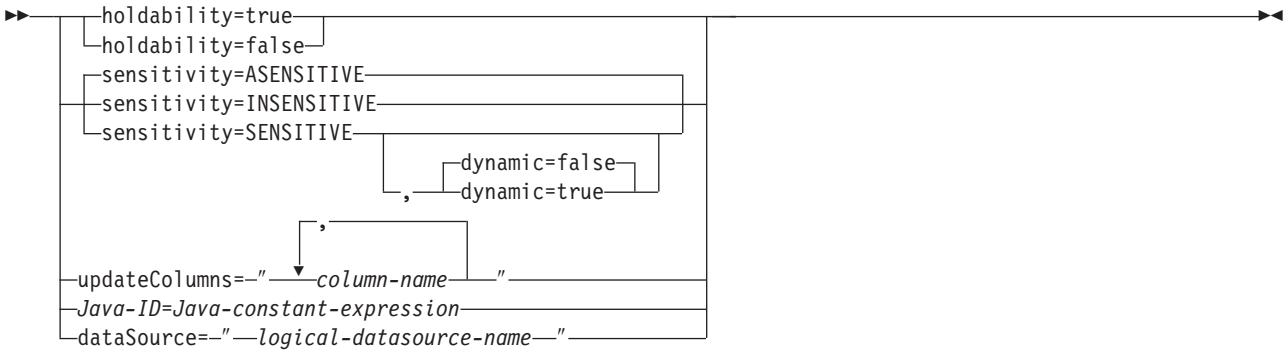
スクロール可能イテレーターを宣言する際に、sqlj.runtime.Scrollable をインプリメントする必要があります。スクロール可能イテレーターについて詳しくは、『SQLJ アプリケーションでのスクロール可能イテレーターの使用』を参照してください。

SQLJ with 節

with 節は、イテレーターまたは接続コンテキストの 1 つ以上の属性のセットを指定します。

構文

with-element:



説明

holdability

イテレーターの場合、COMMIT が実行された後、イテレーターが表内でその位置を保持するかどうかを指定します。 holdability の値は `true` または `false` でなければなりません。

sensitivity

イテレーターの場合、基礎表に対して行われる変更が、イテレーターが開かれた後にそのイテレーターから可視になるかどうかを指定します。値は、INSENSITIVE、SENSITIVE、または ASENSITIVE でなければなりません。デフォルトは ASENSITIVE です。

IBM Informix Dynamic Server (IDS) との接続の場合は、INSENSITIVE のみサポートされます。

dynamic

sensitivity=SENSITIVE で定義されたイテレーターの場合、以下のケースが該当するかどうかを指定します。

- アプリケーションが位置指定 UPDATE および DELETE ステートメントをイテレーターで実行する場合、それらの変更がイテレーターから可視になる。
- アプリケーションが、そのアプリケーションの内部ではあるがイテレーターの外部で INSERT、UPDATE、および DELETE ステートメントを実行する場合、それらの変更がイテレーターから可視になる。

dynamic の値は `true` または `false` でなければなりません。デフォルトは `false` です。

DB2 Database for Linux, UNIX, and Windows サーバーは動的両方向スクロール・カーソルをサポートしません。アプリケーションが DB2 for z/OS サーバーのバージョン 9 以降でデータにアクセスする場合にのみ、`true` を指定します。

IDS との接続の場合は、`false` のみサポートされます。 IDS において、動的カーソルはサポートされていません。

updateColumns
イテレーターの場合、イテレーターが位置指定 UPDATE ステートメントに使用される際に変更される列を指定します。 updateColumns の値は、コンマで区切られた列名を含むリテラル・ストリングでなければなりません。

column-name
イテレーターの場合、イテレーターを使用して更新される結果表の列を指定します。

Java-ID
イテレーターまたは接続コンテキストの場合、イテレーターまたは接続コンテキストのユーザー定義属性を識別する Java 変数を指定します。
Java-constant-expression の値もユーザー定義です。

dataSource
接続コンテキストの場合、アプリケーションの接続先のデータ・ソースを表す、個別に作成された DataSource オブジェクトの論理名を指定します。このオプションは、 IBM Data Server Driver for JDBC and SQLJ にのみ使用可能です。

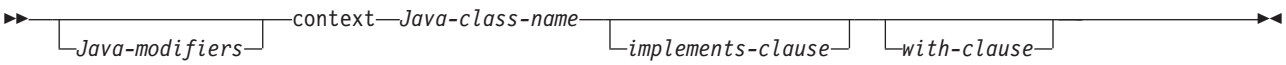
使用上の注意

- with エレメントの左側の値はその with 節内で固有でなければなりません。
- イテレーター宣言節の with エレメントで updateColumns を指定する場合、イテレーター宣言節には、sqlj.runtime.ForUpdate インターフェースを指定する implements 節も含まれていなければなりません。
- SQLJ プログラムをカスタマイズしない場合、JDBC ドライバーは with 節内の holdability の値を無視します。その代わりに、ドライバーは holdability に JDBC ドライバー設定を使用します。

SQLJ 接続宣言節

接続宣言節は、SQLJ アプリケーション・プログラム内のデータ・ソースへの接続を宣言します。

構文



説明

Java-modifiers
Java クラス宣言に有効な修飾子 (static、public、private、または protected など) を指定します。

Java-class-name
有効な Java ID を指定します。プログラム準備処理において、SQLJ は名前がこの ID である接続コンテキスト・クラスを生成します。

implements-clause
このクラスの説明については、『SQLJ インプリメント節』を参照してください。接続宣言節では、インプリメント節が参照するインターフェース・クラスはユーザー定義インターフェース・クラスでなければなりません。

with-clause
このクラスの説明については、『SQLJ with 節』を参照してください。

使用上の注意

- SQLJ は、指定される接続宣言節ごとに接続クラス宣言を生成します。 SQLJ データ・ソース接続は、生成された接続クラスのオブジェクトです。
- Java クラス定義を Java プログラム内で配置できる場所であればどこでも接続宣言節を指定できます。

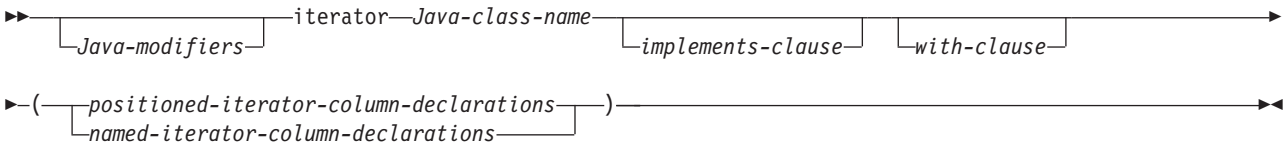
SQLJ イテレーター宣言節

イテレーター宣言節は、SQLJ アプリケーション・プログラムで位置指定イテレーター・クラスまたは名前指定イテレーター・クラスを宣言します。

イテレーターには照会からの結果表が含まれます。 SQLJ は、指定されたイテレーター宣言節ごとにイテレーター・クラスを生成します。イテレーターは、イテレーター・クラスのオブジェクトです。

イテレーター宣言節には、位置指定イテレーターの形式と名前指定イテレーターの形式があります。 2 種類のイテレーターは固有であり、異なるインターフェースでインプリメントされている Java タイプとは互換性がありません。

構文



positioned-iterator-column declarations:

named-iterator-column-declarations:

説明

Java-modifiers
Java クラス宣言に有効な修飾子 (static、public、private、または protected など)。

Java-class-name
有効な Java ID。プログラム準備処理中に、SQLJ は名前がこの ID であるイテレーター・クラスを生成します。

implements-clause
このクラスの説明については、『SQLJ インプリメント節』を参照してください。位置指定 UPDATE または位置指定 DELETE 操作用のイテレーターを宣言するイテレーター宣言節の場合、インプリメント節はインターフェース sqlj.runtime.ForUpdate を指定する必要があります。スクロール可能イテレーターを宣言するイテレーター宣言節の場合、インプリメント節はインターフェース sqlj.runtime.Scrollable を指定する必要があります。

with-clause
このクラスの説明については、『SQLJ with 節』を参照してください。

positioned-iterator-column-declarations
Java データ・タイプのリストを指定します。これは、位置指定イテレーター内の列のデータ・タイプです。リスト内のデータ・タイプはコンマで区切る必要があります。位置指定イテレーター宣言内のデータ・タイプの順序は、結果表内の列の順序と同じでなければなりません。シリアライズド・プロファイル・カスタマイズ中のオンライン検査が成功するには、イテレーター内の列のデータ・タイプは結果表内の列のデータ・タイプと互換性がなければなりません。互換性のあるデータ・タイプのリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

named-iterator-column-declarations
Java データ・タイプおよび Java ID のリストを指定します。これらは、名前指定イテレーター内の列のデータ・タイプおよび名前です。データ・タイプと名前の対はコンマで区切る必要があります。イテレーター内の列の名前は、大/小文字の区別は別として、結果表の列の名前と一致しなければなりません。シリアライズド・プロファイル・カスタマイズ中のオンライン検査が成功するには、イテレーター内の列のデータ・タイプは結果表内の列のデータ・タイプと互換性がなければなりません。互換性のあるデータ・タイプのリストについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。

使用上の注意

- イテレーター宣言節は、Java クラス宣言を配置できる場所であれば、Java プログラム内のどこにでも配置できます。
- 名前指定イテレーター宣言に Java データ・タイプと Java ID の複数の対が含まれている場合、リスト内のすべての Java ID は固有でなければなりません。2 つの Java ID の大/小文字の区別だけが異なる場合は、それらは固有ではありません。

SQLJ 実行可能節

実行可能節には SQL ステートメントまたは代入ステートメントが含まれます。代入ステートメントは、SQL 操作の結果を Java 変数に割り当てます。

このトピックでは、実行可能節の一般形式について説明します。

構文

使用上の注意

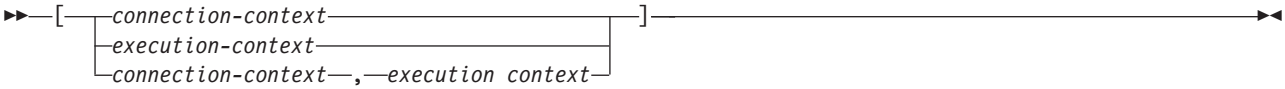
- 実行可能節は、Java ステートメントを配置できる場所であれば、Java プログラム内のどこにでも配置できます。
- SQLJ は、クラス java.sql.SQLException を使用して実行可能節からの負の SQL コードを報告します。

  SQLJ で実行可能節の実行中にランタイム例外が発生する場合、タイプ OUT または INOUT のホスト式の値が未定義です。

SQLJ コンテキスト節

コンテキスト節は、接続コンテキスト、実行コンテキスト、またはその両方を指定します。接続コンテキストを使用して、データ・ソースに接続します。実行コンテキストを使用して、SQL ステートメントの実行をモニターおよび変更します。

構文



説明

connection-context
SQLJ プログラムで以前に宣言されている有効な Java ID を指定します。その ID は、SQLJ が接続宣言節のために生成する接続コンテキスト・クラスのインスタンスとして宣言する必要があります。

execution-context
SQLJ プログラムで以前に宣言されている有効な Java ID を指定します。その ID は、クラス sqlj.runtime.ExecutionContext のインスタンスとして宣言する必要があります。

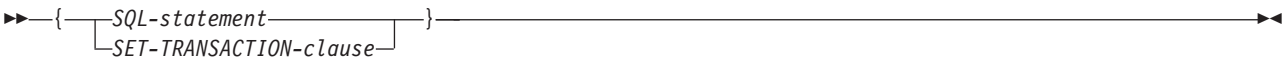
使用上の注意

- 実行可能節で接続コンテキストを指定しない場合、SQLJ はデフォルト接続コンテキストを使用します。
- 実行コンテキストを指定しない場合、SQLJ はステートメントの接続コンテキストから実行コンテキストを取得します。

SQLJ ステートメント節

ステートメント節には SQL ステートメントまたは SET TRANSACTION 節が含まれます。

構文



説明

SQL-statement
表 94 の SQL ステートメントをステートメント節に組み込むことができます。

SET-TRANSACTION-clause
プログラム内の SQL ステートメントの分離レベルおよび接続用のアクセス・モードを設定します。 SET TRANSACTION 節は SET TRANSACTION ステートメントに相当します。これは、1992 年の ANSI/ISO SQL 標準で記述されており、SQL の一部のインプリメンテーションでサポートされます。

表 94. SQLJ ステートメント節内の有効な SQL ステートメント

ステートメント	適用できるデータ・ソース
ALTER DATABASE	1 (394 ページ), 2 (394 ページ)
ALTER FUNCTION	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
ALTER INDEX	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
ALTER PROCEDURE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
ALTER STOGROUP	1 (394 ページ), 2 (394 ページ)
ALTER TABLE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
ALTER TABLESPACE	1 (394 ページ), 2 (394 ページ)
CALL	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
COMMENT ON	1 (394 ページ), 2 (394 ページ)
COMMIT	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
コンパウンド SQL (BEGIN ATOMIC...END)	2 (394 ページ)
CREATE ALIAS	1 (394 ページ), 2 (394 ページ)
CREATE DATABASE	1 (394 ページ), 2 (394 ページ), 3a (394 ページ)
CREATE DISTINCT TYPE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE FUNCTION	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE GLOBAL TEMPORARY TABLE	1 (394 ページ), 2 (394 ページ)
CREATE TEMP TABLE	3 (394 ページ)
CREATE INDEX	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE PROCEDURE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE STOGROUP	1 (394 ページ), 2 (394 ページ)
CREATE SYNONYM	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE TABLE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE TABLESPACE	1 (394 ページ), 2 (394 ページ)
CREATE TYPE (カーソル)	2 (394 ページ)

表 94. SQLJ ステートメント節内の有効な SQL ステートメント (続き)

ステートメント	適用できるデータ・ソース
CREATE TRIGGER	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
CREATE VIEW	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DECLARE GLOBAL TEMPORARY TABLE	1 (394 ページ), 2 (394 ページ)
DELETE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP ALIAS	1 (394 ページ), 2 (394 ページ)
DROP DATABASE	1 (394 ページ), 2 (394 ページ), 3a (394 ページ)
DROP DISTINCT TYPE	1 (394 ページ), 2 (394 ページ)
DROP TYPE	3 (394 ページ)
DROP FUNCTION	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP INDEX	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP PACKAGE	1 (394 ページ), 2 (394 ページ)
DROP PROCEDURE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP STOGROUP	1 (394 ページ), 2 (394 ページ)
DROP SYNONYM	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP TABLE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP TABLESPACE	1 (394 ページ), 2 (394 ページ)
DROP TRIGGER	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
DROP VIEW	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
FETCH	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
GRANT	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
INSERT	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
LOCK TABLE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
MERGE	1 (394 ページ), 2 (394 ページ)
REVOKE	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
ROLLBACK	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
SAVEPOINT	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
SELECT INTO	1 (394 ページ), 2 (394 ページ), 3 (394 ページ)
SET CURRENT APPLICATION ENCODING SCHEME	1 (394 ページ)
SET CURRENT DEBUG MODE	1 (394 ページ)
SET CURRENT DEFAULT TRANSFORM GROUP	2 (394 ページ)
SET CURRENT DEGREE	1 (394 ページ), 2 (394 ページ)
SET CURRENT EXPLAIN MODE	2 (394 ページ)
SET CURRENT EXPLAIN SNAPSHOT	2 (394 ページ)
SET CURRENT ISOLATION	1 (394 ページ), 2 (394 ページ)
SET CURRENT LOCALE LC_CTYPE	1 (394 ページ)
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	1 (394 ページ), 2 (394 ページ)
SET CURRENT OPTIMIZATION HINT	1 (394 ページ), 2 (394 ページ)
SET CURRENT PACKAGE PATH	1 (394 ページ)

表 94. SQLJ ステートメント節内の有効な SQL ステートメント (続き)

ステートメント	適用できるデータ・ソース
SET CURRENT PACKAGESET (USER はサポートされない)	1, 2
SET CURRENT PRECISION	1, 2
SET CURRENT QUERY OPTIMIZATION	2
SET CURRENT REFRESH AGE	1, 2
SET CURRENT ROUTINE VERSION	1
SET CURRENT RULES	1
SET CURRENT SCHEMA	2
SET CURRENT SQLID	1
SET PATH	1, 2
TRUNCATE	1
UPDATE	1, 2, 3

注: SQL ステートメントは以下のデータ・ソースの接続に適用されます。

1. DB2 for z/OS
2. DB2 Database for Linux, UNIX, and Windows
3. IBM Informix Dynamic Server
   a. IBM Informix Dynamic Server (SYSMASTER データベースの場合のみ)

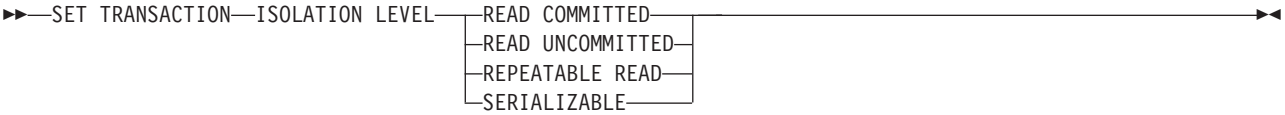
使用上の注意

- SQLJ は位置指定 DELETE および UPDATE 操作と検索の DELETE および UPDATE 操作の両方をサポートします。
- FETCH ステートメント、位置指定 DELETE ステートメント、または位置指定 UPDATE ステートメントの場合、イテレーターを使用して結果表内の行を参照する必要があります。

SQLJ SET TRANSACTION 節

SET TRANSACTION 節は、現在の作業単位の分離レベルを設定するものです。

構文



説明

ISOLATION LEVEL
以下の分離レベルの 1 つを指定します。

READ COMMITTED
現在の DB2 分離レベルをカーソル固定にすることを指定します。

READ UNCOMMITTED
現在の DB2 分離レベルを非コミット読み取りにすることを指定します。

REPEATABLE READ
現在の DB2 分離レベルを読み取り固定にすることを指定します。

SERIALIZABLE
現在の DB2 分離レベルを反復可能読み取りにすることを指定します。

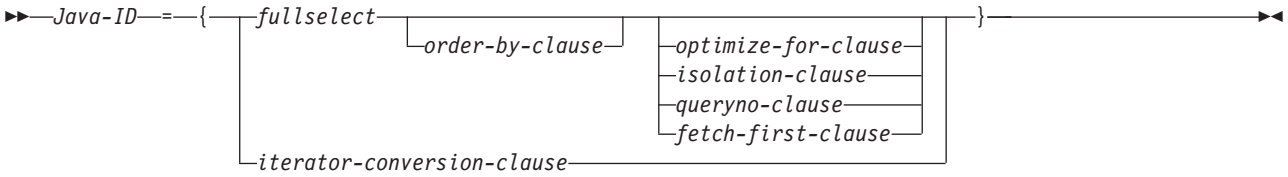
使用上の注意

SET TRANSACTION はトランザクションの開始時にのみ実行できます。

SQLJ 代入節

代入節は、SQL 操作の結果を Java 変数に割り当てます。

構文



説明

Java-ID
イテレーター・クラスのインスタンスとして以前に宣言されたイテレーターを識別します。

fullselect
結果表を生成します。

iterator-conversion-clause
このクラスの説明については、『SQLJ iterator-conversion-clause』を参照してください。

使用上の注意

- *Java-ID* によって識別されるオブジェクトが位置指定イテレーターの場合、結果セット内の列の数がイテレーター内の列の数に一致しなければなりません。さらに、結果セット内の列ごとのデータ・タイプがイテレーター内の対応する列のデータ・タイプと互換性を持っている必要があります。互換性のある Java および SQL データ・タイプについては、『Java、JDBC、および SQL のデータ・タイプ』を参照してください。
- *Java-ID* によって識別されるオブジェクトが名前指定イテレーターの場合、各 accessor メソッドの名前が、大/小文字の区別は別として、結果セット内の列の名前に一致しなければなりません。さらに、accessor メソッドが戻すオブジェクトのデータ・タイプが結果セット内の対応する列のデータ・タイプと互換性を持っている必要があります。
- Java プログラム内の、Java 代入ステートメントを配置できる場所であれば、どこでも代入節を入れることができます。ただし、Java 代入式を配置できる場所には、代入節を入れることはできません。例えば、FOR ステートメントの制御リストでは代入節を指定できません。

SQLJ イテレーター変換節

イテレーター変換節は、JDBC ResultSet をイテレーターに変換します。

構文

説明

host-expression
SQLJ イテレーターに変換される JDBC ResultSet を指定します。

使用上の注意

- JDBC ResultSet が変換されるイテレーターが位置指定イテレーターの場合、ResultSet 内の列の数はイテレーター内の列の数に一致しなければなりません。さらに、ResultSet 内の各列のデータ・タイプがイテレーター内の対応する列のデータ・タイプと互換性を持っている必要があります。
- イテレーターが名前指定イテレーターの場合、各 accessor メソッドの名前が、大/小文字の区別は別として、ResultSet 内の列の名前に一致しなければなりません。さらに、accessor メソッドが戻すオブジェクトのデータ・タイプが ResultSet 内の対応する列のデータ・タイプと互換性を持っている必要があります。
- イテレーター変換節によって生成されたイテレーターが閉じられると、そのイテレーターの生成元の ResultSet も閉じられます。

sqlj.runtime パッケージ内のインターフェースおよびクラス

sqlj.runtime パッケージは、SQLJ プログラマーにより直接または間接に使用されるランタイム・クラスおよびインターフェースを定義します。

AsciiStream などのクラスは、SQLJ プログラマーにより直接使用されます。ResultSetIterator などのインターフェースは、生成されたクラス宣言の一部としてインプリメントされます。

sqlj.runtime インターフェース

以下の表では、sqlj.runtime のインターフェースを要約しています。

表 95. sqlj.runtime インターフェースのサマリー

インターフェースの名前	目的
ConnectionContext	データ・ソースへの接続中に実行される SQL 操作を管理します。
ForUpdate	位置指定 UPDATE または DELETE ステートメントで使用されるイテレーターによりインプリメントされます。
NamedIterator	名前指定イテレーターとして宣言されるイテレーターによりインプリメントされます。
PositionedIterator	位置指定イテレーターとして宣言されるイテレーターによりインプリメントされます。
ResultSetIterator	照会結果を JDBC ResultSet を使用して処理することを許可するすべてのイテレーターによりインプリメントされます。

表 95. *sqlj.runtime* インターフェースのサマリー *(続き)*

インターフェースの名前	目的
Scrollable	スクロール可能イテレーターの操作用の一連のメソッドを提供します。

sqlj.runtime クラス

以下の表では、sqlj.runtime のクラスを要約しています。

表 96. *sqlj.runtime* クラスのサマリー

クラス名	目的
AsciiStream	バイトが ASCII として解釈される入力ストリームの処理のためのクラス。
BinaryStream	バイトがバイナリーとして解釈される入力ストリームの処理のためのクラス。
CharacterStream	バイトが文字として解釈される入力ストリームの処理のためのクラス。
DefaultRuntime	ほとんどの JVM 環境で SQLJ に期待される実行時動作を満たすために、SQLJ によりインプリメントされます。このクラスは内部使用専用であり、この資料では説明されません。
ExecutionContext	SQL 操作の実行を制御するために、SQLJ 実行コンテキストの宣言時にインプリメントされます。
RuntimeContext	実行時環境で提供されるシステム固有サービスを定義します。このクラスは内部使用専用であり、この資料では説明されません。
SQLNullException	java.sql.SQLException クラスから派生します。 SQL NULL 値がフェッチされ、ホスト ID に Java プリミティブ・タイプで入れられると、sqlj.runtime.SQLNullException がスローされます。
StreamWrapper	java.io.InputStream インスタンスをラップします。
UnicodeStream	バイトが Unicode として解釈される入力ストリームの処理のためのクラス。

sqlj.runtime.ConnectionContext インターフェース

sqlj.runtime.ConnectionContext インターフェースは、特定のデータ・ソースとのセッション中に実行される SQL 操作を管理するメソッドのセットを提供します。

SQLJ 接続宣言節を変換すると、SQLJ によって接続コンテキスト・クラスが作成されます。接続コンテキスト・オブジェクトは、動的 SQL 操作を実行できる JDBC Connection オブジェクトを保守します。接続コンテキスト・オブジェクトはデフォルトの ExecutionContext オブジェクトも保守します。

変数

CLOSE_CONNECTION

形式:

```
public static final boolean CLOSE_CONNECTION=true;
```

close メソッドに渡すことができる定数。これは、基礎となる JDBC Connection オブジェクトをクローズする必要があることを示します。

KEEP_CONNECTION

形式:

```
public static final boolean KEEP_CONNECTION=false;
```

close メソッドに渡すことができる定数。これは、基礎となる JDBC Connection オブジェクトをクローズしてはならないことを示します。

メソッド

close()

形式:

```
public abstract void close() throws SQLException
```

以下の機能を実行します。

- 指定された接続コンテキスト・オブジェクトによって使用されるすべてのリソースを解放する。
- オープンしている ConnectedProfile オブジェクトをクローズする。
- 基礎となる JDBC Connection オブジェクトをクローズする。

close() は close(CLOSE_CONNECTION) と同等です。

close(boolean)

形式:

```
public abstract void close (boolean close-connection)
 throws SQLException
```

以下の機能を実行します。

- 指定された接続コンテキスト・オブジェクトによって使用されるすべてのリソースを解放する。
- オープンしている ConnectedProfile オブジェクトをクローズする。
- *close-connection* パラメーターの値に従って、基礎となる JDBC Connection オブジェクトをクローズする。

パラメーターは以下のとおりです。

close-connection
接続コンテキスト・オブジェクトがクローズされるときに基礎となる JDBC Connection オブジェクトがクローズされるかどうかを指定します。

CLOSE_CONNECTION
基礎となる JDBC Connection オブジェクトをクローズします。

KEEP_CONNECTION
基礎となる JDBC Connection オブジェクトをクローズしません。

getConnectedProfile

形式:

```
public abstract ConnectedProfile getConnectedProfile(Object profileKey)
 throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

getConnection

形式:

```
public abstract Connection getConnection()
```

指定された接続コンテキスト・オブジェクトの基礎となる JDBC Connection オブジェクトを戻します。

getExecutionContext

形式:

```
public abstract ExecutionContext getExecutionContect()
```

指定された接続コンテキスト・オブジェクトに関連したデフォルトの ExecutionContext オブジェクトを戻します。

isClosed

形式:

```
public abstract boolean isClosed()
```

指定された接続コンテキスト・オブジェクトがクローズされている場合、`true` を戻します。接続コンテキスト・オブジェクトがクローズされていない場合、`false` を戻します。

コンストラクター

以下のコンストラクターは、ステートメント #sql context Ctx; の変換の結果である `ConnectionContext` インターフェースの具体的なインプリメンテーションで定義されます。

Ctx(String, boolean)

形式:

```
public Ctx(String url, boolean autocommit)
 throws SQLException
```

パラメーターは以下のとおりです。

url JDBC getConnection メソッドで指定されたデータ・ソースの表記。

自動コミット
接続の自動コミットが有効であるかどうかを示します。値 `true` は自動コミットが有効であることを意味します。値 `false` は自動コミットが無効であることを意味します。

Ctx(String, String, String, boolean)

形式:

```
public Ctx(String url, String user, String password,
 boolean autocommit)
 throws SQLException
```

パラメーターは以下のとおりです。

url JDBC getConnection メソッドで指定されたデータ・ソースの表記。

user
データ・ソースへの接続を行うためのユーザー ID。

password
データ・ソースへの接続を行うためのユーザー ID のパスワード。

自動コミット
接続の自動コミットが有効であるかどうかを示します。値 `true` は自動コミットが有効であることを意味します。値 `false` は自動コミットが無効であることを意味します。

Ctx(String, Properties, boolean)

形式:

```
public Ctx(String url, Properties info, boolean autocommit)
 throws SQLException
```

パラメーターは以下のとおりです。

url JDBC getConnection メソッドで指定されたデータ・ソースの表記。

info
接続のドライバー・プロパティーのセットを含むオブジェクト。任意の IBM Data Server Driver for JDBC and SQLJ プロパティーを指定できます。

自動コミット
接続の自動コミットが有効であるかどうかを示します。値 `true` は自動コミットが有効であることを意味します。値 `false` は自動コミットが無効であることを意味します。

Ctx(Connection)

形式:

```
public Ctx(java.sql.Connection JDBC-connection-object)
 throws SQLException
```

パラメーターは以下のとおりです。

JDBC-connection-object
以前に作成された JDBC Connection オブジェクト。

コンストラクター呼び出しが SQLException をスローする場合、JDBC Connection オブジェクトはオープンしたままです。

Ctx(ConnectionContext)

形式:

```
public Ctx(sqlj.runtime.ConnectionContext SQLJ-connection-context-object)
 throws SQLException
```

パラメーターは以下のとおりです。

SQLJ-connection-context-object
以前に作成された SQLJ ConnectionContext オブジェクト。

以下のコンストラクターは、ステートメント #sql context Ctx with (dataSource ="jdbc/TestDS"); の変換の結果である `ConnectionContext` インターフェースの具体的なインプリメンテーションで定義されます。

Ctx()

形式:

```
public Ctx()
 throws SQLException
```

Ctx(String, String)

形式:

```
public Ctx(String user, String password,
)
 throws SQLException
```

パラメーターは以下のとおりです。

user
データ・ソースへの接続を行うためのユーザー ID。

password
データ・ソースへの接続を行うためのユーザー ID のパスワード。

Ctx(Connection)

形式:

```
public Ctx(java.sql.Connection JDBC-connection-object)
 throws SQLException
```

パラメーターは以下のとおりです。

JDBC-connection-object
以前に作成された JDBC Connection オブジェクト。

コンストラクター呼び出しが SQLException をスローする場合、JDBC Connection オブジェクトはオープンしたままです。

Ctx(ConnectionContext)

形式:

```
public Ctx(sqlj.runtime.ConnectionContext SQLJ-connection-context-object)
 throws SQLException
```

パラメーターは以下のとおりです。

SQLJ-connection-context-object
以前に作成された SQLJ ConnectionContext オブジェクト。

メソッド

以下の追加メソッドは、ステートメント #sql context Ctx; の変換の結果である ConnectionContext インターフェースの具体的なインプリメンテーションで生成されます。

getDefaultContext

形式:

```
public static Ctx getDefaultContext()
```

Ctx クラスのデフォルトの接続コンテキスト・オブジェクトを戻します。

getProfileKey

形式:

```
public static Object getProfileKey(sqlj.runtime.profile.Loader loader,
String profileName) throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

getProfile

形式:

```
public static sqlj.runtime.profile.Profile getProfile(Object key)
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

getTypeMap

形式:

```
public static java.util.Map getTypeMap()
```

ConnectionContext に関連したユーザー定義タイプのマップである java.util.Map をインプリメントするクラスのインスタンスを戻します。関連したタイプのマップがない場合、Java NULL が戻されます。

このメソッドは、実行可能節およびイテレーター宣言節用に SQLJ 変換プログラムによって生成されるコードで使用されますが、JDBC ステートメントで直接使用するために SQLJ アプリケーションで呼び出すこともできます。

SetDefaultContext

形式:

```
public static void Ctx setDefaultContext(Ctx default-context)
```

Ctx クラスのデフォルトの接続コンテキスト・オブジェクトを設定します。

推奨: このメソッドをマルチスレッド・アプリケーションに使用しないでください。代わりに、明示的コンテキストを使用してください。

sqlj.runtime.ForUpdate インターフェース

SQLJ は sqlj.runtime.ForUpdate インターフェースを `implements sqlj.runtime.ForUpdate` を指定したイテレーター宣言節を含む SQLJ プログラムでインプリメントします。

位置指定 UPDATE または DELETE 操作 (UPDATE...WHERE CURRENT OF または DELETE...WHERE CURRENT OF) を実行する SQLJ プログラムは、`implements sqlj.runtime.ForUpdate` を指定したイテレーター宣言節を含んでいる必要があります。

メソッド

getCursorName

形式:

```
public abstract String getCursorName() throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

sqlj.runtime.NamedIterator インターフェース

sqlj.runtime.NamedIterator インターフェースは、SQLJ アプリケーションが名前指定イテレーターのイテレーター宣言節を実行するときにインプリメントされます。

名前指定イテレーターには結果表の列名が組み込まれ、イテレーター内の列の順序は重要ではありません。

sqlj.runtime.NamedIterator インターフェースのインプリメンテーションには、結果表の各列の accessor メソッドが組み込まれます。 accessor メソッドは、結果表の列からのデータを戻します。 accessor メソッドの名前は、名前指定イテレーターの対応する列の名前と一致します。

メソッド (ResultSetIterator インターフェースから継承)

close

形式:

```
public abstract void close() throws SQLException
```

イテレーターが使用するデータベース・リソースを解放します。

isClosed

形式:

```
public abstract boolean isClosed() throws SQLException
```

close メソッドが呼び出されている場合に、`true` の値を戻します。 close メソッドが呼び出されていない場合には、`false` を戻します。

next

形式:

```
public abstract boolean next() throws SQLException
```

イテレーターを次の行に進めます。 next メソッドのインスタンスを最初に呼び出す前に、イテレーターは結果表の最初の行の前に置かれます。 next は、次の行が使用可能な場合は `true` の値を、すべての行が検索済みの場合は `false` の値を戻します。

sqlj.runtime.PositionedIterator インターフェース

sqlj.runtime.PositionedIterator インターフェースは、SQLJ アプリケーションが位置指定イテレーターのイテレーター宣言節を実行するときにインプリメントされます。

位置指定イテレーター内の列の順序は、結果表内の列の順序と同じでなければならず、位置指定イテレーターには結果表の列名は組み込まれません。

メソッド

sqlj.runtime.PositionedIterator はすべての **ResultSetIterator** メソッドを継承し、以下の追加のメソッドを組み込みます。

endFetch

形式:

```
public abstract boolean endFetch() throws SQLException
```

イテレーターが行上に置かれていない場合は、true の値を戻します。イテレーターが行上に置かれている場合は、false の値を戻します。

sqlj.runtime.ResultSetIterator インターフェース

sqlj.runtime.ResultSetIterator インターフェースは、すべてのイテレーター宣言節に対して SQLJ によってインプリメントされます。

型なしイテレーターは、sqlj.runtime.ResultSetIterator インターフェースのインスタンスの宣言により直接生成できます。一般には、型なしイテレーターの使用は推奨されていません。

変数

ASENSITIVE

形式:

```
public static final int ASENSITIVE
```

getSensitivity メソッドにより戻すことができる定数。これはイテレーターが ASENSITIVE として定義されていることを示します。

この値は IBM Informix Dynamic Server によって戻されません。

FETCH_FORWARD

形式:

```
public static final int FETCH_FORWARD
```

以下のメソッドにより使用できる定数。

- sqlj.runtime.Scrollable.setFetchDirection および sqlj.runtime.ExecutionContext.setFetchDirection により設定される
- sqlj.runtime.ExecutionContext.getFetchDirection により戻される

これはイテレーターが結果表内の行を、先頭から最後までの順方向でフェッチすることを示します。

FETCH_REVERSE

形式:

```
public static final int FETCH_REVERSE
```

以下のメソッドにより使用できる定数。

- sqlj.runtime.Scrollable.setFetchDirection および sqlj.runtime.ExecutionContext.setFetchDirection により設定される
- sqlj.runtime.ExecutionContext.getFetchDirection により戻される

これはイテレーターが結果表内の行を、最後から先頭までの逆方向でフェッチすることを示します。

この値は IBM Informix Dynamic Server によって戻されません。

FETCH_UNKNOWN

形式:

```
public static final int FETCH_UNKNOWN
```

以下のメソッドにより使用できる定数。

- sqlj.runtime.Scrollable.setFetchDirection および sqlj.runtime.ExecutionContext.setFetchDirection により設定される
- sqlj.runtime.ExecutionContext.getFetchDirection により戻される

これはイテレーターが結果表内の行を、不明の順序でフェッチすることを示します。

この値は IBM Informix Dynamic Server によって戻されません。

INSENSITIVE

形式:

```
public static final int INSENSITIVE
```

getSensitivity メソッドにより戻すことができる定数。これはイテレーターが INSENSITIVE として定義されていることを示します。

SENSITIVE

形式:

```
public static final int SENSITIVE
```

getSensitivity メソッドにより戻すことができる定数。これはイテレーターが SENSITIVE として定義されていることを示します。

この値は IBM Informix Dynamic Server によって戻されません。

メソッド

clearWarnings

形式:

```
public abstract void clearWarnings() throws SQLException
```

clearWarnings の呼び出し後に、getWarnings は新しい警告がイテレーターについて報告されるまで NULL を戻します。

close

形式:

```
public abstract void close() throws SQLException
```

イテレーターをクローズし、基礎となるデータベース・リソースを解放します。

getFetchSize

形式:

```
synchronized public int getFetchSize() throws SQLException
```

複数の行が必要な場合に SQLJ によりフェッチされる行数を戻します。戻り値は、setFetchSize メソッドにより設定された値か、または setFetchSize により値が設定されていない場合は 0 です。

getResultSet

形式:

```
public abstract ResultSet getResultSet() throws SQLException
```

イテレーターに関連付けられている JDBC ResultSet オブジェクトを戻します。

getRow

形式:

```
synchronized public int getRow() throws SQLException
```

現在行の番号を戻します。最初の行は番号 1、2 番目の行は番号 2、というようになります。イテレーターが行上に置かれていない場合は、0 が戻されます。

getSensitivity

形式:

```
synchronized public int getSensitivity() throws SQLException
```

イテレーターの感度を戻します。感度は、指定された感度値、またはイテレーター宣言節の with 節でデフォルト指定された値により決定されます。

getWarnings

形式:

```
public abstract SQLWarning getWarnings() throws SQLException
```

イテレーター上の呼び出しにより報告された最初の警告を戻します。後続のイテレーター警告は、この SQLWarning にチェーニングされます。警告チェーンは、イテレーターが新規行に移動するごとに自動的にクリアされます。

isClosed

形式:

```
public abstract boolean isClosed() throws SQLException
```

イテレーターがクローズされている場合は、`true` の値を戻します。そうでない場合は `false` を戻します。

next

形式:

```
public abstract boolean next() throws SQLException
```

イテレーターを次の行に進めます。 next を最初に呼び出す前に、イテレーターは結果表の最初の行の前に置かれます。 next は、次の行が使用可能な場合は `true` の値を、すべての行が検索済みの場合は `false` の値を戻します。

setFetchSize

形式:

```
synchronized public void setFetchSize(int number-of-rows) throws SQLException
```

複数の行が必要な場合にフェッチすべき行数に関して、SQLJ にヒントを与えます。

パラメーターは以下のとおりです。

number-of-rows
特定の実行コンテキストと関連付けられたイテレーターについて SQLJ がフェッチする予想行数。

number-of-rows が 0 より小さいか、フェッチ可能な行の最大数より大きい場合、SQLException がスローされます。

sqlj.runtime.Scrollable インターフェース

sqlj.runtime.Scrollable は、結果表内を移動し、結果表内での位置を確認するためのメソッドを提供します。

sqlj.runtime.Scrollable は、スクロール可能イテレーターが宣言されるときにインプリメントされます。

メソッド

absolute(int)

形式:

```
public abstract boolean absolute (int n) throws SQLException
```

イテレーターを指定された行に移動させます。

n>0 の場合、結果表の *n* 行にイテレーターが置かれます。 *n*<0 で、*m* が結果表に含まれる行数の場合は、結果表の *m+n+1* 行にイテレーターが置かれます。

n の絶対値が結果表に含まれる行数よりも大きい場合、*n* が正であれば最後の行の後ろ、*n* が負であれば最初の行の前にカーソルが置かれます。

absolute(0) は beforeFirst() と同じです。 absolute(1) は first() と同じです。 absolute(-1) は last() と同じです。

イテレーターが行の上にある場合、true が戻されます。そうでない場合、false が戻されます。

afterLast()

形式:

```
public abstract void afterLast() throws SQLException
```

イテレーターを結果表の最後の行の後ろに移動します。

beforeFirst()

形式:

```
public abstract void beforeFirst() throws SQLException
```

イテレーターを結果表の最初の行の前に移動します。

first()

形式:

```
public abstract boolean first() throws SQLException
```

イテレーターを結果表の最初の行に移動します。

イテレーターが行の上にある場合、true が戻されます。そうでない場合、false が戻されます。

getFetchDirection()

形式:

```
public abstract int getFetchDirection() throws SQLException
```

イテレーターのフェッチ方向を戻します。可能な値は以下のとおりです。

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
行は最初から最後に向けて、順方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
行は最後から最初に向けて、逆方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
処理の順序は不明です。

isAfterLast()
形式:

```
public abstract boolean isAfterLast() throws SQLException
```

イテレーターが結果表の最後の行の後ろにある場合、true が戻されます。そうでない場合、false が戻されます。

isBeforeFirst()
形式:

```
public abstract boolean isBeforeFirst() throws SQLException
```

イテレーターが結果表の最初の行の前にある場合、true が戻されます。そうでない場合、false が戻されます。

isFirst()
形式:

```
public abstract boolean isFirst() throws SQLException
```

イテレーターが結果表の最初の行にある場合、true が戻されます。そうでない場合、false が戻されます。

isLast()
形式:

```
public abstract boolean isLast() throws SQLException
```

イテレーターが結果表の最後の行にある場合、true が戻されます。そうでない場合、false が戻されます。

last()
形式:

```
public abstract boolean last() throws SQLException
```

イテレーターを結果表の最後の行に移動します。

イテレーターが行の上にある場合、true が戻されます。そうでない場合、false が戻されます。

previous()
形式:

```
public abstract boolean previous() throws SQLException
```

イテレーターを結果表の直前の行に移動します。

イテレーターが行の上にある場合、true が戻されます。そうでない場合、false が戻されます。

relative(int)
形式:

```
public abstract boolean relative(int n) throws SQLException
```

n>0 の場合、現在行から *n* 行後ろの行にイテレーターが置かれます。 *n*<0 の場合、現在行から *n* 行前の行にイテレーターが置かれます。 *n*=0 の場合、現在行にイテレーターが置かれます。

このメソッドを使用するには、カーソルが結果表の有効な行に置かれている必要があります。カーソルが最初の行の前あるいは最後の行の後ろにある場合、メソッドは SQLException をスローします。

m が結果表に含まれる行数、*x* が結果表内の現在行の番号とします。 *n*>0 であり、*x*+*n*>*m* である場合、イテレーターは最後の行の後ろに置かれます。 *n*<0 であり、*x*+*n*<1 である場合、イテレーターは最初の行の前に置かれます。

イテレーターが行の上にある場合、`true` が戻されます。そうでない場合、`false` が戻されます。

setFetchDirection(int)

形式:

```
public abstract void setFetchDirection (int) throws SQLException
```

このイテレーター・オブジェクトの行が処理される方向を、SQLJ ランタイム環境に示します。可能な値は以下のとおりです。

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
行は最初から最後に向けて、順方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
行は最後から最初に向けて、逆方向に処理されます。

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
処理の順序は不明です。

sqlj.runtime.AsciiStream クラス

sqlj.runtime.AsciiStream クラスは、指定された長さを持つ ASCII データの入力ストリーム用です。

sqlj.runtime.AsciiStream クラスは java.io.InputStream クラスから派生しており、sqlj.runtime.StreamWrapper クラスを拡張するものです。 SQLJ は、sqlj.runtime.AsciiStream オブジェクト内のバイトを ASCII 文字と解釈します。ASCII 文字を持つ InputStream オブジェクトは、sqlj.runtime.AsciiStream オブジェクトとして渡される必要があります。

コンストラクター

AsciiStream(InputStream)

形式:

```
public AsciiStream(java.io.InputStream input-stream)
```

未指定の長さの ASCII java.io.InputStream オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が AsciiStream オブジェクトとして解釈する InputStream オブジェクト。

AsciiStream(InputStream, int)

形式:

```
public AsciiStream(java.io.InputStream input-stream, int length)
```

指定された長さの ASCII java.io.InputStream オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が AsciiStream オブジェクトとして解釈する InputStream オブジェクト。

length
SQLJ が AsciiStream オブジェクトとして解釈する InputStream オブジェクトの長さ。

sqlj.runtime.BinaryStream クラス

sqlj.runtime.BinaryStream クラスは、指定された長さを持つバイナリー・データの入力ストリーム用です。

sqlj.runtime.BinaryStream クラスは java.io.InputStream クラスから派生しており、sqlj.runtime.StreamWrapper クラスを拡張するものです。 SQLJ は、sqlj.runtime.BinaryStream オブジェクト内のバイトをバイナリー文字として解釈します。バイナリー文字を持つ InputStream オブジェクトは、sqlj.runtime.BinaryStream オブジェクトとして渡される必要があります。

コンストラクター

BinaryStream(InputStream)

形式:

```
public BinaryStream(java.io.InputStream input-stream)
```

未指定の長さのバイナリー java.io.InputStream オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が BinaryStream オブジェクトとして解釈する InputStream オブジェクト。

BinaryStream(InputStream, int)

形式:

```
public BinaryStream(java.io.InputStream input-stream, int length)
```

指定された長さのバイナリー java.io.InputStream オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が BinaryStream オブジェクトとして解釈する InputStream オブジェクト。

length
SQLJ が BinaryStream オブジェクトとして解釈する InputStream オブジェクトの長さ。

sqlj.runtime.CharacterStream クラス

sqlj.runtime.CharacterStream クラスは、指定された長さを持つ文字データの入力ストリーム用です。

sqlj.runtime.CharacterStream クラスは java.io.Reader クラスから派生しており、java.io.FilterReader クラスを拡張するものです。 SQLJ は、sqlj.runtime.CharacterStream オブジェクト内のバイトを Unicode データとして解釈します。 Unicode データを持つ Reader オブジェクトは、sqlj.runtime.CharacterStream オブジェクトとして渡される必要があります。

コンストラクター

CharacterStream(InputStream)

形式:

```
public CharacterStream(java.io.Reader input-stream)
```

未指定の長さの文字 java.io.Reader オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が CharacterStream オブジェクトとして解釈する Reader オブジェクト。

CharacterStream(InputStream, int)

形式:

```
public CharacterStream(java.io.Reader input-stream, int length)
```

指定された長さの文字 java.io.Reader オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が CharacterStream オブジェクトとして解釈する Reader オブジェクト。

length
SQLJ が CharacterStream オブジェクトとして解釈する Reader オブジェクトの長さ。

メソッド

getReader

形式:

```
public Reader getReader()
```

CharacterStream オブジェクトでラップされる、基礎となる Reader オブジェクトを戻します。

getLength

形式:

```
public void getLength()
```

コンストラクターまたは setLength への最後の呼び出しで指定された、ラップされた Reader オブジェクトの文字単位での長さを戻します。

setLength

形式:

```
public void setLength (int length)
```

オブジェクトが SQL 操作への入力引数として渡されるときに、Reader オブジェクトから読み取られる文字数を設定します。

パラメーターは以下のとおりです。

length
Reader オブジェクトから読み取られる文字数。

sqlj.runtime.ExecutionContext クラス

sqlj.runtime.ExecutionContext クラスは、実行コンテキスト用に定義されています。実行コンテキストは、SQL ステートメントの実行を制御するために使用されます。

変数

ADD_BATCH_COUNT

形式:

```
public static final int ADD_BATCH_COUNT
```

getUpdateCount メソッドにより戻すことができる定数。これは前のステートメントが実行されておらず、既存のステートメント・バッチに追加されたことを示します。

AUTO_BATCH

形式:

```
public static final int AUTO_BATCH
```

setBatchLimit メソッドに渡すことができる定数。これは暗黙のバッチ実行を実行する必要があり、SQLJ がバッチ・サイズを決定する必要があることを示します。

EXEC_BATCH_COUNT

形式:

```
public static final int EXEC_BATCH_COUNT
```

getUpdateCount メソッドから戻すことができる定数。これはステートメント・バッチが実行されたばかりであることを示します。

EXCEPTION_COUNT

形式:

```
public static final int EXCEPTION_COUNT
```

getUpdateCount メソッドから戻すことができる定数。これは前の実行が完了する前に例外がスローされたか、または実行コンテキスト・オブジェクトに対して操作が実行されていないことを示します。

NEW_BATCH_COUNT

形式:

```
public static final int NEW_BATCH_COUNT
```

getUpdateCount メソッドから戻すことができる定数。これは前のステートメントが実行されておらず、新規のステートメント・バッチに追加されたことを示します。

QUERY_COUNT

形式:

```
public static final int QUERY_COUNT
```

setBatchLimit メソッドに渡すことができる定数。これは前の実行で結果セットが作成されたことを示します。

UNLIMITED_BATCH

形式:

```
public static final int UNLIMITED_BATCH
```

getUpdateCount メソッドから戻すことができる定数。これは、バッチ・サイズに関係なく、ステートメントを引き続きステートメント・バッチに追加する必要があることを示します。

コンストラクター:

ExecutionContext

形式:

```
public ExecutionContext()
```

ExecutionContext インスタンスを作成します。

メソッド

cancel

形式:

```
public void cancel() throws SQLException
```

実行コンテキスト・オブジェクトを使用するスレッドにより現在実行中の SQL 操作をキャンセルします。実行コンテキスト・オブジェクト上に保留中のステートメント・バッチがある場合、そのステートメント・バッチはキャンセルされてクリアされます。

cancel メソッドは、ステートメントがキャンセルできない場合は SQLException をスローします。

execute

形式:

```
public boolean execute ( ) throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

executeBatch

形式:

```
public synchronized int[] executeBatch() throws SQLException
```

保留中のステートメント・バッチを実行し、更新カウントの配列を戻します。保留中のステートメント・バッチがない場合は、NULL が戻されます。このメソッドが呼び出されると、呼び出しが例外の結果になった場合でも、ステートメント・バッチはクリアされます。

戻される配列内の各要素は、以下のいずれかの値になります。

-2 この値は、SQL ステートメントは正常に実行されたが、更新された行数は判別できなかったことを示します。

-3 この値は SQL ステートメントが失敗したことを示します。

その他の整数
この値は、ステートメントにより更新された行の数です。

executeBatch メソッドは、データベース・エラーがステートメント・バッチの実行中に発生した場合、SQLException をスローします。

executeQuery

形式:

```
public RTResultSet executeQuery ( ) throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

executeUpdate

形式:

```
public int executeUpdate() throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

getBatchLimit

形式:

```
synchronized public int getBatchLimit()
```

バッチが暗黙的に実行される前にバッチに追加されたステートメントの数を戻します。

戻り値は以下のいずれかになります。

UNLIMITED_BATCH
この値はバッチ・サイズが無制限であることを示します。

AUTO_BATCH
この値はバッチ・サイズが有限であるが不明であることを示します。

その他の整数
現在のバッチ制限。

getBatchUpdateCounts

形式:

```
public synchronized int[] getBatchUpdateCounts()
```

バッチ内で正常に実行された各ステートメントにより更新された行数を含む配列を戻します。配列内の要素の順序は、ステートメントがバッチに挿入された順序に対応します。バッチ内のどのステートメントも正常に完了しなかった場合には NULL を戻します。

戻される配列内の各要素は、以下のいずれかの値になります。

-2 この値は、SQL ステートメントは正常に実行されたが、更新された行数は判別できなかったことを示します。

-3 この値は SQL ステートメントが失敗したことを示します。

その他の整数
この値は、ステートメントにより更新された行の数です。

getFetchDirection

形式:

```
synchronized public int getFetchDirection() throws SQLException
```

特定の実行コンテキストから生成されたスクロール可能反復子オブジェクトの、現在のフェッチ方向を戻します。フェッチ方向が実行コンテキストに設定されていない場合、sqlj.runtime.ResultSetIterator.FETCH_FORWARD が戻されます。

getFetchSize

形式:

```
synchronized public int getFetchSize() throws SQLException
```

複数の行が必要な場合に SQLJ によりフェッチされる行数を戻します。この値は、特定の実行コンテキストから生成された反復子オブジェクトだけに適用されます。戻り値は、setFetchSize メソッドにより設定された値か、または setFetchSize により値が設定されていない場合は 0 です。

getMaxFieldSize

形式:

```
public synchronized int getMaxFieldSize()
```

特定の実行コンテキストを使用する照会内のストリング (文字、グラフィック、または可変長バイナリー) 列に対して戻される、バイトの最大数を戻します。この制限を超えた場合、SQLJ は残りのバイトを破棄します。値 0 は、バイトの最大数が無制限であることを意味します。

getMaxRows

形式:

```
public synchronized int getMaxRows()
```

特定の実行コンテキストを使用する照会に対して戻される、行の最大数を戻します。この制限を超えた場合、SQLJ は残りの行を破棄します。値 0 は、行の最大数が無制限であることを意味します。

getNextResultSet()

形式:

```
public ResultSet getNextResultSet() throws SQLException
```

ストアード・プロシージャー呼び出しの後に、ストアード・プロシージャーから結果セットを戻します。

以下のいずれかの条件が当てはまるときは、NULL 値が戻されます。

- 戻される結果セットがそれ以上存在しない。
- ストアード・プロシージャー呼び出しが結果セットを作成しなかった。
- ストアード・プロシージャー呼び出しが実行コンテキストの下で実行されていない。

getNextResultSet() を呼び出す場合、SQLJ は現在オープンしている結果セットをクローズし、次の結果セットに進みます。

getNextResultSet への呼び出し中にエラーが発生した場合、現在の JDBC ResultSet オブジェクトのリソースは解放され、SQLException がスローされます。 getNextResultSet の後続の呼び出しは NULL を戻します。

getNextResultSet(int)

形式:

```
public ResultSet getNextResultSet(int current)
```

ストアード・プロシージャー呼び出しの後に、ストアード・プロシージャーから結果セットを戻します。

以下のいずれかの条件が当てはまるときは、NULL 値が戻されます。

- 戻される結果セットがそれ以上存在しない。
- ストアード・プロシージャー呼び出しが結果セットを作成しなかった。
- ストアード・プロシージャー呼び出しが実行コンテキストの下で実行されていない。

getNextResultSet への呼び出し中にエラーが発生した場合、現在の JDBC ResultSet オブジェクトのリソースは解放され、SQLException がスローされます。 getNextResultSet の後続の呼び出しは NULL を戻します。

パラメーターは以下のとおりです。

current
次の結果セットに進む前に、現在オープンしている結果セットを使用して SQLJ が実行する内容を示します。

java.sql.Statement.CLOSE_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現在の ResultSet オブジェクトをクローズすることを指定します。

java.sql.Statement.KEEP_CURRENT_RESULT
次の ResultSet オブジェクトが戻されるときに、現在の ResultSet オブジェクトをオープンしたままにすることを指定します。

java.sql.Statement.CLOSE_ALL_RESULTS
次の ResultSet オブジェクトが戻されるときに、オープンしているすべての ResultSet オブジェクトをクローズすることを指定します。

getQueryTimeout

形式:

```
public synchronized int getQueryTimeout()
```

特定の実行コンテキスト・オブジェクトを使用する SQL 操作が実行できる、最大秒数を戻します。 SQL 操作がこの制限を超える場合、SQLException がスロ

ーされます。戻り値は、setQueryTimeout メソッドにより設定された値か、または setQueryTimeout により値が設定されていない場合は 0 です。 0 は、実行時間が無制限であることを示します。

getUpdateCount

形式:

```
public abstract int getUpdateCount() throws SQLException
```

以下を戻します。

ExecutionContext.ADD_BATCH_COUNT
ステートメントが既存のバッチに追加された場合。

ExecutionContext.NEW_BATCH_COUNT
ステートメントが新規バッチ内の最初のステートメントであった場合。

ExecutionContext.EXCEPTION_COUNT
前のステートメントが SQLException を生成したか、または前に実行されたステートメントがない場合。

ExecutionContext.EXEC_BATCH_COUNT
ステートメントがバッチの一部であり、バッチが実行された場合。

ExecutionContext.QUERY_COUNT
前のステートメントが反復子オブジェクトまたは JDBC ResultSet を作成した場合。

その他の整数
ステートメントが、バッチに追加されたのではなく実行された場合。この値は、ステートメントにより更新された行の数です。

getWarnings

形式:

```
public synchronized SQLWarning getWarnings()
```

特定の実行コンテキストを使用して実行された最後の SQL 操作によりレポートされた、最初の警告を戻します。後続の警告は、最初の警告にチェーニングされます。警告が出されていない場合は、NULL が戻されます。

getWarnings は、正の SQLCODE を検索するために使用します。

isBatching

形式:

```
public synchronized boolean isBatching()
```

実行コンテキストに対してバッチが使用可能である場合には `true` を戻します。バッチが使用不可の場合は `false` を戻します。

registerStatement

形式:

```
public RTStatement registerStatement(ConnectionContext connCtx,
 Object profileKey, int stmtNdx)
 throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

releaseStatement

形式:

```
public void releaseStatement() throws SQLException
```

このメソッドは、SQLJ 変換プログラムによって生成されるコードで使用されます。これはアプリケーション・プログラムによる直接の使用向けのものではありません。

setBatching

形式:

```
public synchronized void setBatching(boolean batching)
```

パラメーターは以下のとおりです。

batching
特定の実行コンテキストで登録されたバッチ可能ステートメントが、ステートメント・バッチに追加できるかどうかを示します。

true
ステートメントはステートメント・バッチに追加できます。

false
ステートメントは個別に実行されます。

setBatching は、setBatching の呼び出し後のプログラム内で実行されるステートメントだけに影響を与えます。これは前のステートメントまたは既存のステートメント・バッチには影響を与えません。

setBatchLimit

形式:

```
public synchronized void setBatchLimit(int batch-size)
```

バッチが暗黙的に実行される前にバッチに追加されるステートメントの最大数を設定します。

パラメーターは以下のとおりです。

batch-size
以下のいずれかの値になります。

ExecutionContext.UNLIMITED_BATCH
SQLJ がバッチ可能であるが非互換であるステートメント、またはバッチ可能ではないステートメントを検出した場合にのみ、暗黙の実行が行われることを示します。この値を設定することは、setBatchLimit を呼び出さないことと同じです。

ExecutionContext.AUTO_BATCH
バッチ内のステートメントの数が SQLJ により設定された数に達したときに、暗黙の実行が行われることを示します。

正の整数
SQLJ がバッチを暗黙に実行する前にバッチに追加されたステートメン

トの数。 SQLJ がバッチ可能であるが非互換であるステートメント、またはバッチ可能ではないステートメントを検出した場合、この数のステートメントが追加される前にバッチは実行されている可能性があります。

setBatchLimit は、setBatchLimit の呼び出し後のプログラム内で実行されるステートメントだけに影響を与えます。既存のステートメント・バッチには影響を与えません。

setFetchDirection

形式:

```
public synchronized void setFetchDirection(int direction) throws SQLException
```

特定の実行コンテキストから生成されたスクロール可能反復子オブジェクトの現在のフェッチ方向に関して、SQLJ にヒントを与えます。

パラメーターは以下のとおりです。

direction
以下のいずれかの値になります。

sqlj.runtime.ResultSetIterator.FETCH_FORWARD
行は順方向でフェッチされます。これはデフォルトです。

sqlj.runtime.ResultSetIterator.FETCH_REVERSE
行は逆方向でフェッチされます。

sqlj.runtime.ResultSetIterator.FETCH_UNKNOWN
フェッチの順序は不明です。

その他の入力値の結果は SQLException になります。

setFetchSize

形式:

```
synchronized public void setFetchSize(int number-of-rows) throws SQLException
```

複数の行が必要な場合にフェッチすべき行数に関して、SQLJ にヒントを与えます。

パラメーターは以下のとおりです。

number-of-rows
特定の実行コンテキストと関連付けられたイテレーターについて SQLJ がフェッチする予想行数。

number-of-rows が 0 より小さいか、フェッチ可能な行の最大数より大きい場合、SQLException がスローされます。

setMaxFieldSize

形式:

```
public void setMaxFieldSize(int max-bytes)
```

特定の実行コンテキストを使用する照会内のストリング (文字、グラフィック、または可変長バイナリー) 列に対して戻される、バイトの最大数を指定します。この制限を超えた場合、SQLJ は残りのバイトを破棄します。

パラメーターは以下のとおりです。

max-bytes
SQLJ が BINARY、VARBINARY、CHAR、VARCHAR、GRAPHIC、または VARGRAPHIC 列から戻す最大バイト数。値 0 は、バイト数が無制限であることを意味します。 0 がデフォルトです。

setMaxRows
形式:

```
public synchronized void setMaxRows(int max-rows)
```

特定の実行コンテキストを使用する照会に対して戻される、行の最大数を指定します。この制限を超えた場合、SQLJ は残りの行を破棄します。

パラメーターは以下のとおりです。

max-rows
特定の実行コンテキストを使用する照会に対して SQLJ が戻す、行の最大数。値 0 は、行数が無制限であることを意味します。 0 がデフォルトです。

setQueryTimeout
形式:

```
public synchronized void setQueryTimeout(int timeout-value)
```

特定の実行コンテキスト・オブジェクトを使用する SQL 操作が実行できる、最大秒数を指定します。 SQL 操作がこの制限を超える場合、SQLException がスローされます。

パラメーターは以下のとおりです。

timeout-value
特定の実行コンテキスト・オブジェクトを使用する SQL 操作が実行できる、最大秒数。 0 は、実行時間が無制限であることを示します。 0 がデフォルトです。

sqlj.runtime.SQLNullException クラス

sqlj.runtime.SQLNullException クラスは java.sql.SQLException クラスから派生します。

SQL NULL 値がフェッチされ、ホスト ID に Java プリミティブ・タイプで入れられると、sqlj.runtime.SQLNullException がスローされます。SQLNullException のインスタンスの SQLSTATE 値は '22002' です。

sqlj.runtime.StreamWrapper クラス

sqlj.runtime.StreamWrapper クラスは java.io.InputStream インスタンスをラップし、java.io.InputStream クラスを拡張するものです。

sqlj.runtime.AsciiStream、sqlj.runtime.BinaryStream、および sqlj.runtime.UnicodeStream クラスは sqlj.runtime.StreamWrapper を拡張するものです。
`sqlj.runtime.StreamWrapper` は、
sqlj.runtime.AsciiStream、sqlj.runtime.BinaryStream、および sqlj.runtime.UnicodeStream オブジェクトの長さを指定するメソッドをサポートします。

コンストラクター

StreamWrapper(InputStream)

形式:

```
protected StreamWrapper(InputStream input-stream)
```

未指定の長さの sqlj.runtime.StreamWrapper オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
sqlj.runtime.StreamWrapper オブジェクトがラップする InputStream オブジェクト。

StreamWrapper(InputStream, int)

形式:

```
protected StreamWrapper(java.io.InputStream input-stream, int length)
```

指定された長さの sqlj.runtime.StreamWrapper オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
sqlj.runtime.StreamWrapper オブジェクトがラップする InputStream オブジェクト。

length
InputStream オブジェクトのバイト単位の長さ。

メソッド

getInputStream

形式:

```
public InputStream getInputStream()
```

StreamWrapper オブジェクトでラップされる、基礎となる InputStream オブジェクトを戻します。

getLength

形式:

```
public void getLength()
```

コンストラクターまたは setLength への最後の呼び出しで指定された、ラップされた InputStream オブジェクトのバイト単位での長さを戻します。

setLength

形式:

```
public void setLength (int length)
```

オブジェクトが SQL 操作への入力引数として渡されるときに、ラップされた InputStream オブジェクトから読み取られるバイト数を設定します。

パラメーターは以下のとおりです。

length
ラップされた InputStream オブジェクトから読み取られるバイト数。

sqlj.runtime.UnicodeStream クラス

sqlj.runtime.UnicodeStream クラスは、指定された長さを持つ Unicode データの入力ストリーム用です。

sqlj.runtime.UnicodeStream クラスは java.io.InputStream クラスから派生しており、sqlj.runtime.StreamWrapper クラスを拡張するものです。 SQLJ は、sqlj.runtime.UnicodeStream オブジェクト内のバイトを Unicode 文字として解釈します。 Unicode 文字を持つ InputStream オブジェクトは、sqlj.runtime.UnicodeStream オブジェクトとして渡される必要があります。

コンストラクター

UnicodeStream(InputStream)

形式:

```
public UnicodeStream(java.io.InputStream input-stream)
```

未指定の長さの Unicode java.io.InputStream オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が UnicodeStream オブジェクトとして解釈する InputStream オブジェクト。

UnicodeStream(InputStream, int)

形式:

```
public UnicodeStream(java.io.InputStream input-stream, int length)
```

指定された長さの Unicode java.io.InputStream オブジェクトを作成します。

パラメーターは以下のとおりです。

input-stream
SQLJ が UnicodeStream オブジェクトとして解釈する InputStream オブジェクト。

length
SQLJ が UnicodeStream オブジェクトとして解釈する InputStream オブジェクトの長さ。

JDBC に対する IBM Data Server Driver for JDBC and SQLJ 拡張機能

IBM Data Server Driver for JDBC and SQLJ は、JDBC 仕様によって提供されるサポートに対して拡張機能のセットを提供します。

対応する標準クラスが含まれるクラスで IBM Data Server Driver for JDBC and SQLJ 限定のメソッドを使用するには、関連した標準 JDBC クラスのインスタンスを IBM Data Server Driver for JDBC and SQLJ 限定のクラスのインスタンスにキャストします。以下に例を示します。

```
javax.sql.DataSource ds =
 new com.ibm.db2.jcc.DB2SimpleDataSource();
((com.ibm.db2.jcc.DB2BaseDataSource) ds).setServerName("sysmvs1.stl.ibm.com");
```

表 97 は IBM Data Server Driver for JDBC and SQLJ 限定のインターフェースを要約したものです。

表 97. IBM Data Server Driver for JDBC and SQLJ によって提供される IBM Data Server Driver for JDBC and SQLJ 限定のインターフェースのサマリー

インターフェースの名前	適用できるデータ・ソース	目的
DB2CallableStatement	2	java.sql.CallableStatement および com.ibm.db2.jcc.DB2PreparedStatement インターフェースを拡張するものです。
DB2Connection	1, 2, 3	java.sql.Connection インターフェースを拡張するものです。
DB2DatabaseMetaData	1, 2, 3	java.sql.DatabaseMetaData インターフェースを拡張するものです。
DB2Diagnosable	1, 2, 3	DB2 SQLException から DB2 診断を取得するメカニズムを提供します。
DB2PreparedStatement	1, 2, 3	com.ibm.db2.jcc.DB2Statement および java.sql.PreparedStatement インターフェースを拡張するものです。
DB2RowID	1, 2	ROWID データ・タイプで使用するための Java オブジェクトを宣言するのに使用されます。
DB2Statement	1, 2, 3	java.sql.Statement インターフェースを拡張するものです。
DB2SystemMonitor	1, 2, 3	接続用にシステム・モニター・データを収集するのに使用されます。
DB2TraceManagerMXBean	1, 2, 3	リモート・トレース・コントローラー用に MBean インターフェースを提供します。
DB2Xml	1, 2	XML 列のデータの更新、および XML 列からのデータの取り出しを実行するために使用されます。
DBBatchUpdateException	1, 2, 3	自動生成キーを返すステートメントのバッチ実行についてのエラー情報を取り出すために使用されます。

注: インターフェースは以下のデータ・ソースの接続に適用されます。

1. DB2 for z/OS
2. DB2 Database for Linux, UNIX, and Windows
3. IBM Informix Dynamic Server

表 98 は、 IBM Data Server Driver for JDBC and SQLJ 限定のクラスを要約したものです。

表 98. IBM Data Server Driver for JDBC and SQLJ によって提供される IBM Data Server Driver for JDBC and SQLJ 限定のクラスのサマリー

クラス名	適用できるデータ・ソース	目的
DB2Administrator (DB2 Database for Linux, UNIX, and Windows のみ)	2 (424 ページ)	DB2Administrator クラスのインスタンスは、DB2CataloguedDatabase オブジェクトの検索に使用されます。

表 98. IBM Data Server Driver for JDBC and SQLJ によって提供される IBM Data Server Driver for JDBC and SQLJ 限定のクラスのサマリー (続き)

クラス名	適用できるデータ・ソース	目的
DB2BaseDataSource	1, 2, 3	javax.sql.DataSource、javax.sql.ConnectionPoolDataSource、および javax.sql.XADataSource のすべての IBM Data Server Driver for JDBC and SQLJ 固有のインプリメンテーションのための抽象データ・ソース親クラスです。
DB2CataloguedDatabase	2	ローカル DB2 Database for Linux, UNIX, and Windows データベースについての情報を検索するメソッドが含まれています。
DB2ClientRerouteServerList	1, 2	java.io.Serializable および javax.naming.Referenceable インターフェースをインプリメントします。
DB2ConnectionPoolDataSource	1, 2, 3	PooledConnection オブジェクトのファクトリー。
DB2ExceptionFormatter	1, 2, 3	診断情報をストリームに印刷するためのメソッドが含まれます。
DB2JCCPlugin	2	JDBC セキュリティー・プラグインのインプリメンテーションのための抽象クラスです。
DB2PooledConnection	1, 2, 3	以前から存在するトラステッド接続でユーザーを切り替えるためにアプリケーション・サーバーが使用できるメソッドを提供します。
DB2PoolMonitor	1, 2	接続コンセントレーターと Sysplex ワークロード・バランシングのグローバル・トランスポート・オブジェクト・プールをモニターするためのメソッドを提供します。
DB2SimpleDataSource	1, 2, 3	DataBaseDataSource クラスを拡張します。接続プールまたは分散トランザクションはサポートしていません。
DB2Sqlca	1, 2, 3	DB2 SQLCA をカプセル化したものです。
DB2TraceManager	1, 2, 3	グローバル・ログ書き込みプログラムを制御します。
DB2Types	1 (423 ページ)	データ・タイプ定数を定義します。
DB2XADataSource	1, 2, 3	XADataSource オブジェクトのファクトリー。このインターフェースをインプリメントするオブジェクトは、Java Naming and Directory Interface (JNDI) に基づくネーム・サービスで登録されます。

注: このクラスは以下のデータ・ソースの接続に適用されます。

1. DB2 for z/OS
2. DB2 Database for Linux, UNIX, and Windows
3. IBM Informix Dynamic Server

DBBatchUpdateException インターフェース

com.ibm.db2.jcc.DBBatchUpdateException インターフェースは、自動生成キーを返すステートメントのバッチ実行についてのエラー情報を取り出すために使用されます。

DBBatchUpdateException メソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getDBGeneratedKeys

形式:

```
public java.sql.ResultSet[] getDBGeneratedKeys()
  throws java.sql.SQLException
```

INSERT ステートメントがバッチで実行されたときに作成された自動生成キーを取り出します。戻される ResultSet オブジェクトにはそれぞれ、バッチの 1 つのステートメントに対する自動生成キーが含まれます。 ResultSet オブジェクトがヌルの場合、それは失敗したステートメントに対応しています。

DB2Administrator クラス

com.ibm.db2.jcc.DB2Administrator クラスのインスタンスは、DB2CataloguedDatabase オブジェクトの検索に使用されます。 DB2Administrator は、 DB2 Database for Linux, UNIX, and Windows データベースだけに適用されます。

DB2Administrator のメソッド

getInstance

形式:

```
public static DB2Administrator getInstance()
```

DB2Administrator クラスのインスタンスを戻します。

getCataloguedDatabases

形式:

```
public DB2CataloguedDatabase[] getCataloguedDatabases()
 throws java.sql.SQLException
```

ローカル・データベース・ディレクトリー内の各ローカル・データベースの DB2CataloguedDatabase オブジェクトを含む配列を検索します。

ローカル DB2 システムが使用可能であり、カタログにデータベースが含まれていない場合、長さゼロの配列が戻されます。使用可能なローカル DB2 システムがない場合は、ヌルが戻されます。ローカル・システムが DB2 Database for Linux, UNIX, and Windows システムではない場合、SQLException がスローされます。

DB2BaseDataSource クラス

com.ibm.db2.jcc.DB2BaseDataSource クラスは、
javax.sql.DataSource、javax.sql.ConnectionPoolDataSource、および

javax.sql.XADataSource のすべての IBM Data Server Driver for JDBC and SQLJ 固有のインプリメンテーションのための抽象データ・ソース親クラスです。

DB2BaseDataSource は、java.sql.Wrapper インターフェースをインプリメントします。

DB2BaseDataSource のプロパティー

以下のプロパティーは、 IBM Data Server Driver for JDBC and SQLJに対してのみ定義されます。

プロパティーはすべて DataSource で、あるいは DriverManager.getConnection 呼び出しの *url* パラメーターで設定することができます。

以下のプロパティーを**除く**すべてのプロパティーに、プロパティーの値を設定する setXXX メソッド、および値を検索する getXXX メソッドがあります。

- minTransportObjects
- maxTransportObjectIdleTime
- maxTransportObjectWaitTime
- dumpPool
- dumpPoolStatisticsOnSchedule
- dumpPoolStatisticsOnScheduleFile

setXXX メソッドの形式は以下のとおりです。

```
void setProperty-name(data-type property-value)
```

getXXX メソッドの形式は以下のとおりです。

```
data-type getProperty-name()
```

Property-name は、非修飾プロパティー名です。 IBM Informix Dynamic Server (IDS) に固有ではないプロパティーの場合、プロパティー名の先頭文字を大文字にしています。IDS のみによって使用されるプロパティーの場合は、プロパティー名の文字すべてが大文字になっています。

以下の表に、 IBM Data Server Driver for JDBC and SQLJ のプロパティーとそのデータ・タイプをリストします。

表 99. DB2BaseDataSource プロパティーとそれらのデータ・タイプ

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.accountingInterval	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.allowNextOnExhaustedResultSet	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.atomicMultiRowInsert	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.blockingReadConnectionTimeout	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.charOutputSize	1 (432 ページ)	short

表 99. *DB2BaseDataSource* プロパティーとそれらのデータ・タイプ (続き)

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.clientAccountingInformation	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientApplicationInformation	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientDebugInfo (IBM Data Server Driver for JDBC and SQLJ Type 4 接続)	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientProgramId	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientProgramName (IBM Data Server Driver for JDBC and SQLJ Type 4 接続)	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteAlternateServerName	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteAlternatePortNumber	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteServerListJNDIContext	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	javax.naming.Context
com.ibm.db2.jcc.DB2BaseDataSource.clientRerouteServerListJNDIName	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientUser (DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ)	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.clientWorkstation (DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ)	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.concurrentAccessResolution	2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.connectNode	2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.currentDegree	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentExplainMode	2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentExplainSnapshot	2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentFunctionPath	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentLockTimeout	2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.currentMaintainedTableTypesForOptimization	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentPackagePath	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentPackageSet	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.currentQueryOptimization	2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.currentRefreshAge	1 (432 ページ), 2 (432 ページ)	long
com.ibm.db2.jcc.DB2BaseDataSource.currentSchema	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.cursorSensitivity	1 (432 ページ), 2 (432 ページ)	int

表 99. *DB2BaseDataSource* プロパティーとそれらのデータ・タイプ (続き)

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.currentSQLID	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.databaseName	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.dateFormat	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.decimalRoundingMode	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.decimalSeparator	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.decimalStringFormat	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.defaultIsolationLevel	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.deferPrepares	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.description	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.downgradeHoldCursorsUnderXa	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.driverType	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.dumpPool	3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.dumpPoolStatisticsOnSchedule	3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.dumpPoolStatisticsOnScheduleFile	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.enableClientAffinitiesList	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.enableNamedParameterMarkers	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.enableConnectionConcentrator	1 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.enableRowsetSupport	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.enableSeamlessFailover	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.enableSysplexWLB	1 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.encryptionAlgorithm	1 (432 ページ), 2 (432 ページ)	int

表 99. *DB2BaseDataSource* プロパティーとそれらのデータ・タイプ (続き)

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.fetchSize	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeInputStreams	1 (432 ページ), 2 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.fullyMaterializeLobData	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.gssCredential	1 (432 ページ), 2 (432 ページ)	オブジェクト
com.ibm.db2.jcc.DB2BaseDataSource.jdbcCollection	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.keepDynamic	1 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.kerberosServerPrincipal	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.loginTimeout (DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続ではサポートされない)	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.logWriter	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	PrintWriter
com.ibm.db2.jcc.DB2BaseDataSource.maxRetriesForClientReroute	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.maxRowsetSize (DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 のみ)	1 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjectIdleTime	3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjectWaitTime	3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.maxTransportObjects	1 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.minTransportObjects	3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.optimizationProfile	2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.optimizationProfileToFlush	2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.password	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.pdqProperties	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.pkList (IBM Data Server Driver for JDBC and SQLJ Type 2 接続)	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.planName (IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ)	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.plugin	2 (432 ページ)	オブジェクト
com.ibm.db2.jcc.DB2BaseDataSource.pluginName	2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.portNumber	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int

表 99. *DB2BaseDataSource* プロパティーとそれらのデータ・タイプ (続き)

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.progressiveStreaming	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.queryDataSize	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.queryCloseImplicit	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.readOnly	1 (432 ページ), 2 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.reportLongTypes	1 (432 ページ)	short
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldability	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.resultSetHoldabilityForCatalogQueries	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.retrieveMessagesFromServerOnGetMessage	1 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.retryIntervalForClientReroute	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.retryWithAlternativeSecurityMechanism (IBM Data Server Driver for JDBC and SQLJ Type 4 接続)	2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.returnAlias	1 (432 ページ), 2 (432 ページ)	short
com.ibm.db2.jcc.DB2BaseDataSource.securityMechanism	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.sendCharInputsUTF8	1 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.sendDataAsIs	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.serverName	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.sqljEnableClassLoaderSpecificProfiles	1 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.ssid (DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 のみ)	1 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.sslConnection (IBM Data Server Driver for JDBC and SQLJ Type 4 接続)	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.sslTrustStoreLocation (IBM Data Server Driver for JDBC and SQLJ Type 4 接続)	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.sslTrustStorePassword (IBM Data Server Driver for JDBC and SQLJ Type 4 接続)	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.statementConcentrator	2 (432 ページ)	int

表 99. *DB2BaseDataSource* プロパティーとそれらのデータ・タイプ (続き)

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.streamBufferSize	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.supportsAsynchronousXARollback	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.sysSchema	1 (432 ページ), 2 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.timeFormat	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.timestampFormat	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.timestampPrecisionReporting	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.traceDirectory	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.traceFile	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.traceFileAppend	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.traceLevel	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.useCachedCursor	1 (432 ページ), 2 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useJDBC4ColumnNameAndLabelSemantics	1 (432 ページ), 2 (432 ページ)	int
com.ibm.db2.jcc.DB2BaseDataSource.user	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.useRowsetCursor	1 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.useTransactionRedirect	2 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.xaNetworkOptimization	1 (432 ページ), 2 (432 ページ), 3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.DBANSIWARN	3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.DBDATE	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.DBPATH	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.DBSPACETEMP	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.DBTEMP	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.DBUPSPACE	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.DELIMIDENT	3 (432 ページ)	boolean
com.ibm.db2.jcc.DB2BaseDataSource.IFX_DIRECTIVES	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.IFX_EXTDIRECTIVES	3 (432 ページ)	String
com.ibm.db2.jcc.DB2BaseDataSource.IFX_UPDDESC	3 (432 ページ)	String

表 99. *DB2BaseDataSource* プロパティーとそれらのデータ・タイプ *(続き)*

プロパティー名	適用できるデータ・ソース	データ・タイプ
com.ibm.db2.jcc.DB2BaseDataSource.IFX_XASTDCOMPLIANCE_XAEND	3	String
com.ibm.db2.jcc.DB2BaseDataSource.INFORMIXOPCACHE	3	String
com.ibm.db2.jcc.DB2BaseDataSource.INFORMIXSTACKSIZE	3	String
com.ibm.db2.jcc.DB2BaseDataSource.NODEFDAC	3	String
com.ibm.db2.jcc.DB2BaseDataSource.OPTCOMPIND	3	String
com.ibm.db2.jcc.DB2BaseDataSource.OPTOFC	3	String
com.ibm.db2.jcc.DB2BaseDataSource.PDQPRIORITY	3	String
com.ibm.db2.jcc.DB2BaseDataSource.PSORT_DBTEMP	3	String
com.ibm.db2.jcc.DB2BaseDataSource.PSORT_NPROCS	3	String
com.ibm.db2.jcc.DB2BaseDataSource.STMT_CACHE	3	String

注: プロパティーは以下のデータ・ソースの接続に適用されます。

1. DB2 for z/OS
2. DB2 Database for Linux, UNIX, and Windows
3. IBM Informix Dynamic Server

DB2BaseDataSource のメソッド

DB2BaseDataSource プロパティーの getXXX および setXXX メソッドに加えて、以下のメソッドが IBM Data Server Driver for JDBC and SQLJ のためだけに定義されています。

getReference

形式:

```
public javax.naming.Reference getReference()
  throws javax.naming.NamingException
```

DataSource オブジェクトの Reference を検索します。 Reference については、以下の URL にある JNDI 文書から javax.naming.Referenceable についての説明を参照してください。

```
http://java.sun.com/products/jndi/docs.html
```

DB2CallableStatement インターフェース

com.ibm.db2.jcc.DB2CallableStatement インターフェースは、java.sql.CallableStatement および com.ibm.db2.jcc.DB2PreparedStatement インターフェースを拡張するものです。

DB2CallableStatement メソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

registerJccOutParameterAtName

形式:

```
public void registerJccOutParameterAtName(String parameterMarkerName,
  int sqlType)
  throws java.sql.SQLException
public void registerJccOutParameterAtName(String parameterMarkerName,
```

```
  int sqlType,
  int scale)
  throws java.sql.SQLException
public void registerJccOutParameterAtName(String parameterMarkerName,
  int sqlType,
  String typeName)
  throws java.sql.SQLException
```

parameterMarkerName によって JDBC タイプ *sqlType* として識別される OUT パラメーターを登録します。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
登録されるパラメーターのパラメーター・マーカーの名前。

sqlType
登録されるパラメーターの JDBC タイプ・コード。これは、java.sql.Types で定義されるものです。

scale
登録されるパラメーターのスケール。このパラメーターは以下の場合にのみ適用されます。

- *sqlType* が java.sql.Types.DECIMAL または java.sql.Types.NUMERIC である場合、*scale* は小数点の右側の桁数です。

typeName
jdbcType が java.sql.Types.DISTINCT または java.sql.Types.REF である場合、登録されるパラメーターの SQL ユーザー定義タイプの完全修飾名。

setJccXXXAtName メソッド
これらのメソッドは DB2PreparedStatement から継承されます。

DB2CataloguedDatabase クラス

com.ibm.db2.jcc.DB2CataloguedDatabase クラスには、ローカル DB2 Database for Linux, UNIX, and Windows データベースについての情報を検索するメソッドが含まれています。

DB2CataloguedDatabase メソッドの呼び出しにはデータベース接続は不要です。

DB2CataloguedDatabase のメソッド

getServerName
形式:

```
public String getServerName()
```

データベースが置かれているサーバーの名前を検索します。

getPortNumber
形式:

```
public int getPortNumber()
```

DB2 インスタンスと関連付けられているポート番号を検索します。

getDatabaseName

形式:

```
public String getDatabaseName()
```

データベース名を検索します。

getDatabaseAlias

形式:

```
public String getDatabaseAlias()
```

データベース別名を検索します。

DB2ClientRerouteServerList クラス

com.ibm.db2.jcc.DB2ClientRerouteServerList クラスは、java.io.Serializable および javax.naming.Referenceable インターフェースをインプリメントします。

DB2ClientRerouteServerList のメソッド

getAlternatePortNumber

形式:

```
public int[] getAlternatePortNumber()
```

代替サーバーと関連付けられているポート番号を検索します。

getAlternateServerName

形式:

```
public String[] getAlternateServerName()
```

代替サーバーの名前を含む配列を検索します。これらの値は IP アドレスまたは DNS サーバー名です。

getPrimaryPortNumber

形式:

```
public int getPrimaryPortNumber()
```

1 次サーバーと関連付けられているポート番号を検索します。

getPrimaryServerName

形式:

```
public String[] getPrimaryServerName()
```

1 次サーバーの名前を検索します。この値は IP アドレスまたは DNS サーバー名です。

setAlternatePortNumber

形式:

```
public void setAlternatePortNumber(int[] alternatePortNumberList)
```

代替サーバーと関連付けられているポート番号を設定します。

setAlternateServerName

形式:

```
public void setAlternateServerName(String[] alternateServer)
```

サーバーの代替サーバー名を設定します。これらの値は IP アドレスまたは DNS サーバー名です。

setPrimaryPortNumber

形式:

```
public void setPrimaryPortNumber(int primaryPortNumber)
```

1 次サーバーと関連付けられているポート番号を設定します。

setPrimaryServerName

形式:

```
public void setPrimaryServerName(String primaryServer)
```

サーバーの 1 次サーバー名を設定します。この値は IP アドレスまたは DNS サーバー名です。

DB2Connection インターフェース

com.ibm.db2.jcc.DB2Connection インターフェースは java.sql.Connection を拡張するものです。

DB2Connection は java.sql.Wrapper インターフェースをインプリメントします。

DB2Connection のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

alternateWasUsedOnConnect

形式:

```
public boolean alternateWasUsedOnConnect()
  throws java.sql.SQLException
```

接続を取得するためにドライバーが代替サーバー情報を使用した場合は、true を戻します。代替サーバー情報は、データベース・サーバーが 1 次サーバーと代替サーバーの変更時に更新する、DB2BaseDataSource 上の一時 clientRerouteServerList 情報から入手可能です。

changeDB2Password

形式:

```
public abstract void changeDB2Password(String oldPassword,
  String newPassword)
  throws java.sql.SQLException
```

Connection オブジェクトのユーザー用の、データ・ソースへのアクセス用のパスワードを変更します。

パラメーターの説明:

oldPassword
Connection のオリジナル・パスワード。

newPassword
Connection の新規パスワード。

createArrayOf

形式:

```
Array createArrayOf(String typeName,
  Object[] elements)
  throws SQLException;
```

java.sql.Array オブジェクトを作成します。

パラメーターの説明:

typeName
配列のエレメントがマップする SQL データ・タイプ。 typeName は組み込みデータ・タイプまたは特殊タイプにすることができます。

elements
Array オブジェクトを取り込むエレメント。

deregisterDB2XmlObject

形式:

```
public void deregisterDB2XmlObject(String sqlIdSchema,
  String sqlIdName)
  throws SQLException
```

データ・ソースから、事前に登録された XML スキーマを除去します。

パラメーターの説明:

sqlIdSchema
XML スキーマの SQL スキーマ名。 sqlIdSchema は最大長が 128 バイトのストリング値です。 sqlIdSchema の値は、SQL スキーマ名の命名規則に従う必要があります。名前の先頭をストリング 'SYS' にすることはできません。 sqlIdSchema の値が NULL の場合、データベース・システムは CURRENT SCHEMA 特殊レジスター内の値を使用します。

sqlIdName
XML スキーマの SQL 名。 sqlIdName は最大長が 128 バイトのストリング値です。 sqlIdName の値は SQL ID の規則に従わなければなりません。sqlIdSchema の値が NULL の場合、sqlIdName の値は NULL にすることができます。この場合、データベース・システムは sqlIdName の値を生成します。

getDB2ClientProgramId

形式:

```
public String getDB2ClientProgramId()
  throws java.sql.SQLException
```

クライアントのユーザー定義プログラム ID を戻します。プログラム ID は、データ・ソースでアプリケーションを識別するために使用できます。

getDB2ClientAccountingInformation

形式:

```
public String getDB2ClientAccountingInformation()
  throws SQLException
```

現在のクライアントのアカウンティング情報を戻します。

重要: getDB2ClientAccountingInformation は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.getClientInfo を使用してください。

getDB2ClientApplicationInformation

形式:

```
public String getDB2ClientApplicationInformation()
  throws java.sql.SQLException
```

現在のクライアントのアプリケーション情報を戻します。

重要: getDB2ClientApplicationInformation は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.getClientInfo を使用してください。

getDB2ClientUser

形式:

```
public String getDB2ClientUser()
  throws java.sql.SQLException
```

接続のための現在のクライアントのユーザー名を戻します。この名前は、JDBC 接続のユーザー値ではありません。

重要: getDB2ClientUser は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.getClientInfo を使用してください。

getDB2ClientWorkstation

形式:

```
public String getDB2ClientWorkstation()
  throws java.sql.SQLException
```

現在のクライアントの現在のクライアント・ワークステーション名を戻します。

重要: getDB2ClientWorkstation は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.getClientInfo を使用してください。

getDB2Correlator

形式:

```
String getDB2Correlator()
  throws java.sql.SQLException
```

DRDA が ACCRDB コマンドで送る、crrtkn (相関トークン) インスタンス変数の値を戻します。相関トークンは、サーバーへの論理接続を一意的に識別します。

getDB2CurrentPackagePath

形式:

```
public String getDB2CurrentPackagePath()
  throws java.sql.SQLException
```

JDBC および SQLJ パッケージが検索される DB2 パッケージ・コレクションのリストを戻します。

getDB2CurrentPackagePath メソッドは、DB2 データベース・システムへの接続にのみ適用されます。

getDB2CurrentPackageSet

形式:

```
public String getDB2CurrentPackageSet()
  throws java.sql.SQLException
```

接続のためのコレクション ID を戻します。

getDB2CurrentPackageSet メソッドは、DB2 データベース・システムへの接続にのみ適用されます。

getDB2ProgressiveStreaming

形式:

```
public int getDB2ProgressiveStreaming()
  throws java.sql.SQLException
```

接続に関する現在の連続ストリーミング設定を戻します。

戻り値は、データ・ソースで連続ストリーミングがサポートされるかどうか、progressiveStreaming プロパティーがどのように設定されているか、および DB2Connection.setProgressiveStreaming が呼び出されたかどうかに応じて異なります。

- データ・ソースで連続ストリーミングがサポートされない場合、progressiveStreaming プロパティーの設定値とは無関係に常に 2 (NO) が戻されます。
- データ・ソースで連続ストリーミングがサポートされ、DB2Connection.setProgressiveStreaming が呼び出された場合には、DB2Connection.setProgressiveStreaming で設定された値が戻り値になります。
- データ・ソースで連続ストリーミングがサポートされるものの、DB2Connection.setProgressiveStreaming が呼び出されなかった場合には、progressiveStreaming の設定値が DB2BaseDataSource.NO であれば戻り値は 2 (NO) です。 progressiveStreaming の設定値が DB2BaseDataSource.YES であるか、設定されていない場合には、戻り値は 1 (YES) です。

getDB2SecurityMechanism

形式:

```
public int getDB2SecurityMechanism()
  throws java.sql.SQLException
```

接続に有効なセキュリティー・メカニズムを戻します。

3 平文パスワード・セキュリティー

4 ユーザー ID のみのセキュリティー

7 パスワード暗号化セキュリティー

9 ユーザー ID とパスワードの暗号化セキュリティー

11 Kerberos セキュリティー

12 ユーザー ID とデータの暗号化セキュリティー

13 ユーザー ID、パスワード、およびデータの暗号化セキュリティー

15 プラグイン・セキュリティー

16 ユーザー ID のみの暗号化セキュリティー

getDB2SystemMonitor

形式:

```
public abstract DB2SystemMonitor getDB2SystemMonitor()
  throws java.sql.SQLException
```

接続のシステム・モニター・オブジェクトを戻します。 IBM Data Server Driver for JDBC and SQLJ 接続ごとに 1 つのシステム・モニターを使用できます。

getDBConcurrentAccessResolution

形式:

```
public int getDBConcurrentAccessResolution()
  throws java.sql.SQLException
```

接続に関する並行アクセス設定を戻します。並行アクセス設定は setDBConcurrentAccessResolution メソッドまたは concurrentAccessResolution プロパティーによって設定されます。

getDBProgressiveStreaming

形式:

```
public int getDB2ProgressiveStreaming()
  throws java.sql.SQLException
```

接続に関する現在の連続ストリーミング設定を戻します。

戻り値は、データ・ソースで連続ストリーミングがサポートされるかどうか、progressiveStreaming プロパティーがどのように設定されているか、および DB2Connection.setProgressiveStreaming が呼び出されたかどうかに応じて異なります。

- データ・ソースで連続ストリーミングがサポートされない場合、progressiveStreaming プロパティーの設定値とは無関係に常に 2 (NO) が戻されます。
- データ・ソースで連続ストリーミングがサポートされ、DB2Connection.setProgressiveStreaming が呼び出された場合には、DB2Connection.setProgressiveStreaming で設定された値が戻り値になります。
- データ・ソースで連続ストリーミングがサポートされるものの、DB2Connection.setProgressiveStreaming が呼び出されなかった場合には、progressiveStreaming の設定値が DB2BaseDataSource.NO であれば戻り値は 2 (NO) です。 progressiveStreaming の設定値が DB2BaseDataSource.YES であるか、設定されていない場合には、戻り値は 1 (YES) です。

getDBStatementConcentrator

形式:

```
public int getDBStatementConcentrator()
  throws java.sql.SQLException
```

接続に関するステートメント・コンセントレーター使用の設定を戻します。ステートメント・コンセントレーター使用の設定は setDBStatementConcentrator メソッドまたは statementConcentrator プロパティーによって設定されます。

getJccLogWriter

形式:

```
public PrintWriter getJccLogWriter()
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ トレースの現在のトレースの宛先を戻します。

getJccSpecialRegisterProperties

形式:

```
public java.util.Properties getJccSpecialRegisterProperties()
  throws java.sql.SQLException
```

java.util.Properties オブジェクトを戻します。その中で、キーは、ターゲット・データ・ソースでサポートされる特殊レジスターで、キー値は、それらの特殊レジスターの現行値です。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

getSavePointUniqueOption

形式:

```
public boolean getSavePointUniqueOption()
  throws java.sql.SQLException
```

固有のセーブポイントが既に設定された場合は `true` を戻します。そうでない場合は `false` を戻します。

installDB2JavaStoredProcedure

形式:

```
public void DB2Connection.installDB2JavaStoredProcedure(
  java.io.InputStream jarFile,
  int jarFileLength,
  String jarId)
  throws java.sql.SQLException
```

sqlj.install_jar ストアード・プロシージャーを DB2 Database for Linux, UNIX, and Windows サーバー上で呼び出し、そのサーバー用のカタログ内の JAR ファイルの新規定義を作成します。

パラメーターの説明:

jarFile
サーバーに定義される JAR ファイルの内容。

jarFileLength
サーバーに定義される JAR ファイルの長さ。

jarId
データベース内の JAR の名前。*schema.JAR-id* または *JAR-id* の形式になります。これは、SQL ステートメントで JAR を参照する場合に使用する名前です。 *schema* を省略した場合、データベース・システムは CURRENT SCHEMA 特殊レジスター内にある SQL 許可 ID を使用します。 JAR の所有者は、CURRENT SQLID 特殊レジスター内の許可 ID です。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

isDB2Alive

形式:

```
public boolean DB2Connection.isDB2Alive()
  throws java.sql.SQLException
```

データ・ソースへの接続のソケットが依然としてアクティブである場合は、true を戻します。

重要: isDB2Alive は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、Connection.isValid を使用してください。

isDBValid

形式:

```
public boolean DB2Connection.isDBValid(boolean throwException, int timeout)
  throws java.sql.SQLException
```

接続がまだ閉じていない状態で、有効であれば、true を戻します。そうでない場合は false を戻します。

パラメーターの説明:

throwException
接続が無効な場合に isDBValid によって SQLException がスローされるかどうかを指定します。可能な値は以下のとおりです。

true 接続が無効な場合、isDBValid は SQLException をスローします。

false *timeout* の値が 0 より小さい場合に限り、isDBValid が SQLException をスローします。

timeout
ドライバーによってサブミットされるデータベース操作の完了を待機する時間 (秒)。ドライバーは、接続を検証するために、データ・ソースに対してそのデータベース操作をサブミットします。データベース操作が完了する前にタイムアウト期間が満了した場合、isDBValid は false を戻します。値 0 は、データベース操作のタイムアウト期間が存在しないことを意味します。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

prepareDB2OptimisticLockingQuery

形式:

```
public java.sql.PreparedStatement
  DB2Connection.prepareDB2OptimisticLockingQuery(String sql,
  int returnOptimisticLockingColumns)
  throws SQLException
```

オプティミスティック・ロック情報を要求できる PreparedStatement オブジェクトを作成します。

パラメーターの説明:

sql 準備される SQL ステートメント。

returnOptimisticLockingColumns
オプティミスティック・ロック列を戻すかどうかを指定します。可能な値は以下のとおりです。

値	説明
DB2Statement.RETURN_OPTLOCK_COLUMN_NONE (0)	オプティミスティック・ロック列を戻しません。
DB2Statement.RETURN_OPTLOCK_COLUMN_ALWAYS (1)	行の変更列が単一の行を一意的に表さない場合でも、行の変更列を結果セットに追加します。この設定は、データベース準備属性 WITH ROW CHANGE COLUMNS POSSIBLY DISTINCT に相当します。
DB2Statement.RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES (2)	行の変更列が単一の行を一意的に表す場合にのみ、その行の変更列を結果セットに追加します。この設定は、データベース準備属性 WITH ROW CHANGE COLUMNS ALWAYS DISTINCT に相当します。

reconfigureDB2Connection
形式:

```
public void reconfigureDB2Connection(java.util.Properties properties)
  throws SQLException
```

接続を新規設定で再構成します。接続は、再構成前に接続プールに戻す必要はありません。このメソッドは、トランザクションの進行中に呼び出して、トラステッドまたは非トラステッド接続に使用することができます。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

パラメーターの説明:

プロパティー
接続の新規プロパティー。これらのプロパティーは、DB2Connection インスタンスですでに定義されているプロパティーをオーバーライドします。

registerDB2XmlSchema
形式:

```
public void registerDB2XmlSchema(String[] sqlIdSchema,
  String[] sqlIdName,
  String[] xmlSchemaLocations,
  InputStream[] xmlSchemaDocuments,
  int[] xmlSchemaDocumentsLengths,
  InputStream[] xmlSchemaDocumentsProperties,
  int[] xmlSchemaDocumentsPropertiesLengths,
  InputStream xmlSchemaProperties,
  int xmlSchemaPropertiesLength,
  boolean isUsedForShredding)
  throws SQLException
public void registerDB2XmlSchema(String[] sqlIdSchema,
  String[] sqlIdName,
  String[] xmlSchemaLocations,
  String[] xmlSchemaDocuments,
```

```
String[] xmlSchemaDocumentsProperties,
String xmlSchemaProperties,
boolean isUsedForShredding)
throws SQLException
```

XML スキーマを 1 つ以上の XML スキーマ文書に登録します。複数の XML スキーマ文書が registerDB2XmlSchema への 1 呼び出しで処理される場合、これらの文書は単一トランザクションの一部として処理されます。

registerDB2XmlSchema の最初のフォームは、入力ストリームから読み取られる XML スキーマ文書です。 registerDB2XmlSchema の 2 番目のフォームは、ストリングから読み取られる XML スキーマ文書です。

パラメーターの説明:

sqlIdSchema
XML スキーマの SQL スキーマ名。 sqlIdSchema 配列の最初のエレメントだけが使用されます。 sqlIdSchema は最大長が 128 バイトのストリング値です。 sqlIdSchema の値は、SQL スキーマ名の命名規則に従う必要があります。名前の先頭をストリング 'SYS' にすることはできません。
sqlIdSchema の値が NULL の場合、データベース・システムは CURRENT SCHEMA 特殊レジスター内の値を使用します。

sqlIdName
XML スキーマの SQL 名。 sqlIdName 配列の最初のエレメントだけが使用されます。 sqlIdName は最大長が 128 バイトのストリング値です。
sqlIdName の値は SQL ID の規則に従わなければなりません。 sqlIdSchema の値が NULL の場合、sqlIdName の値は NULL にすることができます。この場合、データベース・システムは sqlIdName の値を生成します。

xmlSchemaLocations
登録されるスキーマの 1 次 XML スキーマ文書の、XML スキーマ・ロケーション。 XML スキーマ・ロケーションの値は、通常は URI 形式です。各 xmlSchemaLocations 値は、最大長が 1000 バイトのストリング値です。この値は、この文書を参照する XML スキーマ文書内で指定された情報と突き合わせるためだけに使用されます。データベース・システムは、フォーマットの検査は実行せず、URI の解決を試行することもありません。

xmlSchemaDocuments
1 次 XML スキーマ文書の内容。各 xmlSchemaDocuments 値は、最大長が 30MB の String または InputStream 値です。値は NULL であってはなりません。

xmlSchemaDocumentsLengths
registerDB2XmlSchema の最初のフォームが使用される場合、xmlSchemaDocuments パラメーター内の XML スキーマ文書の長さ。各 xmlSchemaDocumentsLengths 値は int 値です。

xmlSchemaDocumentsProperties
1 次 XML スキーマ文書のプロパティーが含まれます。これは、外部 XML スキーマ・バージョン管理システムで使用されるプロパティーなどです。データベース・システムは、これらの値の内容の検証は行いません。これらの値は検索のために XSR 表に保管され、他のツールまたは XML スキーマ・リポジトリーのインプリメンテーションで使用されます。各

xmlSchemaDocumentsProperties 値は、最大長が 5MB の String または InputStream 値です。渡されるプロパティーがない場合、値は NULL です。

xmlSchemaDocumentsPropertiesLengths
registerDB2XmlSchema の最初のフォームが使用される場合、xmlSchemaDocumentsProperties パラメーター内の XML スキーマ・プロパティーの長さ。各 xmlSchemaDocumentsPropertiesLengths 値は int 値です。

xmlSchemaProperties
XML スキーマ全体のプロパティーが含まれます。これは、外部 XML スキーマ・バージョン管理システムで使用されるプロパティーなどです。データベース・システムは、この値の内容の検証は行いません。これらの値は検索のために XSR 表に保管され、他のツールまたは XML スキーマ・リポジトリーのインプリメンテーションで使用されます。 xmlSchemaProperties 値は、最大長が 5MB の String または InputStream 値です。渡されるプロパティーがない場合、値は NULL です。

xmlSchemaPropertiesLengths
registerDB2XmlSchema の最初のフォームが使用される場合、xmlSchemaProperties パラメーター内の XML スキーマ・プロパティーの長さ。 xmlSchemaPropertiesLengths 値は int 値です。

isUsedForShredding
XML 分解に使用されるアノテーションがスキーマ内にあるかどうかを示します。 isUsedForShredding は Boolean 値です。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

setDBConcurrentAccessResolution
形式:

```
public void setDBConcurrentAccessResolution(int concurrentAccessResolution)
  throws java.sql.SQLException
```

データ・ソースが Currently Commited アクセスをサポートし、アプリケーションの分離レベルがカーソル固定 (CS) または読み取り固定 (RS) の場合に、読み取りトランザクションが、書き込みトランザクションによって非互換でロックされている行のコミット済みで一貫性のあるイメージにアクセスできることを、IBM Data Server Driver for JDBC and SQLJ が要求するかどうかを指定します。このオプションは DB2 CONCURRENTACCESSRESOLUTION バインド・オプションと同じ効果を及ぼします。 setDBConcurrentAccessResolution は、setDBConcurrentAccessResolution の実行後に作成されたステートメントにのみ影響を与えます。

パラメーターの説明:

concurrentAccessResolution
以下のいずれかの整数値になります。

DB2BaseDataSource.-
CONCURRENTACCESS_USE_CURRENTLY_COMMITTED (1)
IBM Data Server Driver for JDBC and SQLJ は以下の事柄を要求します。

- データの更新時または削除時に、読み取りトランザクションは currently committed データにアクセスする。
- 読み取りトランザクションは、挿入されようとしている行をスキップする。

DB2BaseDataSource.CONCURRENTACCESS_WAIT_FOR_OUTCOME (2)
IBM Data Server Driver for JDBC and SQLJ は以下の事柄を要求します。

- 更新または削除されようとしているデータが検出された場合、読み取りトランザクションはコミットまたはロールバック操作を待機する。
- 読み取りトランザクションは、挿入されようとしている行をスキップしない。

DB2BaseDataSource.CONCURRENTACCESS_NOT_SET (0)
ロック競合が発生した場合の読み取りトランザクションに関するデータ・サーバーのデフォルト動作を使用可能にします。これはデフォルト値です。

setDBProgressiveStreaming

形式:

```
public void setDB2ProgressiveStreaming(int newSetting)
  throws java.sql.SQLException
```

接続で作成されるすべての ResultSet オブジェクトに関する連続ストリーミングを設定します。

パラメーターの説明:

newSetting
新しい連続ストリーミング設定。可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)
連続ストリーミングを有効にします。データ・ソースで連続ストリーミングがサポートされない場合、これを設定しても効果はありません。

DB2BaseDataSource.NO (2)
連続ストリーミングを無効にします。

setDBStatementConcentrator

形式:

```
public void setDBStatementConcentrator(int statementConcentratorUse)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ で、データ・ソースのステートメント・コンセントレーター機能を使用するかどうかを指定します。ステートメント・コンセントレーターは、(リテラル値を除いて) 動的ステートメント・キャッシュ内のステートメントと同じである場合にステートメントの準備を迂回する機能です。ステートメント・コンセントレーター機能は、リテラルを含み、パラメーター・マーカーを含まない SQL ステートメントにのみ適用されます。
setDBStatementConcentrator は statementConcentrator Connection または DataSource プロパティー設定値をオーバーライドします。

setDBStatementConcentrator は、setDBStatementConcentrator の実行後に作成されたステートメントにのみ影響を与えます。

パラメーターの説明:

statementConcentratorUse
以下のいずれかの整数値になります。

DB2BaseDataSource.STATEMENT_CONCENTRATOR_OFF (1)
IBM Data Server Driver for JDBC and SQLJ はデータ・ソースのステートメント・コンセントレーター機能を使用しません。

DB2BaseDataSource.STATEMENT_CONCENTRATOR_WITH_LITERALS (2)
IBM Data Server Driver for JDBC and SQLJ はデータ・ソースのステートメント・コンセントレーター機能を使用します。

DB2BaseDataSource.STATEMENT_CONCENTRATOR_NOT_SET (0)
データ・ソースは、ステートメント・コンセントレーター機能を使用するかどうかを決定します。これはデフォルト値です。

ステートメント・コンセントレーター機能をサポートする DB2 Database for Linux, UNIX, and Windows データ・ソースの場合、STMT_CONC 構成パラメーターが ON に設定されていれば、この機能が使用されます。それ以外の場合は、ステートメント・コンセントレーター機能は使用されません。

updateDB2XmlSchema
形式:

```
public void updateDB2XmlSchema(String[] targetSqlIdSchema,
  String[] targetSqlIdName,
  String[] sourceSqlIdSchema,
  String[] sourceSqlIdName,
  String[] xmlSchemaLocations,
  boolean dropSourceSchema)
  throws SQLException
```

XML スキーマの内容を、XML スキーマ・リポジトリー内の別の XML スキーマの内容で更新し、オプションでソース・スキーマをドロップします。ターゲット XML スキーマ内のスキーマ文書は、ソース XML スキーマからのスキーマ文書で置き換えられます。 updateDB2XmlSchema を呼び出す前に、ソースおよびターゲットの XML スキーマの登録を完了させておく必要があります。

ターゲット XML スキーマの更新には、SQL ALTERIN 特権が必要です。ソース XML スキーマのドロップには、SQL DROPIN 特権が必要です。

パラメーターの説明:

targetSqlIdSchema
更新される登録済み XML スキーマの SQL スキーマ名。
targetSqlIdSchema は最大長が 128 バイトのストリング値です。

targetSqlIdName
更新される登録済み XML スキーマの名前。 targetSqlIdName は最大長が 128 バイトのストリング値です。

sourceSqlIdSchema
ターゲット XML スキーマの更新に使用される、登録済み XML スキーマの SQL スキーマ名。 sourceSqlIdSchema は最大長が 128 バイトのストリング値です。

sourceSqlIdName
ターゲット XML スキーマの更新に使用される、登録済み XML スキーマの名前。 sourceSqlIdName は最大長が 128 バイトのストリング値です。

dropSourceSchema
ターゲット XML スキーマの更新後にソース XML スキーマがドロップされるかどうかを示します。 dropSourceSchema は Boolean 値です。デフォルトは `false` です。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

removeDB2JavaStoredProcedure
形式:

```
public void DB2Connection.removeDB2JavaStoredProcedure(
  String jarId)
  throws java.sql.SQLException
```

sqlj.remove_jar ストアード・プロシージャーを DB2 Database for Linux, UNIX, and Windows サーバー上で呼び出し、そのサーバー用のカタログ内の JAR ファイルの定義を削除します。

パラメーターの説明:

jarId
データベース内の JAR の名前。*schema.JAR-id* または *JAR-id* の形式になります。これは、SQL ステートメントで JAR を参照する場合に使用する名前です。 *schema* を省略した場合、データベース・システムは CURRENT SCHEMA 特殊レジスター内にある SQL 許可 ID を使用します。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

replaceDB2JavaStoredProcedure
形式:

```
public void DB2Connection.replaceDB2JavaStoredProcedure(
  java.io.InputStream jarFile,
  int jarFileLength,
  String jarId)
  throws java.sql.SQLException
```

sqlj.replace_jar ストアード・プロシージャーを DB2 Database for Linux, UNIX, and Windows サーバー上で呼び出し、そのサーバー用のカタログ内の JAR ファイルの定義を置き換えます。

パラメーターの説明:

jarFile
サーバー上で置き換えられる JAR ファイルの内容。

jarFileLength
サーバー上で置き換えられる JAR ファイルの長さ。

jarId

データベース内の JAR の名前。*schema.JAR-id* または *JAR-id* の形式になります。これは、SQL ステートメントで JAR を参照する場合に使用する名前です。 *schema* を省略した場合、データベース・システムは CURRENT SCHEMA 特殊レジスター内にある SQL 許可 ID を使用します。 JAR の所有者は、CURRENT SQLID 特殊レジスター内の許可 ID です。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

reuseDB2Connection (トラステッド接続再利用)

形式:

```
public void reuseDB2Connection(byte[] cookie,
  String user,
  String password,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
public void reuseDB2Connection(byte[] cookie,
  org.ietf.GSSCredential gssCredential,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
```

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

2 番目の形式の reuseDB2Connection は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 に対しては適用されません。

これらの形式の reuseDB2Connection は、トラステッド・アプリケーション・サーバーで使用され、新規ユーザーの代わりに、前から存在するトラステッド接続を再利用します。リセットできるプロパティー (新規ユーザー ID を含む) が渡されます。データベース・サーバーは、関連した物理接続をリセットします。reuseDB2Connection が正常に実行する場合、異なるプロパティーを持つ接続が使用可能になり、新規ユーザーが即時使用することができます。

パラメーターの説明:

cookie

JDBC ドライバーが Connection インスタンスについて生成する固有の cookie。 cookie は、初期トラステッド接続を確立したアプリケーション・サーバーおよび基礎となる JDBC ドライバーにのみ認識されます。アプリケーション・サーバーは、プール接続インスタンスの作成時にドライバーによって作成された cookie を渡します。 JDBC ドライバーは、提供された cookie が基礎となるトラステッド物理接続の cookie と一致することを調べて、そのトラステッド物理接続を確立したアプリケーション・サーバーから

発信された要求を確認します。 cookies が一致する場合、異なるプロパティーを持つ接続が使用可能になり、新規ユーザーが即時使用することができます。

user
データベース・システムがデータベース許可 ID を確立するために使用するクライアント ID。ユーザーがアプリケーション・サーバーによって認証されなかった場合、アプリケーション・サーバーは非認証ユーザーを表すクライアント ID を渡す必要があります。

password
user のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

userNameRegistry
ワークステーション・ユーザー ID を z/OS RACF ID にマップするマッピング・サービスを識別する名前。マッピング・サービスの例として、Integrated Security Services Enterprise Identity Mapping (EIM) があります。マッピング・サービスはプラグインによって定義されます。*userNameRegistry* の有効な値はプラグイン・プロバイダーによって定義されます。 *userNameRegistry* が NULL の場合、*user* のマッピングは行われません。

userSecToken
クライアントのセキュリティー・トークン。この値は、 DB2 for z/OS アカウンティング・データの一部としてトレースされます。 *userSecToken* の内容はアプリケーション・サーバーによって記述され、データベース・システムによってアプリケーション・サーバー・セキュリティー・トークンとして参照されます。

originalUser
アプリケーション・サーバーによって使用されたオリジナル・ユーザー ID。

プロパティー
再利用される接続のプロパティー。

reuseDB2Connection (再認証付きの非トラステッド再利用)
形式:

```
public void reuseDB2Connection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public void reuseDB2Connection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

最初の形式の reuseDB2Connection は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続ではサポートされません。

2 番目の形式の reuseDB2Connection は、 DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 に対しては適用されません。

異種プール環境で、これらの形式の reuseDB2Connection は、再認証後に既存の Connection インスタンスを再利用します。

パラメーターの説明:

user
接続の確立に使用される許可 ID。

password
接続の確立に使用される許可 ID のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

プロパティー
再利用される接続のプロパティー。これらのプロパティーは、DB2Connection インスタンスですでに定義されているプロパティーをオーバーライドします。

reuseDB2Connection (再認証なしの非トラステッドまたはトラステッド再利用)
形式:

```
public void reuseDB2Connection(java.util.Properties properties)
  throws java.sql.SQLException
```

再認証なしで、既存の接続インスタンスを再利用します。このメソッドは、プロパティーが変更されない場合に Connection インスタンスを再利用することを意図しています。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

このメソッドは、接続のダーティー*再利用*のためのものです。これは、オブジェクトがプールから再利用されると接続状態がリセットされないことを意味します。特殊レジスター設定およびプロパティー設定は、渡されるプロパティーによってオーバーライドされない限り、依然として有効です。グローバル一時表は削除されません。指定されなかったプロパティーは再初期化されません。分離レベル、自動コミット・モード、および読み取り専用モードなどの、JDBC 標準の一時プロパティーはすべて JDBC デフォルトにリセットされます。 `user`、`password`、`databaseName`、`serverName`、`portNumber`、`planName`、および `pkList` などの特定のプロパティーは未変更のままです。

パラメーターの説明:

プロパティー
再利用される接続のプロパティー。これらのプロパティーは、DB2Connection インスタンスですでに定義されているプロパティーをオーバーライドします。

setDB2ClientAccountingInformation
形式:

```
public void setDB2ClientAccountingInformation(String info)
  throws java.sql.SQLException
```

接続のアカウンティング情報を指定します。この情報は、クライアントのアカウンティングを目的としています。この値は、接続中に変更することもできます。

パラメーターの説明:

info
ユーザー指定のアカウンティング情報。最大長は、サーバーによって異なります。 DB2 Database for Linux, UNIX, and Windows サーバーの場合、最大長は 255 バイトです。 DB2 for z/OS サーバーの場合、最大長は 22 バイトです。 Java 空ストリング (″″) はこのパラメーター値として有効ですが、Java NULL 値は無効です。

重要: setDB2ClientAccountingInformation は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.setClientInfo を使用してください。

setDB2ClientApplicationInformation

形式:

```
public String setDB2ClientApplicationInformation(String info)
  throws java.sql.SQLException
```

現在のクライアントのアプリケーション情報を指定します。

重要: setDB2ClientApplicationInformation は、 IBM Data Server Driver for JDBC and SQLJの JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.setClientInfo を使用してください。

パラメーターの説明:

info
ユーザー指定のアプリケーション情報。最大長は、サーバーによって異なります。 DB2 Database for Linux, UNIX, and Windows サーバーの場合、最大長は 255 バイトです。 DB2 for z/OS サーバーの場合、最大長は 32 バイトです。 Java 空ストリング (″″) はこのパラメーター値として有効ですが、Java NULL 値は無効です。

setDB2ClientDebugInfo

形式:

```
public void setDB2ClientDebugInformation(String debugInfo)
  throws java.sql.SQLException
public void setDB2ClientDebugInformation(String mgrInfo,
  String traceInfo)
  throws java.sql.SQLException
```

CLIENT DEBUGINFO 接続属性の値を設定します。これはデータベース・システムに、接続を使用しているストアード・プロシージャーおよびユーザー定義関数がデバッグ・モードで実行していることを通知します。 CLIENT DEBUGINFO は、DB2 統合デバッガーにより使用されます。 CLIENT DEBUGINFO ストリング全体を設定するには、最初の形式を使用してください。 CLIENT DEBUGINFO ストリング内のセッション・マネージャーおよびトレース情報のみを変更するには、2 番目の形式を使用してください。

setDB2ClientDebugInfo メソッドは、 DB2 for z/OS データベース・システムへの接続にのみ適用されます。

CLIENT DEBUGINFO 属性を、0 より長いストリングに設定する場合は、以下のいずれかの特権が必要です。

- DEBUGSESSION 特権
- SYSADM 権限

パラメーターの説明:

debugInfo
以下の形式の、最大で 254 バイトのストリング。

```
Mip:port,Iip,Ppid,Ttid,Cid,Llvl
```

ストリングの各部の意味は、以下のとおりです。

M*ip:port*
セッション・マネージャーの IP アドレスおよびポート番号。

I*ip* クライアント IP アドレス

P*pid* クライアント・プロセス ID

T*tid* クライアント・スレッド ID (オプション)

C*id* データ接続生成済み ID

L*lvl* デバッグ・ライブラリー診断トレース・レベル。

以下に例を示します。

```
M9.72.133.89:8355,I9.72.133.89,P4552,T123,C1,L0
```

このストリングの詳細記述については、SET CLIENT DEBUGINFO の説明を参照してください。

mgrInfo
以下の形式のストリング。これは統合デバッガー・セッション・マネージャーの IP アドレスおよびポート番号を指定します。

```
Mip:port
```

以下に例を示します。

```
M9.72.133.89:8355
```

このストリングの詳細記述については、SET CLIENT DEBUGINFO の説明を参照してください。

trcInfo
以下の形式のストリング。これはデバッグ・ライブラリー診断トレース・レベルを指定します。

```
Llvl
```

以下に例を示します。

```
L0
```

このストリングの詳細記述については、SET CLIENT DEBUGINFO の説明を参照してください。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続には適用されません。

setDB2ClientProgramId

形式:

```
public abstract void setDB2ClientProgramId(String program-ID)
  throws java.sql.SQLException
```

DB2 for z/OS サーバー上で、接続用のユーザー定義プログラム ID を設定します。このプログラム ID は、呼び出し元を識別するために使用される 80 バイトのストリングです。 DB2 for z/OS サーバーは、ストリングを IFCID 316 トレース・レコード内に他の統計と共に配置するので、どのプログラムが特定の SQL ステートメントと関連しているかを識別できます。

setDB2ClientUser

形式:

```
public void setDB2ClientUser(String user)
  throws java.sql.SQLException
```

接続のための現行クライアントのユーザー名を指定します。この名前はクライアントのアカウンティングを目的としており、JDBC 接続のユーザー値ではありません。 JDBC 接続のユーザーの場合とは異なり、現在のクライアントのユーザー名は接続時に変更することができます。

パラメーターの説明:

user
現在のクライアントのユーザー ID。最大長は、サーバーによって異なります。 DB2 Database for Linux, UNIX, and Windows サーバーの場合、最大長は 255 バイトです。 DB2 for z/OS サーバーの場合、最大長は 16 バイトです。 Java 空ストリング (″″) はこのパラメーター値として有効ですが、Java NULL 値は無効です。

重要: getDB2ClientUser は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.getClientInfo を使用してください。

setDB2ClientWorkstation

形式:

```
public void setDB2ClientWorkstation(String name)
  throws java.sql.SQLException
```

接続のための現在のクライアントのワークステーション名を指定します。この名前は、クライアントのアカウンティングを目的としています。現在のクライアントのワークステーション名は、接続時に変更することができます。

パラメーターの説明:

name
現在のクライアントのワークステーション名。最大長は、サーバーによって異なります。 DB2 Database for Linux, UNIX, and Windows サーバーの場合、最大長は 255 バイトです。 DB2 for z/OS サーバーの場合、最大長は 18 バイトです。 Java 空ストリング (″″) はこのパラメーター値として有効ですが、Java NULL 値は無効です。

重要: getDB2ClientWorkstation は、 IBM Data Server Driver for JDBC and SQLJ の JDBC 4.0 インプリメンテーションでは推奨されていません。代わりに、java.sql.Connection.getClientInfo を使用してください。

setDB2CurrentPackagePath

形式:

```
public void setDB2CurrentPackagePath(String packagePath)
  throws java.sql.SQLException
```

データベース・システムが JDBC および SQLJ パッケージを検索するコレクション ID のリストを指定します。

setDB2CurrentPackagePath メソッドは、DB2 データベース・システムへの接続にのみ適用されます。

パラメーターの説明:

packagePath
コレクション ID の、コンマで区切られたリスト。

setDB2CurrentPackageSet

形式:

```
public void setDB2CurrentPackageSet(String packageSet)
  throws java.sql.SQLException
```

接続のためのコレクション ID を指定します。この値を設定すると、接続に使用される IBM Data Server Driver for JDBC and SQLJ インスタンスのコレクション ID も設定されます。

setDB2CurrentPackageSet メソッドは、DB2 データベース・システムへの接続にのみ適用されます。

パラメーターの説明:

packageSet
接続のためのコレクション ID。 packageSet 値の最大長は 18 バイトです。このメソッドは、SQL SET CURRENT PACKAGESET ステートメントの実行の代替方法としてプログラム内で呼び出すことができます。

setDB2ProgressiveStreaming

形式:

```
public void setDB2ProgressiveStreaming(int newSetting)
  throws java.sql.SQLException
```

接続で作成されるすべての ResultSet オブジェクトに関する連続ストリーミングを設定します。

パラメーターの説明:

newSetting
新しい連続ストリーミング設定。可能な値は以下のとおりです。

DB2BaseDataSource.YES (1)
連続ストリーミングを有効にします。データ・ソースで連続ストリーミングがサポートされない場合、これを設定しても効果はありません。

DB2BaseDataSource.NO (2)
連続ストリーミングを無効にします。

setJccLogWriter
形式:

```
public void setJccLogWriter(PrintWriter logWriter)
  throws java.sql.SQLException

public void setJccLogWriter(PrintWriter logWriter, int traceLevel)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ トレースを有効または無効にします。または、アクティブな接続中にトレースの宛先を変更します。

パラメーターの説明:

logWriter
IBM Data Server Driver for JDBC and SQLJ がトレース出力を書き込む、タイプ java.io.PrintWriter のオブジェクト。トレースをオフにするには、*logWriter* の値を `null` に設定します。

traceLevel
収集するトレースのタイプを指定します。有効な値については、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』の traceLevel プロパティーに関する説明を参照してください。

setSavePointUniqueOption
形式:

```
public void setSavePointUniqueOption(boolean flag)
  throws java.sql.SQLException
```

アプリケーションがリカバリー単位内のセーブポイント名を再使用できるかどうかを指定します。可能な値は以下のとおりです。

true Connection.setSavepoint(savepoint-name) メソッドでは、同じリカバリー単位の中で *savepoint-name* に同じ値を複数回にわたって指定することができません。

false Connection.setSavepoint(savepoint-name) メソッドでは、同じリカバリー単位の中で *savepoint-name* に同じ値を複数回にわたって指定できます。

`false` を指定した場合、Connection.setSavepoint(savepoint-name) メソッドの実行時に *savepoint-name* という名前のセーブポイントがリカバリー単位に既に存在していれば、データベース・マネージャーは既存のセーブポイントを破棄して *savepoint-name* という名前の新しいセーブポイントを作成します。

セーブポイントの再使用は、Connection.releaseSavepoint(savepoint-name) の実行と同じではありません。 Connection.releaseSavepoint(savepoint-name) は、*savepoint-name* およびそれ以降に設定されたセーブポイントをすべて解放します。

DB2ConnectionPoolDataSource クラス

DB2ConnectionPoolDataSource は PooledConnection オブジェクトのファクトリーです。このインターフェースをインプリメントするオブジェクトは、Java Naming and Directory Interface (JNDI) に基づくネーム・サービスで登録されます。

com.ibm.db2.jcc.DB2ConnectionPoolDataSource クラスは
com.ibm.db2.jcc.DB2BaseDataSource クラスを拡張し、
javax.sql.ConnectionPoolDataSource、java.io.Serializable、および
javax.naming.Referenceable インターフェースをインプリメントします。

DB2ConnectionPoolDataSource のプロパティー

これらのプロパティーは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。これらのプロパティーの説明については、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』を参照してください。

これらのプロパティーには、プロパティーの値を設定する setXXX メソッド、および値を検索する getXXX メソッドがあります。 setXXX メソッドの形式は以下のとおりです。

```
void setProperty-name(data-type property-value)
```

getXXX メソッドの形式は以下のとおりです。

```
data-type getProperty-name()
```

Property-name は、先頭文字が大文字になっている非修飾プロパティー名です。

以下の表に、 IBM Data Server Driver for JDBC and SQLJ のプロパティーとそのデータ・タイプをリストします。

表 100. DB2ConnectionPoolDataSource のプロパティーとそれらのデータ・タイプ

プロパティー名	データ・タイプ
com.ibm.db2.jcc.DB2ConnectionPoolDataSource.maxStatements	int

DB2ConnectionPoolDataSource のメソッド

getDB2PooledConnection

形式:

```
public DB2PooledConnection getDB2PooledConnection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2PooledConnection getDB2PooledConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

異種プール環境での初期非トラステッド接続を確立します。

最初のフォーム getDB2PooledConnection はユーザー ID とパスワードを提供します。 2 番目のフォーム getDB2PooledConnection は、Kerberos セキュリティーを使用する接続のためのものです。

パラメーターの説明:

user
接続の確立に使用される許可 ID。

password
接続の確立に使用される許可 ID のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

プロパティー
接続のプロパティー。

getDB2TrustedPooledConnection

形式:

```
public Object[] getDB2TrustedPooledConnection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedPooledConnection(
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedPooledConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

システム許可 ID を使用するアプリケーション・サーバーは、トラステッド接続を確立するためにこのメソッドを使用します。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

以下のエレメントは Object[] で戻されます。

- 最初のエレメントは、トラステッド DB2PooledConnection インスタンスです。
- 2 番目のエレメントは、生成されたプール接続インスタンスの固有の cookie です。

最初のフォーム getDB2TrustedPooledConnection はユーザー ID とパスワードを提供し、2 番目のフォーム getDB2TrustedPooledConnection は、DB2ConnectionPoolDataSource オブジェクトのユーザー ID とパスワードを使用します。 3 番目のフォーム getDB2TrustedPooledConnection は、Kerberos セキュリティーを使用する接続のためのものです。

パラメーターの説明:

user
データベース・サーバーへのトラステッド接続の確立に使用される DB2 許可 ID。

password
トラステッド接続の確立に使用される許可 ID のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

プロパティー
接続のプロパティー。

DB2DatabaseMetaData インターフェース

com.ibm.db2.jcc.DB2DatabasemetaData は java.sql.DatabaseMetaData インターフェースを拡張するものです。

DB2DatabaseMetaData は java.sql.Wrapper インターフェースをインプリメントします。

DB2DatabaseMetaData のメソッド:

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

isIDSDatabaseAnsiCompliant
形式:

```
public boolean isIDSDatabaseAnsiCompliant();
```

アクティブな IBM Informix Dynamic Server (IDS) データベースが ANSI 準拠である場合、true を戻します。そうでない場合は false を戻します。

ANSI 準拠データベースとは、WITH LOG MODE ANSI オプションを指定して作成されたデータベースのことです。

このメソッドは、IDS データ・ソースへの接続にのみ適用されます。データ・ソースが IDS データ・ソースでない場合、SQLException がスローされます。

isIDSDatabaseLogging
形式:

```
public boolean isIDSDatabaseLogging();
```

現在のアクティブな IDS データベースがロギングをサポートする場合、true を戻します。そうでない場合は false を戻します。

ロギングをサポートする IDS データベースとは、WITH LOG MODE ANSI、WITH BUFFERED LOG、または WITH LOG オプションを指定して作成されたデータベースです。

このメソッドは、IDS データ・ソースへの接続にのみ適用されます。データ・ソースが IDS データ・ソースでない場合、SQLException がスローされます。

isResetRequiredForDB2eWLM
形式:

```
public boolean isResetRequiredForDB2eWLM();
```

ターゲット・データベース・サーバーが eWLM をサポートするためにクリーンな再利用を必要とする場合は、true を戻します。そうでない場合は false を戻します。

supportsDB2ProgressiveStreaming

形式:

```
public boolean supportsDB2ProgressiveStreaming();
```

ターゲット・データ・ソースが連続ストリーミングをサポートする場合は、true を戻します。そうでない場合は false を戻します。

DB2Diagnosable インターフェース

com.ibm.db2.jcc.DB2Diagnosable インターフェースは、SQLException からの DB2 診断を取得するメカニズムを提供します。

DB2Diagnosable のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getSqlca

形式:

```
public DB2Sqlca getSqlca();
```

IBM Data Server Driver for JDBC and SQLJ の下で作成された java.sql.Exception からの DB2Sqlca オブジェクトを戻します。

getThrowable

形式:

```
public Throwable getThrowable();
```

IBM Data Server Driver for JDBC and SQLJ の下で作成された java.sql.Exception からの java.lang.Throwable オブジェクトを戻します。

printTrace

形式:

```
static public void printTrace(java.io.PrintWriter printWriter,
  String header);
```

IBM Data Server Driver for JDBC and SQLJ の下で java.sql.Exception がスローされた後に診断情報を印刷します。

パラメーターの説明:

printWriter
診断情報の宛先。

header
出力の先頭に印刷されるユーザー定義情報。

DB2ExceptionFormatter クラス

com.ibm.db2.jcc.DB2ExceptionFormatter クラスには、診断情報をストリームに印刷するためのメソッドが含まれます。

DB2ExceptionFormatter のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

printTrace

形式:

```
static public void printTrace(java.sql.SQLException sqlException,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(DB2Sqlca sqlca,
  java.io.PrintWriter printWriter, String header)

static public void printTrace(java.lang.Throwable throwable,
  java.io.PrintWriter printWriter, String header)
```

例外がスローされた後に診断情報を印刷します。

パラメーターの説明:

sqlException|sqlca|throwable
前の JDBC または Java 操作中にスローされた例外。

printWriter
診断情報の宛先。

header
出力の先頭に印刷されるユーザー定義情報。

DB2JCCPlugin クラス

com.ibm.db2.jcc.DB2JCCPlugin クラスは、 DB2 Database for Linux, UNIX, and Windows プラグイン・サポートを提供するためにインプリメント可能なメソッドを定義する、抽象クラスです。このクラスは DB2 Database for Linux, UNIX, and Windows だけに適用されます。

DB2JCCPlugin のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getTicket

形式:

```
public abstract byte[] getTicket(String user,
  String password,
  byte[] returnedToken)
  throws org.ietf.jgss.GSSException
```

ユーザー用の Kerberos チケットを検索します。

パラメーターの説明:

user
Kerberos チケットが検索されるユーザー ID。

password
user のパスワード。

returnedToken

DB2PooledConnection クラス

com.ibm.db2.jcc.DB2PooledConnection クラスは、以前から存在するトラステッド接続でユーザーを切り替えるためにアプリケーション・サーバーが使用できるメソッドを提供します。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

DB2PooledConnection のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getConnection (再認証なしの非トラステッドまたはトラステッド再利用)

形式:

```
public DB2Connection getConnection()
  throws java.sql.SQLException
```

このメソッドは、接続のダーティー*再利用*のためのものです。これは、オブジェクトがプールから再利用されると接続状態がリセットされないことを意味します。特殊レジスター設定およびプロパティー設定は、渡されるプロパティーによってオーバーライドされない限り、依然として有効です。グローバル一時表は削除されません。指定されなかったプロパティーは再初期化されません。分離レベル、自動コミット・モード、および読み取り専用モードなどの、JDBC 標準の一時プロパティーはすべて JDBC デフォルトにリセットされます。 `user`、`password`、`databaseName`、`serverName`、`portNumber`、`planName`、および `pkList` などの特定のプロパティーは未変更のままです。

getDB2Connection (トラステッド再利用)

形式:

```
public DB2Connection getDB2Connection(byte[] cookie,
  String user,
  String password,
  String userRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
public Connection getDB2Connection(byte[] cookie,
  org.ietf.GSSCredential gssCredential,
  String usernameRegistry,
  byte[] userSecToken,
  String originalUser,
  java.util.Properties properties)
  throws java.sql.SQLException
```

トラステッド接続に関連したユーザーを認証なしで切り替えます。

getDB2Connection の 2 番目の形式は、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続でのみサポートされます。

パラメーターの説明:

cookie
JDBC ドライバーが Connection インスタンスについて生成する固有の cookie。 cookie は、初期トラステッド接続を確立したアプリケーション・サーバーおよび基礎となる JDBC ドライバーにのみ認識されます。アプリケーション・サーバーは、プール接続インスタンスの作成時にドライバーによって作成された cookie を渡します。 JDBC ドライバーは、提供された cookie が基礎となるトラステッド物理接続の cookie と一致することを調べて、そのトラステッド物理接続を確立したアプリケーション・サーバーから発信された要求を確認します。 cookie が一致する場合、異なるプロパティーを持つ接続が使用可能になり、新規ユーザーは即時使用することができます。

user
データ・ソースがデータベース・サーバーの許可 ID を確立するために使用するクライアント ID。ユーザーがアプリケーション・サーバーによって認証されなかった場合、アプリケーション・サーバーは非認証ユーザーを表すユーザー ID を渡す必要があります。

password
user のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

userNameRegistry
ワークステーション・ユーザー ID を z/OS RACF ID にマップするマッピング・サービスを識別する名前。マッピング・サービスの例として、Integrated Security Services Enterprise Identity Mapping (EIM) があります。マッピング・サービスはプラグインによって定義されます。*userNameRegistry* の有効な値はプラグイン・プロバイダーによって定義されます。 *userNameRegistry* が NULL の場合、接続はマッピング・サービスを使用しません。

userSecToken
クライアントのセキュリティー・トークン。この値は、DB2 for z/OS アカウンティング・データの一部としてトレースされます。*userSecToken* の内容はアプリケーション・サーバーによって記述され、データ・ソースによってアプリケーション・サーバー・セキュリティー・トークンとして参照されます。

originalUser
オリジナル要求をアプリケーション・サーバーに送信するクライアント ID。 *originalUser* は、アプリケーション・サーバーによって使用されたオリジナル・ユーザー ID として、DB2 for z/OS アカウンティング・データに組み込まれます。

プロパティー
再利用される接続のプロパティー。これらのプロパティーは、DB2PooledConnection インスタンスですでに定義されているプロパティーをオーバーライドします。

getDB2Connection (再認証付きの非トラステッド再利用)

形式:

```
public DB2Connection getDB2Connection(
  String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2Connection getDB2Connection(org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

非トラステッド接続に関連したユーザーを認証付きで切り替えます。

最初のフォーム getDB2Connection はユーザー ID とパスワードを提供します。2 番目のフォーム getDB2Connection は、Kerberos セキュリティーを使用する接続のためのものです。

パラメーターの説明:

user
データ・ソースがデータベース・サーバーの許可 ID を確立するために使用するユーザー ID。

password
user のパスワード。

プロパティー
再利用される接続のプロパティー。これらのプロパティーは、DB2PooledConnection インスタンスですでに定義されているプロパティーをオーバーライドします。

getDB2Connection (再認証なしの非トラステッドまたはトラステッド再利用)

形式:

```
public java.sql.Connection getDB2Connection(
  java.util.Properties properties)
  throws java.sql.SQLException
```

再認証なしで、非トラステッド接続を再利用します。

このメソッドは、接続のダーティー*再利用*のためのものです。これは、オブジェクトがプールから再利用されると接続状態がリセットされないことを意味します。特殊レジスター設定およびプロパティー設定は、渡されるプロパティーによってオーバーライドされない限り、依然として有効です。グローバル一時表は削除されません。指定されなかったプロパティーは再初期化されません。分離レベル、自動コミット・モード、および読み取り専用モードなどの、JDBC 標準の一時プロパティーはすべて JDBC デフォルトにリセットされます。 `user`、`password`、`databaseName`、`serverName`、`portNumber`、`planName`、および `pkList` などの特定のプロパティーは未変更のままです。

パラメーターの説明:

プロパティー
再利用される接続のプロパティー。これらのプロパティーは、DB2PooledConnection インスタンスですでに定義されているプロパティーをオーバーライドします。

DB2PoolMonitor クラス

com.ibm.db2.jcc.DB2PoolMonitor クラスでは、接続コンセントレーターと Sysplex ワークロード・バランシングで使用されるグローバル・トランスポート・オブジェクト・プールをモニターするメソッドが提供されます。

DB2PoolMonitor フィールド

以下のフィールドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

public static final int TRANSPORT_OBJECT = 1
この値は DB2PoolMonitor.getPoolMonitor メソッドのパラメーターです。

DB2PoolMonitor メソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

agedOutObjectCount
形式:

```
public abstract int agedOutObjectCount()
```

db2.jcc.maxTransportObjectIdleTime で指定されていたアイドル時間を超過したためにプールから削除されたオブジェクトの数を取得します。

createdObjectCount
形式:

```
public abstract int createdObjectCount()
```

プールの作成時以降に IBM Data Server Driver for JDBC and SQLJ により作成されたオブジェクトの数を取得します。

getMonitorVersion
形式:

```
public int getMonitorVersion()
```

IBM Data Server Driver for JDBC and SQLJ に付属している DB2PoolMonitor クラスのバージョンを取得します。

getPoolMonitor
形式:

```
public static DB2PoolMonitor getPoolMonitor(int monitorType)
```

DB2PoolMonitor クラスのインスタンスを取得します。

パラメーターの説明:

monitorType
モニター・タイプ。この値は DB2PoolMonitor.TRANSPORT_OBJECT でなければなりません。

heavyWeightReusedObjectCount
形式:

```
public abstract int heavyWeightReusedObjectCount()
```

プールから再利用されたオブジェクトの数を取得します。

lightWeightReusedObjectCount

形式:

```
public abstract int lightWeightReusedObjectCount()
```

再利用されたがプール内に存在しなかったオブジェクトの数を取得します。これは、接続オブジェクトによってトランスポート・オブジェクトがトランザクション境界で解放された場合に発生することがあります。この接続オブジェクトで、そのトランスポート・オブジェクトが後で必要になった際に、元のトランスポート・オブジェクトが他のどの接続オブジェクトによってもまだ使用されていない場合は、この接続オブジェクトでそのトランスポート・オブジェクトを使用することができます。

longestBlockedRequestTime

形式:

```
public abstract long longestBlockedRequestTime()
```

要求がブロックされた最長の時間をミリ秒単位で取得します。

numberOfConnectionReleaseRefused

形式:

```
public abstract int numberOfConnectionReleaseRefused()
```

接続の解除が拒否された回数を取得します。

numberOfRequestsBlocked

形式:

```
public abstract int numberOfRequestsBlocked()
```

プールに対して IBM Data Server Driver for JDBC and SQLJ から出されたが、プールがその最大容量に達したためプールによってブロックされた要求の数を取得します。ブロックされた要求が正常に完了する可能性があるのは、db2.jcc.maxTransportObjectWaitTime を超過して例外がスローされる前にオブジェクトがプールに返された場合です。

numberOfRequestsBlockedDataSourceMax

形式:

```
public abstract int numberOfRequestsBlockedDataSourceMax()
```

プールに対して IBM Data Server Driver for JDBC and SQLJから出されたが、プールが DataSource オブジェクトの最大容量に達したためプールによってブロックされた要求の数を取得します。

numberOfRequestsBlockedPoolMax

形式:

```
public abstract int numberOfRequestsBlockedPoolMax()
```

プールに対して IBM Data Server Driver for JDBC and SQLJ から出されたが、プールの最大数に達したためプールによってブロックされた要求の数を取得します。

removedObjectCount

形式:

```
public abstract int removedObjectCount()
```

プールの作成時以降にプールから削除されたオブジェクトの数を取得します。

shortestBlockedRequestTime

形式:

```
public abstract long shortestBlockedRequestTime()
```

要求がブロックされた最短の時間をミリ秒単位で取得します。

successfullRequestsFromPool

形式:

```
public abstract int successfullRequestsFromPool()
```

プールの作成時以降にプールに対して IBM Data Server Driver for JDBC and SQLJ から出されて正常に完了した要求の数を取得します。正常に完了した要求とは、プールからオブジェクトが返されたことを意味します。

totalPoolObjects

形式:

```
public abstract int totalPoolObjects()
```

プール内に現在存在するオブジェクトの数を取得します。

totalRequestsToPool

形式:

```
public abstract int totalRequestsToPool()
```

プールの作成時以降にプールに対して IBM Data Server Driver for JDBC and SQLJ から出された要求の総数を取得します。

totalTimeBlocked

形式:

```
public abstract long totalTimeBlocked()
```

プールによってブロックされた要求のミリ秒単位の合計時間を取得します。アプリケーションによって複数のスレッドが使用される場合は、この時間は、アプリケーションの経過実行時間よりはるかに長くなる可能性があります。

DB2PreparedStatement インターフェース

com.ibm.db2.jcc.DB2PreparedStatement インターフェースは、com.ibm.db2.jcc.DB2Statement および java.sql.PreparedStatement インターフェースを拡張するものです。

DB2PreparedStatement のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

executeDB2QueryBatch

形式:

```
public void executeDB2QueryBatch()
  throws java.sql.SQLException
```

パラメーターが指定された照会を含むステートメント・バッチを実行します。

この方法は、 IBM Informix Dynamic Server データ・ソースへの接続の場合はサポートされません。

getDBGeneratedKeys

形式:

```
public java.sql.ResultSet[] getDBGeneratedKeys()
  throws java.sql.SQLException
```

INSERT ステートメントがバッチで実行されたときに作成された自動生成キーを取り出します。戻される ResultSet オブジェクトにはそれぞれ、バッチの 1 つのステートメントに対する自動生成キーが含まれます。

getDBGeneratedKeys は以下の条件下で長さ 0 の配列を戻します。

- getDBGeneratedKeys が順序を外れて呼び出される。例えば、getDBGeneratedKeys が executeBatch の前に呼び出されると、長さ 0 の配列が戻されます。
- バッチで実行される PreparedStatement が、以下のメソッドのいずれかを使って作成されなかった。

  ```
  Connection.prepareStatement(String sql, int[] autoGeneratedKeys)
  Connection.prepareStatement(String sql, String[] autoGeneratedColumnNames)
  Connection.prepareStatement(String sql, Statement.RETURN_GENERATED_KEYS)
  ```

上記リストのいずれかのメソッドを使って作成された PreparedStatement に対して getDBGeneratedKeys が呼び出され、PreparedStatement がバッチにない場合、単一の ResultSet が戻されます。

setJccArrayAtName

形式:

```
public void setJccArrayAtName(String parameterMarkerName,
  java.sql.Array x)
  throws java.sql.SQLException
```

名前付きパラメーターに java.sql.Array 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.Array 値。

setJccAsciiStreamAtName

形式:

IBM Data Server Driver for JDBC and SQLJ バージョン 3.57 以降のサポート:

```
public void setJccAsciiStreamAtName(String parameterMarkerName,
  java.io.InputStream x, int length)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ バージョン 4.7 以降のサポート:

```
public void setJccAsciiStreamAtName(String parameterMarkerName,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccAsciiStreamAtName(String parameterMarkerName,
  java.io.InputStream x, long length)
  throws java.sql.SQLException
```

名前付きパラメーターに、java.io.InputStream 内の ASCII 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる ASCII java.io.InputStream 値。

length
パラメーター・マーカーに割り当てられる java.io.InputStream 値の長さ (バイト数)。

setJccBigDecimalAtName

形式:

```
public void setJccBigDecimalAtName(String parameterMarkerName,
  java.math.BigDecimal x)
  throws java.sql.SQLException
```

名前付きパラメーターに java.math.BigDecimal 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.math.BigDecimal 値。

setJccBinaryStreamAtName

形式:

IBM Data Server Driver for JDBC and SQLJ バージョン 3.57 以降のサポート:

```
public void setJccBinaryStreamAtName(String parameterMarkerName,
  java.io.InputStream x, int length)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ バージョン 4.7 以降のサポート:

```
public void setJccBinaryStreamAtName(String parameterMarkerName,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccBinaryStreamAtName(String parameterMarkerName,
  java.io.InputStream x, long length)
  throws java.sql.SQLException
```

名前付きパラメーターに、java.io.InputStream 内のバイナリー値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられるバイナリー java.io.InputStream 値。

length
パラメーター・マーカーに割り当てられる java.io.InputStream 値のバイト数。

setJccBlobAtName

形式:

IBM Data Server Driver for JDBC and SQLJ バージョン 3.57 以降のサポート:

```
public void setJccBlobAtName(String parameterMarkerName,
  java.sql.Blob x)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ バージョン 4.7 以降のサポート:

```
public void setJccBlobAtName(String parameterMarkerName,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccBlobAtName(String parameterMarkerName,
  java.io.InputStream x, long length)
  throws java.sql.SQLException
```

名前付きパラメーターに BLOB 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.Blob 値または java.io.InputStream 値。

length
パラメーター・マーカーに割り当てられる java.io.InputStream 値のバイト数。

setJccBooleanAtName

形式:

```
public void setJccBooleanAtName(String parameterMarkerName,
  boolean x)
  throws java.sql.SQLException
```

名前付きパラメーターに Boolean 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる Boolean 値。

setJccByteAtName

形式:

```
public void setJccByteAtName(String parameterMarkerName,
  byte x)
  throws java.sql.SQLException
```

名前付きパラメーターにバイト値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられるバイト値。

setJccBytesAtName

形式:

```
public void setJccBytesAtName(String parameterMarkerName,
  byte[] x)
  throws java.sql.SQLException
```

バイト値から成る配列を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられるバイト配列。

setJccCharacterStreamAtName

形式:

IBM Data Server Driver for JDBC and SQLJ バージョン 3.57 以降のサポート:

```
public void setJccCharacterStreamAtName(String parameterMarkerName,
  java.io.Reader x, int length)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ バージョン 4.7 以降のサポート:

```
public void setJccCharacterStreamAtName(String parameterMarkerName,
  java.io.Reader x)
  throws java.sql.SQLException
public void setJccCharacterStreamAtName(String parameterMarkerName,
  java.io.Reader x, long length)
  throws java.sql.SQLException
```

名前付きパラメーターに、java.io.Reader 内の Unicode 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる Unicode java.io.Reader 値。

length
パラメーター・マーカーに割り当てられる java.io.InputStream 値の文字数。

setJccClobAtName

形式:

IBM Data Server Driver for JDBC and SQLJ バージョン 3.57 以降のサポート:

```
public void setJccClobAtName(String parameterMarkerName,
  java.sql.Blob x)
  throws java.sql.SQLException
```

IBM Data Server Driver for JDBC and SQLJ バージョン 4.7 以降のサポート:

```
public void setJccClobAtName(String parameterMarkerName,
  java.io.InputStream x)
  throws java.sql.SQLException
public void setJccClobAtName(String parameterMarkerName,
  java.io.InputStream x, long length)
  throws java.sql.SQLException
```

名前付きパラメーターに CLOB 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.Clob 値または java.io.Reader 値。

length
パラメーター・マーカーに割り当てられる java.io.InputStream 値のバイト数。

setJccDateAtName

形式:

```
public void setJccDateAtName(String parameterMarkerName,
  java.sql.Date x)
  throws java.sql.SQLException
public void setJccDateAtName(String parameterMarkerName,
  java.sql.Date x,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

名前付きパラメーターに java.sql.Date 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.Date 値。

cal 日付を構成するために IBM Data Server Driver for JDBC and SQLJ によって使用される java.util.Calendar オブジェクト。

setJccDoubleAtName

形式:

```
public void setJccDoubleAtName(String parameterMarkerName,
  double x)
  throws java.sql.SQLException
```

double タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる double タイプの値。

setJccFloatAtName

形式:

```
public void setJccFloatAtName(String parameterMarkerName,
  double x)
  throws java.sql.SQLException
```

float タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる float タイプの値。

setJccIntAtName

形式:

```
public void setJccIntAtName(String parameterMarkerName,
  int x)
  throws java.sql.SQLException
```

int タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる int タイプの値。

setJccLongAtName

形式:

```
public void setJccLongAtName(String parameterMarkerName,
  long x)
  throws java.sql.SQLException
```

long タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる long タイプの値。

setJccNullAtName

形式:

```
public void setJccNullAtName(String parameterMarkerName,
  int jdbcType)
  throws java.sql.SQLException
public void setJccNullAtName(String parameterMarkerName,
  int jdbcType,
  String typeName)
  throws java.sql.SQLException
```

名前付きパラメーターに SQL NULL 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

jdbcType
java.sql.Types で定義された、パラメーター・マーカーに割り当てられる NULL 値の JDBC タイプ・コード。

typeName
jdbcType が java.sql.Types.DISTINCT または java.sql.Types.REF である場合、パラメーター・マーカーに割り当てられる NULL 値の SQL ユーザー定義タイプの完全修飾名。

setJccObjectAtName

形式:

```
public void setJccObjectAtName(String parameterMarkerName,
  java.sql.Object x)
  throws java.sql.SQLException
public void setJccObjectAtName(String parameterMarkerName,
  java.sql.Object x,
  int targetJdbcType)
  throws java.sql.SQLException
public void setJccObjectAtName(String parameterMarkerName,
  java.sql.Object x,
  int targetJdbcType,
  int scale)
  throws java.sql.SQLException
```

java.lang.Object タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる Object タイプの値。

targetJdbcType
データ・ソースに送られるときに入力値に割り当てられる、java.sql.Types で定義されるデータ・タイプ。

scale
パラメーター・マーカーに割り当てられる値のスケール。このパラメーターは、以下の場合にのみ適用されます。

- *targetJdbcType* が java.sql.Types.DECIMAL または java.sql.Types.NUMERIC である場合、*scale* は小数点の右側の桁数です。
- *x* のタイプが java.io.InputStream または java.io.Reader である場合、*scale* は Stream または Reader オブジェクト内のデータの長さです。

setJccShortAtName

形式:

```
public void setJccShortAtName(String parameterMarkerName,
  long x)
  throws java.sql.SQLException
```

short タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる short タイプの値。

setJccSQLXMLAtName

形式:

```
public void setJccSQLXMLAtName(String parameterMarkerName,
  long x)
  throws java.sql.SQLException
```

java.sql.SQLXML タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.SQLXML タイプの値。

setJccStringAtName

形式:

```
public void setJccStringAtName(String parameterMarkerName,
  long x)
  throws java.sql.SQLException
```

String タイプの値を名前付きパラメーターに割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる String タイプの値。

setJccTimeAtName

形式:

```
public void setJccTimeAtName(String parameterMarkerName,
  java.sql.Time x)
  throws java.sql.SQLException
public void setJccTimeAtName(String parameterMarkerName,
  java.sql.Time x,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

名前付きパラメーターに java.sql.Time 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.Time 値。

cal 時間を構成するために IBM Data Server Driver for JDBC and SQLJ によって使用される java.util.Calendar オブジェクト。

setJccTimestampAtName

形式:

```
public void setJccTimestampAtName(String parameterMarkerName,
  java.sql.Timestamp x)
  throws java.sql.SQLException
public void setJccTimestampAtName(String parameterMarkerName,
  java.sql.Timestamp x,
  java.util.Calendar cal)
  throws java.sql.SQLException
```

名前付きパラメーターに java.sql.Timestamp 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる java.sql.Timestamp 値。

cal タイム・スタンプを構成するために IBM Data Server Driver for JDBC and SQLJ によって使用される java.util.Calendar オブジェクト。

setJccUnicodeStreamAtName
形式:

```
public void setJccUnicodeStreamAtName(String parameterMarkerName,
  java.io.InputStream x, int length)
  throws java.sql.SQLException
```

名前付きパラメーターに、java.io.InputStream 内の Unicode 値を割り当てます。

このメソッドは、 DB2 Database for Linux, UNIX, and Windows データ・ソースへの接続の場合のみサポートされます。

パラメーターは以下のとおりです。

parameterMarkerName
値が割り当てられるパラメーター・マーカーの名前。

x パラメーター・マーカーに割り当てられる Unicode java.io.InputStream 値。

length
パラメーター・マーカーに割り当てられる java.io.InputStream 値のバイト数。

DB2ResultSet インターフェース

com.ibm.db2.jcc.DB2ResultSet インターフェースは、 IBM Data Server Driver for JDBC and SQLJ 限定の照会情報が入手できるオブジェクトの作成に使用されます。

DB2ResultSet は、java.sql.Wrapper インターフェースをインプリメントします。

DB2ResultSet のメソッド:

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getDB2RowChangeToken
形式:

```
public long DB2ResultSet.getDB2RowChangeToken()
  throws java.sql.SQLException
```

現在行の行変更トークンを (使用可能であれば) 戻します。オプティミスティック・ロック列が要求されていないか、または使用不可である場合は、0 を戻します。

getDB2RID
形式:

```
public Object DB2ResultSet.getDB2RID()
  throws java.sql.SQLException
```

現在行の RID を (使用可能であれば) 戻します。オプティミスティック・ロック列が要求されており、使用可能であれば、RID は使用可能です。オプティミスティック・ロック列が要求されていないか、または使用不可である場合は、NULL を戻します。

getDB2RIDType

形式:

```
public int DB2ResultSet.getDB2RIDType()
  throws java.sql.SQLException
```

DB2ResultSet 内の RID 列のデータ・タイプを戻します。戻り値は java.sql.Types 定数にマップされます。 DB2ResultSet に RID 列が含まれていない場合、java.sql.Types.NULL が戻されます。

DB2ResultSetMetaData インターフェース

com.ibm.db2.jcc.DB2ResultSetMetaData インターフェースには、ResultSet オブジェクトに関する情報を提供するメソッドが用意されています。

com.ibm.db2.jcc.DB2ResultSetMetaData メソッドを使用する前に、java.sql.ResultSet.getMetaData 呼び出しから戻される java.sql.ResultSetMetaData オブジェクトを com.ibm.db2.jcc.DB2ResultSetMetaData にキャストする必要があります。

DB2ResultSetMetaData のメソッド:

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getDB2OptimisticLockingColumns

形式:

```
public int getDB2OptimisticLockingColumns()
  throws java.sql.SQLException
```

オプティミスティック・ロック列が使用可能であるかどうかを示す値を戻します。可能な値は以下のとおりです。

0 オプティミスティック・ロック列は使用不可です。

1 オプティミスティック・ロック列は使用可能ですが、変更トークンが false negative を避ける細分性を持っていない可能性があります。

2 オプティミスティック・ロック列は使用可能であり、変更トークンは false negative を避ける細分性を持っています。

isDB2ColumnNameDerived

形式:

```
public boolean isDB2ColumnNameDerived (int column)
  throws java.sql.SQLException
```

ResultSet 列の名前が、ResultSet を生成した SQL SELECT リストにある場合に、true を戻します。

例えば、ResultSet が SQL ステートメント SELECT EMPNAME, SUM(SALARY) FROM EMP から生成されるとします。列名 EMPNAME は

SQL SELECT リストから派生していますが、SUM(SALARY) に対応する ResultSet 内の列名は SELECT リストから派生しません。

パラメーターの説明:

column
ResultSet 内の列の名前。

DB2RowID インターフェース

com.ibm.db2.jcc.DB2RowID インターフェースは、SQL ROWID データ・タイプで使用するための Java オブジェクトを宣言するために使用されます。

com.ibm.db2.jcc.DB2RowID インターフェースは IBM Informix Dynamic Server との接続には適用されません。

DB2RowID のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getBytes
形式:

```
public byte[] getBytes()
```

com.ibm.jcc.DB2RowID オブジェクトをバイトに変換します。

DB2SimpleDataSource クラス

com.ibm.db2.jcc.DB2SimpleDataSource クラスは DB2BaseDataSource クラスを拡張するものです。

DB2BaseDataSource オブジェクトは、接続プールまたは分散トランザクションはサポートしていません。これには DB2BaseDataSource クラスに含まれるすべてのプロパティーとメソッドが含まれています。加えて、DB2SimpleDataSource には以下の IBM Data Server Driver for JDBC and SQLJ 専用プロパティーが含まれています。

DB2SimpleDataSource は java.sql.Wrapper インターフェースをインプリメントします。

DB2SimpleDataSource のプロパティー

以下のプロパティーは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。このプロパティーの説明については、『IBM Data Server Driver for JDBC and SQLJ のプロパティー』を参照してください。
String com.ibm.db2.jcc.DB2SimpleDataSource.password

DB2SimpleDataSource のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

setPassword
形式:

```
public void setPassword(String password)
```

DB2SimpleDataSource オブジェクトのパスワードを設定します。対応する getPassword メソッドはありません。したがって、パスワードを検索して暗号化解除する方法がないため、パスワードは暗号化できません。

DB2Sqlca クラス

com.ibm.db2.jcc.DB2Sqlca クラスは SQLCA をカプセル化したものです。

DB2Sqlca のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

getMessage

形式:

```
public abstract String getMessage()
```

エラー・メッセージ・テキストを戻します。

getSqlCode

形式:

```
public abstract int getSqlCode()
```

SQL エラー・コード値を戻します。

getSqlErrd

形式:

```
public abstract int[] getSqlErrd()
```

各エレメントに SQLCA SQLERRD が含まれている配列を戻します。

getSqlErrmc

形式:

```
public abstract String getSqlErrmc()
```

スペースで区切られた SQLCA SQLERRMC 値を含むストリングを戻します。

getSqlErrmcTokens

形式:

```
public abstract String[] getSqlErrmcTokens()
```

各エレメントに SQLCA SQLERRMC トークンが含まれている配列を戻します。

getSqlErrp

形式:

```
public abstract String getSqlErrp()
```

SQLCA SQLERRP 値を戻します。

getSqlState

形式:

```
public abstract String getSqlState()
```

SQLCA SQLSTATE 値を戻します。

getSqlWarn

形式:

```
public abstract char[] getSqlWarn()
```

各エレメントに SQLCA SQLWARN 値が含まれている配列を戻します。

DB2Statement インターフェース

com.ibm.db2.jcc.DB2Statement インターフェースは java.sql.Statement インターフェースを拡張するものです。

DB2Statement は java.sql.Wrapper インターフェースをインプリメントします。

DB2Statement のフィールド

以下のフィールドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

public static final int RETURN_OPTLOCK_COLUMN_NONE = 0
public static final int RETURN_OPTLOCK_COLUMN_ALWAYS = 1
public static final int RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES = 2

これらの値は DB2Statement.executeDB2OptimisticLockingQuery メソッドの引数です。

DB2Statement のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

executeDB2OptimisticLockingQuery

形式:

```
public java.sql.ResultSet DB2Statement.executeDB2OptimisticLockingQuery(
  String sql,
  int returnOptLockingColumn)
  throws java.sql.SQLException
```

SQL 照会ステートメントを実行し、要求された場合はオプティミスティック・ロック情報を含む ResultSet を戻します。

パラメーターの説明:

sql 単一の ResultSet を戻す SQL SELECT ステートメント。

returnOptimisticLockingColumns

オプティミスティック・ロック列を戻すかどうかを指定します。可能な値は以下のとおりです。

値	説明
DB2Statement.RETURN_OPTLOCK_COLUMN_NONE (0)	オプティミスティック・ロック列を戻しません。
DB2Statement.RETURN_OPTLOCK_COLUMN_ALWAYS (1)	行の変更列が単一の行を一意的に表さない場合でも、行の変更列を結果セットに追加します。この設定は、データベース準備属性 WITH ROW CHANGE COLUMNS POSSIBLY DISTINCT に相当します。

値	説明
DB2Statement.RETURN_OPTLOCK_COLUMN_NO_FALSE_NEGATIVES (2)	行の変更列が単一の行を一意的に表す場合にのみ、その行の変更列を結果セットに追加します。この設定は、データベース準備属性 WITH ROW CHANGE COLUMNS ALWAYS DISTINCT に相当します。

getDB2ClientProgramId

形式:

```
public String getDB2ClientProgramId()
  throws java.sql.SQLException
```

データ・ソース上に保管されている、接続のユーザー定義クライアント・プログラム ID を戻します。

setDB2ClientProgramId

形式:

```
public abstract void setDB2ClientProgramId(String program-ID)
  throws java.sql.SQLException
```

DB2 for z/OS サーバー上で、接続用のユーザー定義プログラム ID を設定します。このプログラム ID は、呼び出し元を識別するために使用される 80 バイトのストリングです。 DB2 for z/OS サーバーは、ストリングを IFCID 316 トレース・レコード内に他の統計と共に配置するので、どのプログラムが特定の SQL ステートメントと関連しているかを識別できます。

getIDSBigSerial

形式:

```
public int getIDSBigSerial()
  throws java.sql.SQLException
```

前に実行された INSERT ステートメントによって自動生成キーが挿入された後、BIGSERIAL 列から自動生成キーを検索します。

getIDSBigSerial を正常に実行するには、以下の条件が当てはまらなければなりません。

- INSERT ステートメントが、このメソッドが呼び出される前に実行された最後の SQL ステートメントである。
- この行が挿入される表に BIGSERIAL 列が含まれている。
- INSERT ステートメントを準備または実行する JDBC Connection.prepareStatement メソッドまたは Statement.executeUpdate メソッドのフォームには、自動生成キーを要求するパラメーターがない。

このメソッドは、 IBM Informix Dynamic Server (IDS) データベースへの接続にのみ適用されます。

getIDSSerial

形式:

```
public int getIDSSerial()
  throws java.sql.SQLException
```

前に実行された INSERT ステートメントによって自動生成キーが挿入された後、SERIAL 列から自動生成キーを検索します。

getIDSSerial を正常に実行するには、以下の条件が当てはまらなければなりません。

- INSERT ステートメントが、このメソッドが呼び出される前に実行された最後の SQL ステートメントである。
- この行が挿入される表に SERIAL 列が含まれている。
- INSERT ステートメントを準備または実行する JDBC Connection.prepareStatement メソッドまたは Statement.executeUpdate メソッドのフォームには、自動生成キーを要求するパラメーターがない。

このメソッドは、 IBM Informix Dynamic Server (IDS) データベースへの接続にのみ適用されます。

getIDSSerial8

形式:

```
public long getIDSSerial8()
  throws java.sql.SQLException
```

前に実行された INSERT ステートメントによって自動生成キーが挿入された後、SERIAL8 列から自動生成キーを検索します。

getIDSSerial8 を正常に実行するには、以下の条件が当てはまらなければなりません。

- INSERT ステートメントが、このメソッドが呼び出される前に実行された最後の SQL ステートメントである。
- この行が挿入される表に SERIAL8 列が含まれている。
- INSERT ステートメントを準備または実行する JDBC Connection.prepareStatement メソッドまたは Statement.executeUpdate メソッドのフォームには、自動生成キーを要求するパラメーターがない。

このメソッドは、IDS データ・ソースへの接続にのみ適用されます。

getIDSSQLStatementOffSet

形式:

```
public int getIDSSQLStatementOffSet()
  throws java.sql.SQLException
```

IDS データ・ソース上で SQL ステートメントを実行した後に、ステートメントに構文エラーがあると、getIDSSQLStatementOffSet は構文エラーのステートメント・テキストにオフセットを戻します。

getIDSSQLStatementOffSet は以下を戻します。

- 0、ステートメントに構文エラーがない場合。
- -1、データ・ソースが IDS ではない場合。

このメソッドは、IDS データ・ソースへの接続にのみ適用されます。

DB2SystemMonitor インターフェース

com.ibm.db2.jcc.DB2SystemMonitor インターフェースは、接続用のシステム・モニター・データを収集するために使用されます。接続ごとに 1 つの DB2SystemMonitor インスタンスを持つことができます。

DB2SystemMonitor のフィールド

以下のフィールドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

public final static int RESET_TIMES
public final static int ACCUMULATE_TIMES
これらの値は DB2SystemMonitor.start メソッドの引数です。 RESET_TIMES は、モニターの開始前に時間カウンターをゼロに設定します。 ACCUMULATE_TIMES は時間カウンターをゼロに設定しません。

DB2SystemMonitor のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

enable
形式:

```
public void enable(boolean on)
  throws java.sql.SQLException
```

接続と関連付けられたシステム・モニターを使用可能にします。このメソッドをモニター中に呼び出すことはできません。 enable を呼び出すとすべての時間がリセットされます。

getApplicationTimeMillis
形式:

```
public long getApplicationTimeMillis()
  throws java.sql.SQLException
```

アプリケーション、JDBC ドライバー、ネットワーク入出力、およびデータベース・サーバーの経過時間の合計を戻します。ミリ秒単位の時間。

モニターされる経過時間インターバルは、JDBC ドライバーの処理における以下のポイント間の、ミリ秒単位の差異です。

インターバルの開始
start の呼び出し時。

インターバルの終了
stop の呼び出し時。

システム・モニターが無効の場合、getApplicationTimeMillis は 0 を戻します。最初に stop メソッドを呼び出さずにこのメソッドを呼び出すと、結果は SQLException になります。

getCoreDriverTimeMicros
形式:

```
public long getCoreDriverTimeMicros()
  throws java.sql.SQLException
```

システム・モニターが有効であったときに収集された、経過モニター API 時間の合計を戻します。この時間はマイクロ秒単位です。

モニターされる API は、処理時間が収集される JDBC ドライバー・メソッドです。一般に、経過時間はネットワーク入出力またはデータベース・サーバー対

話の結果となる API についてのみモニターされます。例えば、PreparedStatement.setXXX メソッドと ResultSet.getXXX メソッドはモニターされません。

モニターされる API 経過時間には、メソッド呼び出しのドライバーに費やされる合計時間が含まれます。この時間には、ネットワーク入出力時間およびデータベース・サーバー経過時間が含まれます。

モニターされる API 経過時間インターバルは、JDBC ドライバーの処理における以下のポイント間の、マイクロ秒単位の差異です。

インターバルの開始
モニターされる API がアプリケーションにより呼び出される時。

インターバルの終了
モニターされる API が制御をアプリケーションに戻す直前。

システム・モニターが無効の場合、getCoreDriverTimeMicros は 0 を戻します。最初に stop メソッドを呼び出さずにこのメソッドを呼び出すか、または基礎となる JVM がマイクロ秒単位のレポート時間をサポートしないと、結果は SQLException になります。

getNetworkIOTimeMicros

形式:

```
public long getNetworkIOTimeMicros()
  throws java.sql.SQLException
```

システム・モニターが有効であったときに収集された、経過ネットワーク入出力時間の合計を戻します。この時間はマイクロ秒単位です。

経過ネットワーク入出力時間には、ネットワーク入出力ストリームとの間での DRDA データの読み取り/書き込み時間が含まれます。ネットワーク入出力経過時間インターバルは、JDBC ドライバーで以下の操作を実行するための時間インターバルです。

- DRDA メッセージをデータベース・サーバーに送信するために TCP/IP コマンドを発行する。この時間インターバルは、ネットワーク入出力ストリームとの間での書き込み/フラッシュが実行される直前と直後のポイント間の、マイクロ秒単位の差異です。
- DRDA 応答メッセージをデータベース・サーバーから受け取るために TCP/IP コマンドを発行する。この時間インターバルは、ネットワーク入出力ストリーム上で読み取りが実行される直前と直後のポイント間の、マイクロ秒単位の差異です。

ネットワーク入出力時間インターバルは、すべての送受信操作についてキャプチャーされます。これにはコミットとロールバックのためのメッセージの送信も含まれます。

ネットワーク入出力を待機するために費やす時間は、優先順位が低い SQL 要求のためのデータベース・サーバーでの CPU ディスパッチングの遅延により影響を受ける場合があります。

システム・モニターが無効の場合、getNetworkIOTimeMicros は 0 を戻します。最初に stop メソッドを呼び出さずにこのメソッドを呼び出すか、または基礎となる JVM がマイクロ秒単位のレポート時間をサポートしないと、結果は SQLException になります。

getServerTimeMicros

形式:

```
public long getServerTimeMicros()
  throws java.sql.SQLException
```

システム・モニターが有効であったときに収集された、レポートされたすべてのデータベース・サーバー経過時間の合計を戻します。この時間はマイクロ秒単位です。

データベース・サーバーは、経過時間を以下の条件下でレポートします。

- データベース・サーバーが経過時間データをクライアントに戻すことをサポートしている。

  DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS はこの機能をサポートします。
- データベース・サーバーがモニター可能な操作を実行している。例えば、データベース・サーバー経過時間はコミットまたはロールバックのためには戻されません。

*DB2 Database for Linux, UNIX, and Windows への IBM Data Server Driver for JDBC and SQLJ Type 2 接続、および IBM Data Server Driver for JDBC and SQLJ Type 4 接続の場合:*データベース・サーバー経過時間は、要求データ・ストリームの解析、コマンドの処理、データベース・サーバーでの応答データ・ストリームの生成のための経過時間と定義されます。データ・ストリームの送受信のためのネットワーク時間は含まれません。データベース・サーバー経過時間インターバルは、データベース・サーバー処理における以下のポイント間の、マイクロ秒単位の差異です。

インターバルの開始
オペレーティング・システムが、JDBC ドライバーから受け取った TCP/IP メッセージの処理をデータベース・サーバーにディスパッチした時。

インターバルの終了
データベース・サーバーが応答メッセージをクライアントに戻すために TCP/IP コマンドを発行する準備ができている時。

DB2 for z/OS への IBM Data Server Driver for JDBC and SQLJ Type 2 接続の場合: データベース・サーバー経過時間インターバルは、JDBC ドライバー・ネイティブ処理における以下のポイント間の、マイクロ秒単位の差異です。

インターバルの開始
JDBC ドライバーのネイティブ・メソッドが SQL 要求を処理するために RRS 接続機能を呼び出すときの z/OS Store Clock (STCK) 値。

インターバルの終了
SQL 要求を処理するために、RRS 接続機能の呼び出しに続いて JDBC ドライバーのネイティブ・メソッドに制御が戻るときの z/OS Store Clock (STCK) 値。

システム・モニターが無効の場合、getServerTimeMicros は 0 を戻します。最初に stop メソッドを呼び出さずにこのメソッドを呼び出すと、結果は SQLException になります。

start

形式:

```
public void start (int lapMode)
  throws java.sql.SQLException
```

システム・モニターが有効な場合、start は接続のシステム・モニター・データの収集を開始します。 *lapMode* の有効値は、RESET_TIMES または ACCUMULATE_TIMES です。

このメソッドを無効なシステム・モニターを使用して呼び出しても、何も実行されません。このメソッドを複数回、stop 呼び出しを間に挟まずに呼び出すと、結果は SQLException になります。

stop

形式:

```
public void stop()
  throws java.sql.SQLException
```

システム・モニターが有効な場合、stop は接続のシステム・モニター・データの収集を停止します。モニターが停止した後に、モニター時間は DB2SystemMonitor の getXXX メソッドで入手できます。

このメソッドを無効なシステム・モニターを使用して呼び出しても、何も実行されません。このメソッドを最初に start を呼び出さずに呼び出すか、または start 呼び出しを間に挟まずに複数回呼び出すと、結果は SQLException になります。

DB2TraceManager クラス

com.ibm.db2.jcc.DB2TraceManager クラスはグローバル・ログ書き込みプログラムを制御します。

グローバル・ログ書き込みプログラムはドライバー全体にわたるものであり、すべての接続に適用されます。グローバル・ログ書き込みプログラムは、その他の JDBC ログ書き込みプログラムをオーバーライドします。DB2TraceManager クラスは、グローバル・ログ書き込みプログラムを始動するだけでなく、すべてのタイプのログ書き込みプログラムによるトレースを中断および再開する機能を提供します。つまり、DB2TraceManager クラスの中断および再開メソッドは、現在および将来の DriverManager ログ書き込みプログラム、DataSource ログ書き込みプログラム、または IBM Data Server Driver for JDBC and SQLJ 限定の接続レベルのログ書き込みプログラムすべてに適用されることになります。

DB2TraceManager のメソッド

getTraceManager

形式:

```
static public DB2TraceManager getTraceManager()
  throws java.sql.SQLException
```

グローバル・ログ書き込みプログラムのインスタンスを取得します。

setLogWriter

形式:

```
public abstract void setLogWriter(String traceDirectory,
  String baseTraceFileName, int traceLevel)
  throws java.sql.SQLException
public abstract void setLogWriter(String traceFile,
  boolean fileAppend, int traceLevel)
  throws java.sql.SQLException
public abstract void setLogWriter(java.io.PrintWriter logWriter,
  int traceLevel)
  throws java.sql.SQLException
```

グローバル・トレースを有効にします。 setLogWriter が呼び出されると、DB2TraceManager.unsetLogWriter が呼び出されるまで、DataSource または Connection トレースに対するすべての呼び出しは廃棄されます。

setLogWriter が呼び出されると、使用する setLogWriter の形式に応じて、将来の Connection または DataSource トレースはすべてトレース・ファイルまたは PrintWriter にリダイレクトされます。 setLogWriter が呼び出されるとグローバル・トレースは中断され、トレースが再開されたときに指定した設定が有効になります。

パラメーターの説明:

traceDirectory
グローバル・トレース情報の書き込み先のディレクトリーを指定します。この設定は、DataSource または DriverManager 接続の traceDirectory および logWriter プロパティーの設定をオーバーライドします。

traceDirectory パラメーターを指定して setLogWriter の形式を使用すると、JDBC ドライバーは、setLogWriter が呼び出されたときに traceFileAppend プロパティーを false に設定します。つまり、既存のログ・ファイルは上書きされます。 JDBC ドライバーの各接続は、指定したディレクトリーの異なるファイルにトレースされます。そのディレクトリー内のファイルの命名規則は、baseTraceFileName に NULL 以外の値を指定するかどうかに応じて以下のように異なります。

- baseTraceFileName に NULL 値を指定すると、接続のトレースは traceFile_global_*n* という名前のファイルへ書き込まれます。

  n は *n* 番目の JDBC ドライバー接続です。
- baseTraceFileName に NULL 以外の値を指定すると、接続のトレースは *baseTraceFileName*_global_*n* という名前のファイルへ書き込まれます。

  baseTraceFileName は baseTraceFileName パラメーターの値です。

  n は *n* 番目の JDBC ドライバー接続です。

baseTraceFileName
グローバル・トレース情報の書き込み先のファイル名の語幹を指定します。baseTraceFileName と traceDirectory の組み合わせによって、グローバル・トレース・ログ・ファイルの絶対パス名が決定されます。

traceFileName
グローバル・トレース情報の書き込み先のファイルを指定します。この設定

は、DataSource または DriverManager 接続の traceFile および logWriter プロパティーの設定をオーバーライドします。

traceFileName パラメーターを指定して setLogWriter の形式を使用すると、1 つのログ・ファイルだけが書き込まれます。

traceFileName にはディレクトリー・パスを含めることができます。

logWriter
すべてのグローバル・ログ・レコードの書き込み先の文字出力ストリームを指定します。

この値は、DataSource または DriverManager 接続で logWriter プロパティーをオーバーライドします。

traceLevel
何をトレースするかを指定します。

traceLevel パラメーターで、以下のトレースを 1 つ以上指定することができます。

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE (X'00')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS (X'01')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS (X'02')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS (X'04')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION (X'10')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS (X'20')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS (X'40')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA (X'80')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA (X'100')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS (X'200')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ (X'400')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (DB2 Database for Linux, UNIX, and Windows 用の IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ) (X'800')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS (X'2000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS (X'4000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS (X'8000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR (X'20000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS () (X'40000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL (X'FFFFFFFF')

複数のトレースを指定するには、以下の技法のいずれかを使用します。

- 2 つ以上のトレース値にビット単位 OR (|) 演算子を使用する。例えば、DRDA フローおよび接続呼び出しをトレースするには、traceLevel に以下の値を指定します。

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- ビット単位の補数 (波形記号 (~)) 演算子とトレース値を使用して、特定のトレース以外のすべてを指定する。例えば、DRDA フロー以外のすべてをトレースする場合、traceLevel に以下の値を指定します。

  ```
  ~TRACE_DRDA_FLOWS
  ```

fileAppend
traceFile パラメーターによって指定されるファイルへ付加するか、あるいは上書きするかを指定します。 true は既存のファイルを上書きしないことを意味します。

unsetLogWriter
形式:

```
public abstract void unsetLogWriter()
  throws java.sql.SQLException
```

将来の接続に対してグローバル・ログ書き込みプログラムを無効にします。

suspendTrace
形式:

```
public void suspendTrace()
  throws java.sql.SQLException
```

現在および将来の接続に対して、Connection レベルまたは DataSource レベルのグローバル・トレースをすべて中断します。グローバル・ログ書き込みプログラムを有効または無効にするときに、suspendTrace を呼び出すことができます。

resumeTrace
形式:

```
public void resumeTrace()
  throws java.sql.SQLException
```

現在および将来の接続に対して、Connection レベルまたは DataSource レベルのグローバル・トレースをすべて再開します。グローバル・ログ書き込みプログラムを有効または無効にするときに、resumeTrace を呼び出すことができます。グローバル・ログ書き込みプログラムを無効にすると、resumeTrace は Connection レベルまたは DataSource レベルのトレースを再開します。グローバル・ログ書き込みプログラムを有効にすると、resumeTrace はグローバル・トレースを再開します。

getLogWriter
形式:

```
public abstract java.io.PrintWriter getLogWriter()
  throws java.sql.SQLException
```

グローバル・ログ書き込みプログラムが設定されている場合、その PrintWriter を戻します。それ以外の場合、getLogWriter は NULL を戻します。

getTraceFile
形式:

```
public abstract String getTraceFile()
  throws java.sql.SQLException
```

グローバル・ログ書き込みプログラムが設定されている場合、その宛先ファイルの名前を戻します。それ以外の場合、getTraceFile は NULL を戻します。

getTraceDirectory

形式:

```
public abstract String getTraceDirectory()
  throws java.sql.SQLException
```

グローバル・ログ書き込みプログラム・ファイルが設定されている場合、その宛先ディレクトリーの名前を戻します。それ以外の場合、getTraceDirectory は NULL を戻します。

getTraceLevel

形式:

```
public abstract int getTraceLevel()
  throws java.sql.SQLException
```

グローバル・トレースが設定されている場合、そのトレース・レベルを戻します。それ以外の場合、getTraceLevel は -1 (TRACE_ALL) を戻します。

getTraceFileAppend

形式:

```
public abstract boolean getTraceFileAppend()
  throws java.sql.SQLException
```

グローバル・トレース・レコードがトレース・ファイルに追加されている場合、true を戻します。それ以外の場合、getTraceFileAppend は false を戻します。

DB2TraceManagerMXBean インターフェース

com.ibm.db2.jcc.mx.DB2TraceManagerMXBean インターフェースは、アプリケーションがリモート・トレース・コントローラー用の MXBean として DB2TraceManager を使用可能にするための手段です。

DB2TraceManagerMXBean のメソッド

setTraceFile

形式:

```
public void setTraceFile(String traceFile,
  boolean fileAppend, int traceLevel)
  throws java.sql.SQLException
```

リモート・トレース・マネージャーがトレース情報を書き込むファイルの名前、およびトレースされる情報のタイプを指定します。

パラメーターの説明:

traceFileName

グローバル・トレース情報の書き込み先のファイルを指定します。この設定は、DataSource または DriverManager 接続の traceFile および logWriter プロパティーの設定をオーバーライドします。

traceFileName パラメーターを指定して setLogWriter の形式を使用すると、1 つのログ・ファイルだけが書き込まれます。

traceFileName にはディレクトリー・パスを含めることができます。

fileAppend

traceFile パラメーターによって指定されるファイルへ付加するか、あるいは上書きするかを指定します。 true は既存のファイルを上書きしないことを意味します。

traceLevel

何をトレースするかを指定します。

traceLevel パラメーターで、以下のトレースを 1 つ以上指定することができます。

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE (X'00')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS (X'01')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS (X'02')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS (X'04')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION (X'10')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS (X'20')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS (X'40')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA (X'80')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA (X'100')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS (X'200')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ (X'400')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (DB2 Database for Linux, UNIX, and Windows 用の IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ) (X'800')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS (X'2000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS (X'4000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS (X'8000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR (X'20000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS () (X'40000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL (X'FFFFFFFF')

複数のトレースを指定するには、以下の技法のいずれかを使用します。

- 2 つ以上のトレース値にビット単位 OR (|) 演算子を使用する。例えば、DRDA フローおよび接続呼び出しをトレースするには、traceLevel に以下の値を指定します。

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- ビット単位の補数 (波形記号 (~)) 演算子とトレース値を使用して、特定のトレース以外のすべてを指定する。例えば、DRDA フロー以外のすべてをトレースする場合、traceLevel に以下の値を指定します。

  ```
  ~TRACE_DRDA_FLOWS
  ```

getTraceFile

形式:

```
public void getTraceFile()
  throws java.sql.SQLException
```

リモート・トレース・コントローラーの宛先ファイルの名前が設定されている場合、その宛先ファイルの名前を戻します。それ以外の場合、getTraceFile は NULL を戻します。

setTraceDirectory

形式:

```
public void setTraceDirectory(String traceDirectory,
  String baseTraceFileName,
  int traceLevel) throws java.sql.SQLException
```

リモート・トレース・コントローラーがトレース情報を書き込むディレクトリーの名前、およびトレースされる情報のタイプを指定します。

パラメーターの説明:

traceDirectory

トレース情報の書き込み先のディレクトリーを指定します。この設定は、DataSource または DriverManager 接続の traceDirectory および logWriter プロパティーの設定をオーバーライドします。

JDBC ドライバーの各接続は、指定したディレクトリーの異なるファイルにトレースされます。そのディレクトリー内のファイルの命名規則は、baseTraceFileName に NULL 以外の値を指定するかどうかに応じて以下のように異なります。

- baseTraceFileName に NULL 値を指定すると、接続のトレースは traceFile_global_*n* という名前のファイルへ書き込まれます。

  n は *n* 番目の JDBC ドライバー接続です。
- baseTraceFileName に NULL 以外の値を指定すると、接続のトレースは *baseTraceFileName*_global_*n* という名前のファイルへ書き込まれます。

  baseTraceFileName は baseTraceFileName パラメーターの値です。

  n は *n* 番目の JDBC ドライバー接続です。

baseTraceFileName

グローバル・トレース情報の書き込み先のファイル名の語幹を指定します。baseTraceFileName と traceDirectory の組み合わせによって、グローバル・トレース・ログ・ファイルの絶対パス名が決定されます。

traceLevel

何をトレースするかを指定します。

traceLevel パラメーターで、以下のトレースを 1 つ以上指定することができます。

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_NONE (X'00')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTION_CALLS (X'01')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_STATEMENT_CALLS (X'02')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_CALLS (X'04')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRIVER_CONFIGURATION (X'10')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_CONNECTS (X'20')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DRDA_FLOWS (X'40')

- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_RESULT_SET_META_DATA (X'80')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_PARAMETER_META_DATA (X'100')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DIAGNOSTICS (X'200')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SQLJ (X'400')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_XA_CALLS (DB2 Database for Linux, UNIX, and Windows 用の IBM Data Server Driver for JDBC and SQLJ Type 2 接続のみ) (X'800')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_META_CALLS (X'2000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_DATASOURCE_CALLS (X'4000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_LARGE_OBJECT_CALLS (X'8000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_SYSTEM_MONITOR (X'20000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_TRACEPOINTS () (X'40000')
- com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL (X'FFFFFFFF')

複数のトレースを指定するには、以下の技法のいずれかを使用します。

- 2 つ以上のトレース値にビット単位 OR (|) 演算子を使用する。例えば、DRDA フローおよび接続呼び出しをトレースするには、traceLevel に以下の値を指定します。

  ```
  TRACE_DRDA_FLOWS|TRACE_CONNECTION_CALLS
  ```

- ビット単位の補数 (波形記号 (~)) 演算子とトレース値を使用して、特定のトレース以外のすべてを指定する。例えば、DRDA フロー以外のすべてをトレースする場合、traceLevel に以下の値を指定します。

  ```
  ~TRACE_DRDA_FLOWS
  ```

getTraceFileAppend

形式:

```
public abstract boolean getTraceFileAppend()
  throws java.sql.SQLException
```

トレース・コントローラーによって生成されるトレース・レコードがトレース・ファイルに追加される場合、true を戻します。それ以外の場合、getTraceFileAppend は false を戻します。

getTraceDirectory

形式:

```
public void getTraceDirectory()
  throws java.sql.SQLException
```

トレース・コントローラーによって生成されるトレース・レコードの宛先ディレクトリーの名前が設定されている場合、その宛先ディレクトリーの名前を戻します。それ以外の場合、getTraceDirectory は NULL を戻します。

getTraceLevel

形式:

```
public void getTraceLevel()
  throws java.sql.SQLException
```

トレース・コントローラーによって生成されるトレース・レコードのトレース・レベルが設定されている場合、そのトレース・レベルを戻します。それ以外の場合、getTraceLevel は -1 (TRACE_ALL) を戻します。

unsetLogWriter

形式:

```
public abstract void unsetLogWriter()
  throws java.sql.SQLException
```

将来の接続に対してグローバル・ログ書き込みプログラムを無効にします。

suspendTrace

形式:

```
public void suspendTrace()
  throws java.sql.SQLException
```

現在および将来の接続に対して、Connection レベルまたは DataSource レベルのグローバル・トレースをすべて中断します。グローバル・ログ書き込みプログラムを有効または無効にするときに、suspendTrace を呼び出すことができます。

resumeTrace

形式:

```
public void resumeTrace()
  throws java.sql.SQLException
```

現在および将来の接続に対して、Connection レベルまたは DataSource レベルのグローバル・トレースをすべて再開します。グローバル・ログ書き込みプログラムを有効または無効にするときに、resumeTrace を呼び出すことができます。グローバル・ログ書き込みプログラムを無効にすると、resumeTrace は Connection レベルまたは DataSource レベルのトレースを再開します。グローバル・ログ書き込みプログラムを有効にすると、resumeTrace はグローバル・トレースを再開します。

DB2Types クラス

com.ibm.db2.jcc.DB2Types クラスは、 IBM Data Server Driver for JDBC and SQLJ 専用のデータ・タイプを定義するフィールドを提供します。

DB2Types フィールド

次の定数は、 IBM Data Server Driver for JDBC and SQLJ 専用のタイプ・コードを定義します。

- public final static int BLOB_FILE = -100002
- public final static int CLOB_FILE = -100003
- public final static int CURSOR = -100008
- public final static int DECFLOAT = -100001
- public final static int XML_AS_BLOB_FILE = -100004
- public final static int XML_AS_CLOB_FILE = -100005

DB2XADataSource クラス

DB2XADataSource は XADataSource オブジェクトのファクトリーです。このインターフェースをインプリメントするオブジェクトは、Java Naming and Directory Interface (JNDI) に基づくネーム・サービスで登録されます。

com.ibm.db2.jcc.DB2XADataSource クラスは com.ibm.db2.jcc.DB2BaseDataSource クラスを拡張し、javax.sql.XADataSource、java.io.Serializable、および javax.naming.Referenceable インターフェースをインプリメントします。

DB2XADataSource のメソッド

getDB2TrustedXAConnection

形式:

```
public Object[] getDB2TrustedXAConnection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedXAConnection(
  java.util.Properties properties)
  throws java.sql.SQLException
public Object[] getDB2TrustedXAConnection(
  org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

システム許可 ID を使用するアプリケーション・サーバーは、トラステッド接続を確立するためにこのメソッドを使用します。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

以下のエレメントは Object[] で戻されます。

- 最初のエレメントは、DB2TrustedXAConnection インスタンスです。
- 2 番目のエレメントは、生成された XA 接続インスタンスの固有の cookie です。

最初のフォーム getDB2TrustedXAConnection はユーザー ID とパスワードを提供します。 2 番目のフォーム getDB2TrustedXAConnection は、DB2XADataSource オブジェクトのユーザー ID とパスワードを使用します。 3 番目のフォーム getDB2TrustedXAConnection は、Kerberos セキュリティーを使用する接続のためのものです。

パラメーターの説明:

user
トラステッド接続の確立に使用される許可 ID。

password
トラステッド接続の確立に使用される許可 ID のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

properties
接続のプロパティー。

getDB2TrustedPooledConnection

形式:

```
public Object[] getDB2TrustedPooledConnection(java.util.Properties properties)
  throws java.sql.SQLException
```

システム許可 ID を使用するアプリケーション・サーバーは、DB2XADataSource オブジェクトのユーザー ID とパスワードを使用して、トラステッド接続を確立するためにこのメソッドを使用します。

トラステッド接続は、以下の場合にサポートされます。

- DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降、および DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 4 接続
- DB2 for z/OS バージョン 9.1 以降への IBM Data Server Driver for JDBC and SQLJ Type 2 接続

以下のエレメントは Object[] で戻されます。

- 最初のエレメントは、トラステッド DB2TrustedPooledConnection インスタンスです。
- 2 番目のエレメントは、生成されたプール接続インスタンスの固有の cookie です。

パラメーターの説明:

properties
接続のプロパティー。

getDB2XAConnection

形式:

```
public DB2XAConnection getDB2XAConnection(String user,
  String password,
  java.util.Properties properties)
  throws java.sql.SQLException
public DB2XAConnection getDB2XAConnection(
   org.ietf.jgss.GSSCredential gssCredential,
  java.util.Properties properties)
  throws java.sql.SQLException
```

異種プール環境での初期非トラステッド接続を確立します。

最初のフォーム getDB2PooledConnection はユーザー ID とパスワードを提供します。 2 番目のフォーム getDB2XAConnection は、Kerberos セキュリティーを使用する接続のためのものです。

パラメーターの説明:

user
接続の確立に使用される許可 ID。

password
接続の確立に使用される許可 ID のパスワード。

gssCredential
データ・ソースが Kerberos セキュリティーを使用する場合に、別のプリンシパルから渡される委任証明書を指定します。

properties
接続のプロパティー。

DB2Xml インターフェース

com.ibm.db2.jcc.DB2Xml インターフェースは、 DB2 XML データ・タイプで使用するための Java オブジェクトを宣言するために使用されます。

DB2Xml のメソッド

以下のメソッドは、 IBM Data Server Driver for JDBC and SQLJ に対してのみ定義されます。

closeDB2Xml
形式:

```
public void closeDB2Xml()
  throws SQLException
```

com.ibm.jcc.DB2Xml オブジェクトと関連付けられているリソースを解放します。

getDB2AsciiStream
形式:

```
public java.io.InputStream getDB2AsciiStream()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトから検索し、そのデータを US-ASCII エンコードに変換します。

getDB2BinaryStream
形式:

```
public java.io.InputStream getDB2BinaryStream()
  throws SQLException
```

データを DB2Xml オブジェクトからバイナリー・ストリームとして検索します。バイナリー・ストリーム内のバイトの文字エンコード方式は、XML 1.0 仕様で定義されています。

getDB2Bytes
形式:

```
public byte[] getDB2Bytes()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトからバイト配列として検索します。バイトの文字エンコード方式は、XML 1.0 仕様で定義されています。

getDB2CharacterStream
形式:

```
public java.io.Reader getDB2CharacterStream()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトから java.io.Reader オブジェクトとして検索します。

getDB2String

形式:

```
public String getDB2String()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトから String 値として検索します。

getDB2XmlAsciiStream

形式:

```
public InputStream getDB2XmlAsciiStream()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトから検索し、そのデータを US-ASCII エンコードに変換し、戻されるデータに US-ASCII のエンコード仕様で XML 宣言を組み込みます。

getDB2XmlBinaryStream

形式:

```
public java.io.InputStream getDB2XmlBinaryStream(String targetEncoding)
  throws SQLExceptionn
```

データを DB2Xml オブジェクトからバイナリー・ストリームとして検索し、そのデータを *targetEncoding* に変換し、戻されるデータに *targetEncoding* のエンコード仕様で XML 宣言を組み込みます。

パラメーター:

targetEncoding
IANA 文字セット・レジストリーにリストされる有効なエンコード名。DB2 サーバーでサポートされているエンコード名は、『CCSID とシリアライズされた XML 出力データのエンコーディング名とのマップ』でリストされています。

getDB2XmlBytes

形式:

```
public byte[] getDB2XmlBytes(String targetEncoding)
  throws SQLExceptionn
```

データを DB2Xml オブジェクトからバイト配列として検索し、そのデータを *targetEncoding* に変換し、戻されるデータに *targetEncoding* のエンコード仕様で XML 宣言を組み込みます。

パラメーター:

targetEncoding
IANA 文字セット・レジストリーにリストされる有効なエンコード名。DB2 サーバーでサポートされているエンコード名は、『CCSID とシリアライズされた XML 出力データのエンコーディング名とのマップ』でリストされています。

getDB2XmlCharacterStream

形式:

```
public java.io.Reader getDB2XmlCharacterStream()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトから java.io.Reader オブジェクトとして検索し、そのデータを ISO-10646-UCS-2 エンコードに変換し、戻されるデータに ISO-10646-UCS-2 のエンコード仕様で XML 宣言を組み込みます。

getDB2XmlString

形式:

```
public String getDB2XmlString()
  throws SQLExceptionn
```

データを DB2Xml オブジェクトから String オブジェクトとして検索し、そのデータを ISO-10646-UCS-2 エンコードに変換し、戻されるデータに ISO-10646-UCS-2 のエンコード仕様で XML 宣言を組み込みます。

isDB2XmlClosed

形式:

```
public boolean isDB2XmlClosed()
  throws SQLException
```

com.ibm.jcc.DB2Xml オブジェクトがクローズされているかどうかを示します。

現行の IBM Data Server Driver for JDBC and SQLJ と以前の DB2 JDBC ドライバーとの JDBC の相違点

JDBC アプリケーションを古いドライバーから IBM Data Server Driver for JDBC and SQLJ にアップグレードする前に、それらのドライバー間の相違点を理解する必要があります。

重要: Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) は、推奨されていません。この情報は、アプリケーションの IBM Data Server Driver for JDBC and SQLJ への移行を支援するために提供されています。

サポートされるメソッド

JDBC ドライバーによりサポートされるメソッドの比較については、『JDBC API でのドライバーのサポート』を参照してください。

JDBC ドライバーによる連続ストリーミングの使用

IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 以降の場合、連続ストリーミングの使用は、 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降への接続の場合の LOB 検索のデフォルトです。

連続ストリーミングは IBM Data Server Driver for JDBC and SQLJ バージョン 3.1 以降でサポートされています。ただし、 IBM Data Server Driver for JDBC and SQLJ バージョン 3.2 以降では、連続ストリーミングの使用は、 DB2 for z/OS バージョン 9.1 以降への接続の場合の LOB および XML 検索のデフォルトです。

IBM Data Server Driver for JDBC and SQLJ の以前のバージョンおよび DB2 JDBC Type 2 ドライバーは、連続ストリーミングをサポートしませんでした。

重要: 連続ストリーミングでは、LOB または XML の値を ResultSet からアプリケーション変数に取り出す場合、カーソルを移動するまで、または ResultSet 上のカーソルをクローズするまでは、そのアプリケーション変数の内容を操作できます。その後は、アプリケーション変数の内容は操作できなくなります。アプリケーション変数内の LOB に対して何らかのアクションを実行すると、SQLException を受け取ります。例えば、連続ストリーミングが有効で、以下のようなステートメントを実行するとします。

```
...
ResultSet rs = stmt.executeQuery("SELECT CLOBCOL FROM MY_TABLE");
rs.next();                // Retrieve the first row of the ResultSet
Clob clobFromRow1  = rs.getClob(1);
                          // Put the CLOB from the first column of
                          // the first row in an application variable
String substr1Clob = clobFromRow1.getSubString(1,50);
                          // Retrieve the first 50 bytes of the CLOB
rs.next();                // Move the cursor to the next row.
                          // clobFromRow1 is no longer available.
// String substr2Clob = clobFromRow1.getSubString(51,100);
                          // This statement would yield an SQLException
Clob clobFromRow2  = rs.getClob(1);
                          // Put the CLOB from the first column of
                          // the second row in an application variable
rs.close();               // Close the ResultSet.
                          // clobFromRow2 is also no longer available.
```

rs.next() を実行してカーソルを ResultSet の 2 番目の行に置いた後は、clobFromRow1 内の CLOB 値は使用できなくなります。同様に、rs.close() を実行して ResultSet をクローズした後、clobFromRow1 および clobFromRow2 内の値は使用できなくなります。

この動作の変更によるエラーを避けるには、以下のアクションのいずれかをとる必要があります。

- アプリケーションを変更します。

  LOB データをアプリケーション変数に取り出すアプリケーションは、データの取り出しに使用されたカーソルが移動されるか閉じられるまでの間に限り、それらのアプリケーション変数のデータを操作できます。
- 連続ストリーミングを使用不可にします。それには、progressiveStreaming プロパティーを DB2BaseDataSource.NO (2) に設定します。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の ResultSetMetaData 値

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の場合、ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel のデフォルトの動作は、それ以前の JDBC ドライバーのデフォルトの動作とは異なります。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降を使用する必要があるが、アプリケーションが旧バージョンの JDBC ドライバーで戻された ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel の値

を戻す必要がある場合、useJDBC4ColumnNameAndLabelSemantics Connection および DataSource プロパティーを DB2BaseDataSource.NO (2) に設定できます。

自動生成キーを使用したバッチ更新の結果はドライバーのバージョンによって異なる

IBM Data Server Driver for JDBC and SQLJ バージョン 3.52 以降では、自動生成キーの取り出し対応の SQL ステートメントを準備する機能がサポートされています。

IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 またはバージョン 3.51 を使用する場合、SQL ステートメントを自動生成キーの取り出しのために準備し、バッチ更新に PreparedStatement オブジェクトを使用すると、SQLException が出されます。

IBM Data Server Driver for JDBC and SQLJ のバージョン 3.50 より前のバージョンは、自動生成キーを戻すように準備された PreparedStatement オブジェクトに対してアプリケーションが addBatch または executeBatch メソッドを呼び出すときに、SQLException をスローしません。ただし、PreparedStatement オブジェクトは自動生成キーを戻しません。

CURRENT CLIENT_ACCTNG 特殊レジスターの初期値

IBM Data Server Driver for JDBC and SQLJ バージョン 2.6 以降で実行される JDBC または SQLJ アプリケーションで、タイプ 4 接続が使用されている場合、DB2 for z/OS CURRENT CLIENT_ACCTNG 特殊レジスターの初期値は、 DB2 for z/OS のバージョン番号と clientWorkStation プロパティーの値とを連結したものです。 JDBC ドライバー、バージョン、または接続のいずれかがそれとは異なる場合、初期値は設定されていません。

スクロール可能 ResultSet および更新可能 ResultSet のサポート

IBM Data Server Driver for JDBC and SQLJ は、スクロール可能 ResultSet および更新可能 ResultSet をサポートします。

DB2 JDBC Type 2 ドライバーはスクロール可能 ResultSet をサポートしますが、更新可能 ResultSet はサポートしません。

URL 構文の相違点

DriverManager.getConnection メソッドの *url* パラメーターの構文はドライバーごとに異なります。詳細については、以下のトピックを参照してください。

- 『DriverManager インターフェースと IBM Data Server Driver for JDBC and SQLJ を使用したデータ・ソースへの接続』
- 『DriverManager インターフェースと DB2 JDBC Type 2 ドライバーを使用したデータ・ソースへの接続』

ドライバー・エラーについて戻されるエラー・コードと SQLSTATE の相違点

IBM Data Server Driver for JDBC and SQLJ は、他のドライバーとは異なり、内部エラーには既存の SQLCODE または SQLSTATE を使用しません。 『IBM Data

Server Driver for JDBC and SQLJ から発行されるエラー・コード』と『IBM Data Server Driver for JDBC and SQLJ から発行される SQLSTATE』を参照してください。

内部エラーが発生すると、z/OS 用の JDBC/SQLJ ドライバーは ODBC SQLSTATE を戻します。

エラー・メッセージ・テキストが戻される程度

IBM Data Server Driver for JDBC and SQLJ では、SQLException.getMessage() を実行する際に、retrieveMessagesFromServerOnGetMessage プロパティーが true に設定されていなければ、フォーマット済みメッセージ・テキストは戻されません。

DB2 JDBC Type 2 ドライバーでは、SQLException.getMessage() を実行すると、フォーマット済みメッセージ・テキストが戻されます。

セキュリティー・メカニズム

JDBC ドライバーには様々なセキュリティー・メカニズムがあります。

IBM Data Server Driver for JDBC and SQLJ セキュリティー・メカニズムについては、『IBM Data Server Driver for JDBC and SQLJ 使用時のセキュリティー』を参照してください。

DB2 JDBC Type 2 ドライバーのセキュリティー・メカニズムについては、『DB2 JDBC Type 2 ドライバー使用時のセキュリティー』を参照してください。

読み取り専用接続のサポート

IBM Data Server Driver for JDBC and SQLJ では、Connection または DataSource オブジェクトの readOnly プロパティーにより、接続を読み取り専用にすることができます。

DB2 JDBC Type 2 ドライバーは、接続を読み取り専用にするかどうかを判別する際に Connection.setReadOnly 値を使用します。ただし、Connection.setReadOnly(true) を設定しても、接続が読み取り専用になるとは限りません。

BIT DATA 列について ResultSet.getString から戻される結果

IBM Data Server Driver for JDBC and SQLJ は、CHAR FOR BIT DATA または VARCHAR FOR BIT DATA 列について ResultSet.getString 呼び出しからのデータを小文字の 16 進数ストリングとして戻します。

DB2 JDBC Type 2 ドライバーはデータを大文字の 16 進数ストリングとして戻します。

TIMESTAMP 列について ResultSet.getString から戻される結果

デフォルトで、 IBM Data Server Driver for JDBC and SQLJ は、ResultSet.getString 呼び出しによって TIMESTAMP 列値のデータを返すとき、後続ゼロを切り捨てます。この動作は、timestampPrecisionReporting プロパティーで変更することができます。

DB2 JDBC Type 2 ドライバーは、ResultSet.getString 呼び出しによって TIMESTAMP 列値のデータを返すとき、後続ゼロを切り捨てません。

どの行にも影響しない executeUpdate 呼び出しの結果

executeUpdate 呼び出しがどの行にも影響を与えない場合、 IBM Data Server Driver for JDBC and SQLJ は SQLWarning を生成します。

DB2 JDBC Type 2 ドライバーは SQLWarning を生成しません。

TIMESTAMP 列についての getDate または getTime 呼び出しの結果

getDate または getTime 呼び出しが TIMESTAMP 列に対して行われる場合、 IBM Data Server Driver for JDBC and SQLJ は SQLWarning を生成しません。

getDate または getTime 呼び出しが TIMESTAMP 列に対して行われる場合、DB2 JDBC Type 2 ドライバーは SQLWarning を生成します。

現実の日付および時刻に対応しない入出力値の日付および時刻の調整

SQL の DATE、TIME、または TIMESTAMP 列のデータを更新したり取り出したりする際に、現実の日付および時刻に対応しない日付および時刻の値は、 IBM Data Server Driver for JDBC and SQLJ によって調整されます。例えば、TIMESTAMP 列を更新して値を 2007-12-31 24:00:00.0 にしようとすると、 IBM Data Server Driver for JDBC and SQLJ によってその値は 2008-01-01 00:00:00.0 に調整されます。 TIMESTAMP 列を更新して値を 9999-12-31 24:00:00.0 にしようとすると、調整後の値 10000-01-01 00:00:00.0 が無効であるため、 IBM Data Server Driver for JDBC and SQLJ により例外がスローされます。

DB2 JDBC Type 2 Driver for Linux, UNIX and Windows では、現実の日付または時刻に対応しない日付または時刻の値に対して調整処理が実行されません。このドライバーでは、データベースとの間で値がそのままやり取りされます。例えば、DB2 JDBC Type 2 Driver for Linux, UNIX and Windows で TIMESTAMP 列を更新して値を 9999-12-31 24:00:00.0 にしようとしても、例外はスローされません。詳しくは、『JDBC および SQLJ アプリケーションで問題が発生する可能性のある日付、時刻、およびタイム・スタンプの値』を参照してください。

長さが一致しない PreparedStatement.setXXXStream について例外がスローされる場合

PreparedStatement.setBinaryStream、PreparedStatement.setCharacterStream、または PreparedStatement.setUnicodeStream メソッドを使用する場合、*length* パラメーター値は入力ストリーム内のバイト数に一致しなければなりません。

バイト数が一致しない場合、後続の PreparedStatement.executeUpdate メソッドが実行するまで、 IBM Data Server Driver for JDBC and SQLJ は例外をスローしません。そのため、 IBM Data Server Driver for JDBC and SQLJ では、長さが一致しないと、一部のデータがサーバーに送信されることがあります。そのデータはサーバーによって切り捨てられるかまたは埋め込まれます。呼び出し側アプリケーショ

ンはロールバック要求を出して、切り捨てられたデータまたは埋め込まれたデータを含むデータベース更新を取り消す必要があります。

DB2 JDBC Type 2 ドライバーは、PreparedStatement.setBinaryStream、PreparedStatement.setCharacterStream、または PreparedStatement.setUnicodeStream メソッドが実行した後、例外をスローします。

PreparedStatement.setXXXStream のデフォルト・マッピング

IBM Data Server Driver for JDBC and SQLJ では、PreparedStatement.setBinaryStream、PreparedStatement.setCharacterStream、または PreparedStatement.setUnicodeStream メソッドを使用しており、ターゲット列のデータ・タイプに関する情報を使用できない場合、入力データは BLOB または CLOB データ・タイプにマップされます。

DB2 JDBC Type 2 ドライバーの場合、入力データは VARCHAR FOR BIT DATA または VARCHAR データ・タイプにマップされます。

文字変換の実行方法

文字データがクライアントとサーバーとの間で転送される場合、データは受信側が処理できる形式に変換される必要があります。

IBM Data Server Driver for JDBC and SQLJ の場合、データ・ソースからクライアントに送信される文字データは Java の組み込み文字コンバーターを使用して変換されます。 IBM Data Server Driver for JDBC and SQLJ がサポートする変換は、基礎となる JRE インプリメンテーションによってサポートされるものに制限されます。

タイプ 4 接続を使用する IBM Data Server Driver for JDBC and SQLJ クライアントは、データを Unicode UTF-8 としてデータ・ソースに送信します。

DB2 JDBC Type 2 ドライバーでは、文字変換が DB2 サーバーによってサポートされる場合に変換を実行できます。

これらのドライバーはデータ・ソースからの CCSID 情報を使用します (使用可能な場合)。ドライバーは入力パラメーター・データを送信する前に、そのデータをデータ・ソースの CCSID に変換します。ターゲット CCSID 情報が使用不可の場合、ドライバーはデータを Unicode UTF-8 として送信します。

入力パラメーターの暗黙的または明示的データ・タイプ変換

PreparedStatement.setXXX メソッドを実行し、setXXX メソッドから生成されるデータ・タイプがパラメーター値が割り当てられている表列のデータ・タイプに一致しない場合、データ・タイプの変換が行われなければ、ドライバーはエラーを戻します。

IBM Data Server Driver for JDBC and SQLJ では、ターゲット・データ・タイプが認識されている場合、および `deferPrepares` と `sendDataAsIs` 接続プロパティーが `false` に設定されている場合に、正しい SQL データ・タイプへの変換が暗黙的に行われます。この場合、setXXX 呼び出しで暗黙値が明示値をオーバーライドします。 `deferPrepares` 接続プロパティーまたは `sendDataAsIs` 接続プロパティーが

true に設定されている場合、PreparedStatement.setObject メソッドを使用してパラメーターを正しい SQL データ・タイプに変換する必要があります。

DB2 JDBC Type 2 ドライバーの場合、パラメーターのデータ・タイプがそのデフォルト SQL データ・タイプに一致しなければ、PreparedStatement.setObject メソッドを使用してパラメーターを正しい SQL データ・タイプに変換する必要があります。

入力パラメーターの String から BINARY への変換のサポート

x がタイプ String のオブジェクトの場合、 IBM Data Server Driver for JDBC and SQLJ は以下の形式の PreparedStatement.setObject 呼び出しをサポートしません。

```
setObject(parameterIndex, x, java.sqlTypes.BINARY)
```

DB2 JDBC Type 2 ドライバーはこのタイプの呼び出しをサポートします。ドライバーは *x* の値を 16 進数ストリングと解釈します。

10 進スケールが一致しない PreparedStatement.setObject の結果

IBM Data Server Driver for JDBC and SQLJ の場合、10 進入力パラメーターを指定して PreparedStatement.setObject を呼び出すときに、入力パラメーターのスケールがターゲット列のスケールより大きければ、ドライバーは入力値の末尾桁を切り捨ててからその値を列に割り当てます。

DB2 JDBC Type 2 ドライバーは入力値の末尾桁を丸めてから、その値を列に割り当てます。

ResultSet.getBigDecimal スケール・パラメーターの有効範囲

推奨されない形式の ResultSet.getBigDecimal には、第 2 のパラメーターとして *scale* パラメーターがあります。 IBM Data Server Driver for JDBC and SQLJ では、0 から 32 の範囲のスケール・パラメーターを許可します。

DB2 JDBC Type 2 ドライバーは -1 から 32 の範囲を許可します。

入力パラメーター用の java.lang.Character データ・タイプからの変換のサポート

次の形式の PreparedStatement.setObject の場合、 IBM Data Server Driver for JDBC and SQLJ は *x* を JDBC データ・タイプに変換する際に、JDBC データ・タイプへの Java オブジェクトの標準データ・タイプ・マッピングをサポートします。

```
setObject(parameterIndex, x)
```

DB2 JDBC Type 2 ドライバーは 、java.lang.Character から CHAR への *x* の非標準マッピングをサポートします。

文字列に対する ResultSet.getBinaryStream のサポート

IBM Data Server Driver for JDBC and SQLJ は、文字列に FOR BIT DATA 属性が含まれる場合にのみ、その文字列を表す引数を指定した ResultSet.getBinaryStream をサポートします。

DB2 JDBC Type 2 ドライバーの場合、ResultSet.getBinaryStream 引数が文字列であれば、列には FOR BIT DATA 属性がなくてもかまいません。

バイナリー列に対する ResultSet.getBinaryStream から戻されるデータ

IBM Data Server Driver for JDBC and SQLJ では、ResultSet.getBinaryStream をバイナリー列に対して実行する場合、戻されるデータの形式は小文字の 16 進数字の対です。

DB2 JDBC Type 2 ドライバーでは、ResultSet.getBinaryStream をバイナリー列に対して実行する場合、戻されるデータの形式は大文字の 16 進数字の対です。

Boolean 入力タイプおよび CHAR ターゲット・タイプの setObject を使用した結果

IBM Data Server Driver for JDBC and SQLJ では、`PreparedStatement.setObject (parameterIndex,x,CHAR)` を実行しており、*x* が Boolean 値の場合、値 ″0″ または ″1″ が表列に挿入されます。

DB2 JDBC Type 2 ドライバーでは、ストリング ″false″ または ″true″ が表列に挿入されます。表列の長さは少なくとも 5 でなければなりません。

getBoolean を使用して CHAR 列から値を取り出した結果

IBM Data Server Driver for JDBC and SQLJ では、ResultSet.getBoolean または CallableStatement.getBoolean を実行して CHAR 列から Boolean 値を取り出すときに、列に値 ″false″ または ″0″ が含まれる場合、値 `false` が戻されます。列にそれ以外の値が含まれる場合、`true` が戻されます。

DB2 JDBC Type 2 ドライバーでは、ResultSet.getBoolean または CallableStatement.getBoolean を実行して CHAR 列から Boolean 値を取り出すときに、列に値 ″true″ または ″1″ が含まれる場合、値 `true` が戻されます。列にそれ以外の値が含まれる場合、`false` が戻されます。

閉じられたカーソルに対して ResultSet.next() を実行した結果

IBM Data Server Driver for JDBC and SQLJ では、閉じられたカーソルに対して ResultSet.next() を実行すると、SQLException がスローされます。これは JDBC 標準に準拠します。

DB2 JDBC Type 2 ドライバーでは、閉じられたカーソルに対して ResultSet.next() を実行すると、`false` の値が戻され、例外はスローされません。

DatabaseMetaData 呼び出しで NULL 引数を指定した結果

IBM Data Server Driver for JDBC and SQLJ では、JDBC 仕様で `null` が許可されている場合にのみ、DatabaseMetaData メソッド呼び出しで引数に `null` を指定できます。そうでない場合、例外がスローされます。

DB2 JDBC Type 2 ドライバーでは、`null` は、検索を絞り込むために引数を使用しないことを意味します。

メソッド引数の大文字への変換

IBM Data Server Driver for JDBC and SQLJ は、メソッド呼び出し内のどの引数も大文字に変換しません。

DB2 JDBC Type 2 ドライバーは、Statement.setCursorName 呼び出しの引数を大文字に変換します。カーソル名が大文字に変換されないようにするには、カーソル名の前と後に文字 ¥″ を付けます。例:

```
Statement.setCursorName("¥"mycursor¥"");
```

タイム・スタンプのエスケープ節のサポート

IBM Data Server Driver for JDBC and SQLJ は、TIME のエスケープ節として、以下の標準形式をサポートします。

```
{t 'hh:mm:ss'}
```

標準形式に加えて、DB2 JDBC Type 2 ドライバーは次の形式の TIME エスケープ節をサポートします。

```
{ts 'hh:mm:ss'}
```

ステートメント・バッチへの CALL ステートメントの組み込み

IBM Data Server Driver for JDBC and SQLJ は、ステートメント・バッチ内の CALL ステートメントをサポートします。

DB2 JDBC Type 2 ドライバーは、ステートメント・バッチ内の CALL ステートメントをサポートしません。

SQL ステートメント・テキストからの余分な文字の除去

IBM Data Server Driver for JDBC and SQLJ は SQL ステートメント・テキストをデータ・ソースに渡す前に、空白文字 (スペース、タブ、改行文字など) をそのテキストから除去しません。

DB2 JDBC Type 2 ドライバーは、SQL ステートメント・テキストをデータ・ソースに渡す前に、空白文字をそのテキストから除去します。

PreparedStatement.executeBatch を実行した結果

PreparedStatement.executeBatch ステートメントが IBM Data Server Driver for JDBC and SQLJ を使用して実行されると、ドライバーは更新カウントの int 配列を戻します。配列の各エレメントには、バッチ内のステートメントによって更新された行数が含まれます。

PreparedStatement.executeBatch ステートメントが DB2 JDBC Type 2 ドライバーを使用して実行されると、ドライバーは更新カウントを判別できないため、更新カウントごとに -3 を戻します。

コンパウンド SQL のサポート

IBM Data Server Driver for JDBC and SQLJ ドライバーはコンパウンド SQL ブロックをサポートしません。

コンパウンド SQL は複数の SQL ステートメントを単一の実行可能ブロックにグループ化できるようにします。以下に例を示します。

```
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
  UPDATE ACCOUNTS SET ABALANCE = ABALANCE + :delta
    WHERE AID = :aid;
  UPDATE TELLERS SET TBALANCE = TBALANCE + :delta
    WHERE TID = :tid;
  INSERT INTO TELLERS (TID, BID, TBALANCE) VALUES (:i, :branch_id, 0);
  COMMIT;
END COMPOUND;
```

DB2 JDBC Type 2 ドライバーは、PreparedStatement.executeUpdate または Statement.executeUpdate を使用してコンパウンド SQL ブロックの実行をサポートします。

バッチ更新でパラメーターを設定しない結果

パラメーターが設定されていない場合、 IBM Data Server Driver for JDBC and SQLJ ドライバーは PreparedStatement.addBatch 呼び出しの後で例外をスローします。

パラメーターがバッチ内のいずれのステートメントにも設定されていない場合、DB2 JDBC Type 2 ドライバーは PreparedStatement.executeBatch 呼び出しの後で例外をスローします。

アンカタログ・ストアード・プロシージャーを呼び出す機能

IBM Data Server Driver for JDBC and SQLJ ドライバーでは、DB2 カタログで定義されていないストアード・プロシージャーを呼び出すことはできません。

DB2 JDBC Type 2 ドライバーでは、DB2 カタログで定義されていないストアード・プロシージャーを呼び出すことができます。

ストアード・プロシージャー・パラメーターのデータ・タイプの指定

IBM Data Server Driver for JDBC and SQLJ ドライバーでは、データ・ソースが CALL ステートメントの動的実行をサポートしていない場合、ストアード・プロシージャー定義で指定されたとおりに**正確に** CALL ステートメント・パラメーターを指定する必要があります。

例えば、 DB2 for z/OS データ・ソースは CALL ステートメントの動的実行をサポートしません。 DB2 for z/OS サーバー上のストアード・プロシージャーの最初のパラメーターが、CREATE PROCEDURE ステートメントで次のように定義されているとします。

```
OUT PARM1 DECIMAL(3,0)
```

呼び出し側アプリケーションでは、cs.registerOutParameter(1, Types.DECIMAL) のようなステートメントは正しくありません。データ・タイプ cs.registerOutParameter (1, Types.DECIMAL, 0) だけでなくスケールを指定する形式の registerOutParameter メソッドを使用する必要があります。

DB2 JDBC Type 2 ドライバーでは、呼び出しアプリケーションのパラメーター・データ・タイプが CREATE PROCEDURE ステートメントのデータ・タイプと一致している必要はありません。

IBM Data Server Driver for JDBC and SQLJ のバージョン間での JDBC の相違点

JDBC アプリケーションを古いドライバーから IBM Data Server Driver for JDBC and SQLJ の新しいバージョンにアップグレードする前に、それらのドライバー間の相違点を理解する必要があります。

サポートされるメソッド

IBM Data Server Driver for JDBC and SQLJ がサポートするメソッドのリストについては、『JDBC API でのドライバーのサポート』を参照してください。

JDBC ドライバーによる連続ストリーミングの使用

IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 以降の場合、連続ストリーミング (動的データ形式ともいう) の動作は、 DB2 Database for Linux, UNIX, and Windows バージョン 9.5 以降への接続の場合の LOB 検索のデフォルトです。

連続ストリーミングは IBM Data Server Driver for JDBC and SQLJ バージョン 3.1 以降でサポートされています。ただし、 IBM Data Server Driver for JDBC and SQLJ バージョン 3.2 以降では、連続ストリーミングの動作は、 DB2 for z/OS バージョン 9.1 以降への接続の場合の LOB および XML 検索のデフォルトです。

IBM Data Server Driver for JDBC and SQLJ の以前のバージョンは、連続ストリーミングをサポートしません。

重要: 連続ストリーミングでは、LOB または XML の値を ResultSet からアプリケーション変数に取り出す場合、カーソルを移動するまで、または ResultSet 上のカーソルをクローズするまでは、そのアプリケーション変数の内容を操作できます。その後は、アプリケーション変数の内容は操作できなくなります。アプリケーション変数内の LOB に対して何らかのアクションを実行すると、SQLException を受け取ります。例えば、連続ストリーミングが有効で、以下のようなステートメントを実行するとします。

```
...
ResultSet rs = stmt.executeQuery("SELECT CLOBCOL FROM MY_TABLE");
rs.next();                // Retrieve the first row of the ResultSet
Clob clobFromRow1  = rs.getClob(1);
                          // Put the CLOB from the first column of
                          // the first row in an application variable
String substr1Clob = clobFromRow1.getSubString(1,50);
                          // Retrieve the first 50 bytes of the CLOB
rs.next();                // Move the cursor to the next row.
                          // clobFromRow1 is no longer available.
// String substr2Clob = clobFromRow1.getSubString(51,100);
                          // This statement would yield an SQLException
Clob clobFromRow2  = rs.getClob(1);
                          // Put the CLOB from the first column of
```

```
                        // the second row in an application variable
rs.close();             // Close the ResultSet.
                        // clobFromRow2 is also no longer available.
```

rs.next() を実行してカーソルを ResultSet の 2 番目の行に置いた後は、clobFromRow1 内の CLOB 値は使用できなくなります。同様に、rs.close() を実行して ResultSet をクローズした後、clobFromRow1 および clobFromRow2 内の値は使用できなくなります。

この動作の変更によるエラーを避けるには、以下のアクションのいずれかをとる必要があります。

- アプリケーションを変更します。

  LOB データをアプリケーション変数に取り出すアプリケーションは、データの取り出しに使用されたカーソルが移動されるか閉じられるまでの間に限り、それらのアプリケーション変数のデータを操作できます。
- 連続ストリーミングを使用不可にします。それには、progressiveStreaming プロパティーを DB2BaseDataSource.NO (2) に設定します。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の ResultSetMetaData 値

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の場合、ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel のデフォルトの動作は、それ以前の JDBC ドライバーのデフォルトの動作とは異なります。

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降を使用する必要があるが、アプリケーションが旧バージョンの JDBC ドライバーで戻された ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel の値を戻す必要がある場合、useJDBC4ColumnNameAndLabelSemantics Connection および DataSource プロパティーを DB2BaseDataSource.NO (2) に設定できます。

自動生成キーを使用したバッチ更新の結果はドライバーのバージョンによって異なる

IBM Data Server Driver for JDBC and SQLJ バージョン 3.52 以降では、自動生成キーの取り出し対応の SQL ステートメントを準備する機能がサポートされています。

IBM Data Server Driver for JDBC and SQLJ バージョン 3.50 またはバージョン 3.51 を使用する場合、SQL ステートメントを自動生成キーの取り出しのために準備し、バッチ更新に PreparedStatement オブジェクトを使用すると、SQLException が出されます。

IBM Data Server Driver for JDBC and SQLJ のバージョン 3.50 より前のバージョンは、自動生成キーを戻すように準備された PreparedStatement オブジェクトに対してアプリケーションが addBatch または executeBatch メソッドを呼び出すときに、SQLException をスローしません。ただし、PreparedStatement オブジェクトは自動生成キーを戻しません。

CURRENT CLIENT_ACCTNG 特殊レジスターの初期値

IBM Data Server Driver for JDBC and SQLJ バージョン 2.6 以降で実行される JDBC または SQLJ アプリケーションで、タイプ 4 接続が使用されている場合、DB2 for z/OS CURRENT CLIENT_ACCTNG 特殊レジスターの初期値は、 DB2 for z/OS のバージョン番号と clientWorkStation プロパティーの値とを連結したものです。 JDBC ドライバー、バージョン、または接続のいずれかがそれとは異なる場合、初期値は設定されていません。

複数行 FETCH の使用を制御するプロパティー

IBM Data Server Driver for JDBC and SQLJ のバージョン 3.7 およびバージョン 3.51 より前は、複数行 FETCH のサポートの有効および無効の制御は useRowsetCursor プロパティーで制御され、それは両方向スクロール・カーソルの場合にのみ使用可能であり、しかも DB2 for z/OS に対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続 の場合にのみ利用可能でした。バージョン 3.7 および 3.51 以降は、次のようになります。

- DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合、 IBM Data Server Driver for JDBC and SQLJ において、両方向スクロールまたは前方スクロール・カーソルで複数行 FETCH を使用するかどうかを決めるのに、enableRowsetSupport プロパティーだけが使用されます。
- DB2 for z/OS または DB2 Database for Linux, UNIX, and Windows に対する IBM Data Server Driver for JDBC and SQLJ Type 4 接続、または DB2 Database for Linux, UNIX, and Windows 上の IBM Data Server Driver for JDBC and SQLJ Type 2 接続 の場合、 IBM Data Server Driver for JDBC and SQLJ では、enableRowsetSupport プロパティーが設定されているなら、両方向スクロール・カーソルで複数行 FETCH を使用するかどうかを決めるのにその enableRowsetSupport が使用されます。 enableRowsetSupport が設定されていない場合、ドライバーでは、useRowsetCursor プロパティーを使用することによって、複数行 FETCH を使用するかどうかが決定されます。

JDBC 1 位置指定の更新および削除と複数行 FETCH

IBM Data Server Driver for JDBC and SQLJ のバージョン 3.7 およびバージョン 3.51 より前は、 DB2 for z/OS の表からの複数行 FETCH は、useRowsetCursor プロパティーによって制御されていました。アプリケーションに JDBC 1 位置指定更新または削除の操作が含まれていた場合、かつ複数行 FETCH サポートが有効であった場合、 IBM Data Server Driver for JDBC and SQLJ では、その更新または削除の操作が許容されていましたが、その更新または削除で予期しない結果が発生することがありました。

IBM Data Server Driver for JDBC and SQLJ のバージョン 3.7 および 3.51 以降では、 DB2 for z/OS の表または DB2 Database for Linux, UNIX, and Windows の表からの複数行 FETCH が有効か無効かは、enableRowsetSupport プロパティーによって制御されます。 enableRowsetSupport プロパティーは、useRowsetCursor プロパティーをオーバーライドします。 enableRowsetSupport プロパティーによって複数行 FETCH が有効になっていて、アプリケーションに JDBC 1 位置指定の更新または削除の操作が含まれている場合、 IBM Data Server Driver for JDBC and SQLJ により SQLException がスローされます。

ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel の値の例

IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の場合、ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel のデフォルトの動作は、それ以前の JDBC ドライバーのデフォルトの動作とは異なります。この動作は、useJDBC4ColumnNameAndLabelSemantics プロパティーを使用して変更することができます。

次の例は、 IBM Data Server Driver for JDBC and SQLJ バージョン 4.0、および useJDBC4ColumnNameAndLabelSemantics プロパティーが設定されていない場合に以前の JDBC ドライバーに戻される値を示しています。

すべての照会は、次のように定義されている表を使用します。

```
CREATE TABLE MYTABLE(INTCOL INT)
```

例: 次の照会には AS 節が含まれます。これは、結果セット内の列のラベルを定義するものです。

```
SELECT MYCOL AS MYLABEL FROM MYTABLE
```

次の表は、照会について戻される ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnName の値をリストしています。

表 101. AS 節を含む照会の、 IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 より前および後の ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnName

ターゲットのデータ・ソース	IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 より前の動作		IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の動作	
	getColumnName 値	getColumnLabel 値	getColumnName 値	getColumnLabel 値
DB2 Database for Linux, UNIX, and Windows	MYLABEL	MYLABEL	MYCOL	MYLABEL
IBM Informix Dynamic Server	MYLABEL	MYLABEL	MYCOL	MYLABEL
DB2 for z/OS バージョン 8 以降、および DB2 UDB for iSeries V5R3 以降	MYLABEL	MYLABEL	MYCOL	MYLABEL
DB2 for z/OS バージョン 7、および DB2 UDB for iSeries V5R2	MYLABEL	MYLABEL	MYLABEL	MYLABEL

例: 次の照会には AS 節が含まれていません。

```
SELECT MYCOL FROM MYTABLE
```

照会に対する ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnLabel メソッドは、ターゲット・データ・ソースに関係なく MYCOL を戻します。

例: DB2 for z/OS またはDB2 for i データ・ソースでは、列のラベルを定義するために次のように LABEL ON ステートメントが使用されます。

```
LABEL ON COLUMN MYTABLE.MYCOL IS 'LABELONCOL'
```

次の照会には AS 節が含まれます。これは、ResultSet 内の列のラベルを定義するものです。

```
SELECT MYCOL AS MYLABEL FROM MYTABLE
```

次の表は、照会について戻される ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnName の値をリストしています。

表 102. AS 節を含む照会で LABEL ON ステートメントを指定した表列の、 IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 より前および後の ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnName

ターゲットのデータ・ソース	IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 より前の動作		IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 以降の動作	
	getColumnName 値	getColumnLabel 値	getColumnName 値	getColumnLabel 値
DB2 for z/OS バージョン 8 以降、および DB2 UDB for iSeries V5R3 以降	MYLABEL	LABELONCOL	MYCOL	MYLABEL
DB2 for z/OS バージョン 7、および DB2 UDB for iSeries V5R2	MYLABEL	LABELONCOL	MYCOL	LABELONCOL

例: DB2 for z/OS またはDB2 for i データ・ソースでは、列のラベルを定義するために次のように LABEL ON ステートメントが使用されます。

```
LABEL ON COLUMN MYTABLE.MYCOL IS 'LABELONCOL'
```

次の照会には AS 節が含まれていません。

```
SELECT MYCOL FROM MYTABLE
```

次の表は、照会について戻される ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnName の値をリストしています。

表 103. AS 節を含まない照会で LABEL ON ステートメントを指定した表列の、 IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 より前および後の ResultSetMetaData.getColumnName および ResultSetMetaData.getColumnName

ターゲットのデータ・ソース	IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 より前の動作		IBM Data Server Driver for JDBC and SQLJ バージョン 4.0 の動作	
	getColumnName 値	getColumnLabel 値	getColumnName 値	getColumnLabel 値
DB2 for z/OSバージョン 8 以降、および DB2 UDB for i5/OS V5R3 以降	MYCOL	LABELONCOL	MYCOL	MYCOL
DB2 for z/OSバージョン 7、および DB2 UDB for i5/OS V5R2	MYCOL	LABELONCOL	MYLABEL	LABELONCOL

IBM Data Server Driver for JDBC and SQLJ と他の DB2 JDBC ドライバーとの SQLJ の相違点

IBM Data Server Driver for JDBC and SQLJ とそれより前の JDBC ドライバーにはいくつかの相違点があります。 IBM Data Server Driver for JDBC and SQLJ に移行する場合、これらの相違点を考慮するように SQLJ プログラムを変更する必要があります。

重要: Linux、UNIX、および Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) は、推奨されていません。この情報は、アプリケーションの IBM Data Server Driver for JDBC and SQLJ への移行を支援するために提供されています。

IBM Data Server Driver for JDBC and SQLJ での SQLJ サポートは、以下の分野でその他の DB2 JDBC ドライバーでの SQLJ サポートと異なります。

db2sqljcustomize エラーおよび -collection パラメーター

IBM Data Server Driver for JDBC and SQLJ の一部である db2sqljcustomize ユーティリティーには -collection パラメーターがあります。 DB2 JDBC Type 2 ドライバーの一部である db2profc ユーティリティーには -collection パラメーターがありません。 db2sqljcustomize ユーティリティーが DB2 for z/OS サーバー上でバインド操作を実行する場合、-collection パラメーターに小文字が含まれていれば、db2sqljcustomize は -4499 エラーを戻します。 DB2 for z/OS ではコレクション ID に小文字を含めることはできないからです。この状況は、db2profc では発生しません。

シリアライズド・プロファイルでの相違点

DB2 JDBC Type 2 ドライバーと IBM Data Server Driver for JDBC and SQLJ は、SQLJ 変換プログラムおよび SQLJ カスタマイザー・ユーティリティーを実行する際に、異なるバイナリー・コードを生成します。そのため、DB2 JDBC Type 2 ドライバー `sqlj` および `db2profc` ユーティリティーを使用して変換およびカスタマイズされた SQLJ アプリケーションは 、 IBM Data Server Driver for JDBC and SQLJ の使用時には実行しません。 ***IBM Data Server Driver for JDBC and SQLJ でそれらの SQLJ アプリケーションを実行するには、その前に IBM Data Server Driver for JDBC and SQLJ `sqlj` および `db2sqljcustomize` ユーティリティーを使用してアプリケーションを再変換および再カスタマイズする必要があります。***これは、アプリケーションを変更していない場合でも行わなければなりません。

SQL VALUES サポート

DB2 JDBC Type 2 ドライバーは SQLJ ステートメント節で SQL VALUES ステートメントをサポートしますが、 IBM Data Server Driver for JDBC and SQLJ はサポートしません。そのため、VALUES ステートメントを含む SQLJ アプリケーションを変更する必要があります。

例: SQLJ プログラムに以下のステートメントが含まれるとします。

```
#sql [ctxt] hv = {VALUES (MY_ROUTINE(1))};
```

IBM Data Server Driver for JDBC and SQLJ の場合、そのステートメントを次のように変更する必要があります。

```
#sql [ctxt] {SELECT MY_ROUTINE(1) INTO :hv FROM SYSIBM.SYSDUMMY1};
```

接続手法の相違点

使用可能な接続手法およびそれらの接続手法に使用されるドライバー名と URL は、ドライバーによって異なります。詳しくは、『SQLJ を使用したデータ・ソースへの接続』を参照してください。

スクロール可能イテレーターおよび更新可能イテレーターのサポート

IBM Data Server Driver for JDBC and SQLJ での SQLJ はスクロール可能イテレーターおよび更新可能イテレーターをサポートします。

Linux 、UNIX 、Windows 用の DB2 JDBC Type 2 ドライバー (DB2 JDBC Type 2 ドライバー) は、両方向スクロール・カーソルをサポートしますが、更新可能イテレーターはサポートしません。

WebSphere Application Server での SQL ステートメントの動的実行

WebSphere Application Server バージョン 5.0.1 以上では、SQLJ プログラムをカスタマイズすれば、SQL ステートメントは静的に実行されます。

db2sqljcustomize および db2sqljprint の代替名はサポートされない

DB2 JDBC Type 2 ドライバーは、元々は、SQLJ プロファイル・カスタマイザー・コマンドでは db2profc という名前を、SQLJ プロファイル・プリンター・コマンドでは db2profp という名前を使用していました。 IBM Data Server Driver for JDBC and SQLJ では、SQLJ プロファイル・カスタマイザー・コマンドの名前は db2sqljcustomize で、SQLJ プロファイル・プリンター・コマンドの名前は db2sqljprint です。 DB2 Database for Linux, UNIX, and Windows の前のリリースでは、 db2profc は db2sqljcustomize の代替名として受け入れられ、db2profp は db2sqljprint の代替名として受け入れられていました。これらの代替名は受け入れられなくなりました。

IBM Data Server Driver for JDBC and SQLJ に影響を与える SDK for Java の相違点

SDK for Java のバージョン間での動作の相違点は、 IBM Data Server Driver for JDBC and SQLJ の下でプログラムを実行するときに受け取る結果のばらつきの原因となる場合があります。

DBCS 置換文字の取り出された値

DBCS 置換文字 (コード・ページ Cp943 では X'FCFC' など) をデータベース表から取り出す場合、取り出される値は、IBM SDK for Java を使用するかまたは Sun SDK for Java を使用するかによって異なります。

Sun SDK for Java の場合、置換文字は U+0000 として取り出されます。 IBM SDK for Java の場合、置換文字は X'FFFD' として取り出されます。

サポートされるコード・ページ

IBM SDK for Java は Sun SDK for Java よりも多くの DBCS コード・ページをサポートします。そのため、Sun SDK for Java でのサポートされないコード・ページのためにエラーが発生する場合は、IBM SDK for Java を使用してください。

暗号化のための IBM SDK for Java 要件

IBM SDK for Java は 256 ビット暗号化をサポートしますが、Sun SDK for Java にはこのサポートがありません。そのため、暗号化を含む IBM Data Server Driver for JDBC and SQLJ セキュリティー・メカニズムのいずれかを使用する場合は、IBM SDK for Java を使用する必要があります。

システム・モニターのサポート

IBM Data Server Driver for JDBC and SQLJ でのシステム・モニターのサポートには、コア・ドライバー時間とネットワーク I/O 時間のコレクションが含まれます。この情報を取り出すには、SDK for Java バージョン 5 またはそれ以降に含まれている機能が必要です。ただし、IBM SDK for Java バージョン 1.4.2 にも、コア・ドライバー時間とネットワーク I/O 時間のコレクションを使用可能にするサポートがあります。 IBM SDK for Java バージョン 1.4.2 を使用する場合、コア・ドライバー時間とネットワーク I/O 時間はその値に最も近いマイクロ秒に丸められます。SDK for Java バージョン 5 またはそれ以降を使用する場合、コア・ドライバー時間とネットワーク I/O 時間はその値に最も近いナノ秒に丸められます。

IBM Data Server Driver for JDBC and SQLJ から発行されるエラー・コード

+4200 から +4299、+4450 から +4499、-4200 から -4299、および -4450 から -4499 の範囲のエラー・コードは、 IBM Data Server Driver for JDBC and SQLJ 用に予約されています。

IBM Data Server Driver for JDBC and SQLJ エラーが発生した後で SQLException.getMessage メソッドを呼び出すと、以下のものを含むストリングが戻されます。

- 接続がタイプ 2 であるかまたはタイプ 4 接続であるか
- IBM ソフトウェア・サポートの診断情報
- ドライバーのレベル
- 説明のためのメッセージ
- エラー・コード
- SQLSTATE

以下に例を示します。

```
[jcc][t4][20128][12071][3.50.54] Invalid queryBlockSize specified: 1,048,576,012.
Using default query block size of 32,767.  ERRORCODE=0, SQLSTATE=
```

現在、 IBM Data Server Driver for JDBC and SQLJ は以下のエラー・コードを発行します。

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
+4204	RETURN DATA UNTIL 節に指定されているとおり、エラーが検出されて許容されました。 **説明:** 許容されたエラーには、フェデレーテッド接続、認証、および許可エラーが含まれます。この警告は、 DB2 Database for Linux, UNIX, and Windows サーバーへの接続にのみ適用されます。これは、カーソル操作 (ResultSet.next または ResultSet.previous 呼び出しなど) で `false` が戻される場合にのみ発行されます。	02506
+4222	*text-from-getMessage* **説明:** データ・ソースへの接続中に警告状態が生じました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4223	*text-from-getMessage* **説明:** 初期化中に警告状態が生じました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4225	*text-from-getMessage* **説明:** データがサーバーへ送信されたか、サーバーから受信されたときに警告状態が生じました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4226	*text-from-getMessage* **説明:** カスタマイズまたはバインド中に警告状態が生じました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4228	*text-from-getMessage* **説明:** 他のカテゴリーに該当しない警告状態が生じました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4450	機能がサポートされていません: *feature-name*	
+4460	*text-from-getMessage* **説明:** 指定値は有効なオプションではありません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード (続き)

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
+4461	*text-from-getMessage* **説明:** 指定値は範囲外または無効です。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4462	*text-from-getMessage* **説明:** 必須値が欠落しています。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4470	*text-from-getMessage* **説明:** ターゲット・リソースが閉じられているため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4471	*text-from-getMessage* **説明:** ターゲット・リソースが使用中であるため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4472	*text-from-getMessage* **説明:** ターゲット・リソースが使用できないため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
+4474	*text-from-getMessage* **説明:** ターゲット・リソースが変更不可であるため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4200	操作が無効です: Global Transaction 中に XA 環境で無効な COMMIT または ROLLBACK が呼び出されました。 **説明:** XA 環境のグローバル・トランザクション中のアプリケーションがコミットまたはロールバックを発行しました。グローバル・トランザクション中のコミットまたはロールバック操作は無効です。	2D521

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード (続き)

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
-4201	操作が無効です: Global Transaction 中は setAutoCommit(true) が許可されていません。 **説明:** XA 環境のグローバル・トランザクション中のアプリケーションが setAutoCommit(true) ステートメントを実行しました。グローバル・トランザクションで setAutoCommit(true) を発行することは無効です。	2D521
-4203	*function* の実行エラー。サーバーは *rc* を戻しました。 **説明:** SQL ステートメントの実行中に XA 接続でエラーが発生しました。 ネットワークの最適化のために、 IBM Data Server Driver for JDBC and SQLJ は次の SQL ステートメントが実行されるまで一部の XA フローを遅らせます。遅延した XA フローでエラーが発生する場合、そのエラーは現行の SQL ステートメントによってスローされる SQLException の一部として報告されます。	
-4210	プールからトランスポート・オブジェクトを取得中にタイムアウトになりました。	57033
-4211	プールからオブジェクトを取得中にタイムアウトになりました。	57033
-4212	Sysplex メンバーが使用できません。	
-4213	タイムアウト。	57033
-4214	*text-from-getMessage* **説明:** 許可に失敗しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	28000
-4220	*text-from-getMessage* **説明:** 文字変換中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4221	*text-from-getMessage* **説明:** 暗号化または暗号化解除中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード (続き)

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
-4222	*text-from-getMessage* **説明:** データ・ソースへの接続中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4223	*text-from-getMessage* **説明:** 初期化中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4224	*text-from-getMessage* **説明:** リソースのクリーンアップ中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4225	*text-from-getMessage* **説明:** データがサーバーへ送信されたか、サーバーから受信されたときにエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4226	*text-from-getMessage* **説明:** カスタマイズまたはバインド中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4227	*text-from-getMessage* **説明:** リセット中にエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4228	*text-from-getMessage* **説明:** 他のカテゴリーに該当しないエラーが発生しました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4450	機能がサポートされていません: *feature-name*	0A504

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード (続き)

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
-4460	*text-from-getMessage* **説明:** 指定値は有効なオプションではありません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4461	*text-from-getMessage* **説明:** 指定値は範囲外または無効です。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	42815
-4462	*text-from-getMessage* **説明:** 必須値が欠落しています。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4463	*text-from-getMessage* **説明:** 指定された値に構文エラーがあります。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	42601
-4470	*text-from-getMessage* **説明:** ターゲット・リソースが閉じられているため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4471	*text-from-getMessage* **説明:** ターゲット・リソースが使用中であるため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4472	*text-from-getMessage* **説明:** ターゲット・リソースが使用不可であるため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード (続き)

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
-4473	*text-from-getMessage* **説明:** ターゲット・リソースが使用できなくなったため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4474	*text-from-getMessage* **説明:** ターゲット・リソースが変更不可であるため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4475	*text-from-getMessage* **説明:** ターゲット・リソースへのアクセスが制限されているため、要求された操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4476	*text-from-getMessage* **説明:** 要求された操作はターゲット・リソースに対しては許可されていないため、この操作を実行できません。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	
-4496	保留カーソルの SQL OPEN が XA 接続に対して発行されました。JDBC ドライバーでは、XA 接続の場合にデータベース・サーバー上で保留カーソルをオープンすることはできません。	
-4497	アプリケーションはロールバックを発行する必要があります。作業単位は既に DB2 サーバーでロールバックされましたが、その作業単位に関与する他のリソース・マネージャーが、変更内容をまだロールバックしていない可能性があります。このアプリケーションの保全性を確保するために、アプリケーションがロールバックを発行するまで、すべての SQL 要求が拒否されます。	

表 104. IBM Data Server Driver for JDBC and SQLJ により発行されるエラー・コード (続き)

エラー・コード	メッセージ・テキストおよび説明	SQLSTATE
-4498	接続は失敗しましたが、再確立されました。ホスト名または IP アドレス: *host-name*、サービス名またはポート番号: *port*、特殊レジスター変更インディケーター: *rc*。 **説明:** *host-name* および *port* は、接続が再確立されたデータ・ソースを示します。 *rc* は、特殊レジスター値を設定した SQL ステートメントが再実行されたかどうかを以下のように示します。 **1** 特殊レジスター値を設定した SQL ステートメントは再実行された。 **2** 特殊レジスター値を設定した SQL ステートメントは再実行されなかった可能性がある。 DB2 for z/OS サーバーに対するクライアント・リルートの場合、最後のコミット・ポイント後に設定された特殊レジスター値は再確立されません。 アプリケーションは以前のコミット・ポイントにロールバックされます。接続状態、グローバル・リソース (例えばグローバル一時表)、およびオープン状態の保留カーソルは維持されない可能性があります。	
-4499	*text-from-getMessage* **説明:**致命的エラーが発生したため、データ・ソースから切断されました。既存の接続が使用不能になりました。 **ユーザーの対応:** SQLException.getMessage を呼び出して、この問題に関する具体的な情報を取得します。	08001 または 58009
-30108	Sysplex においてクライアント・リルート例外が発生しました。	08506
-99999	IBM Data Server Driver for JDBC and SQLJ がまだエラー・コードのないエラーを出しました。	

IBM Data Server Driver for JDBC and SQLJ から発行される SQLSTATE

46600 から 466ZZ の範囲の SQLSTATE は IBM Data Server Driver for JDBC and SQLJ 用に予約されています。

次の表は、 IBM Data Server Driver for JDBC and SQLJ によって生成または使用される SQLSTATE をリストしています。

表 105. IBM Data Server Driver for JDBC and SQLJ により戻される SQLSTATE

SQLSTATE クラス	SQLSTATE	説明
01*xxx*		警告
02*xxx*		データなし
	02501	カーソル位置が現在行の FETCH に対して無効です。
	02506	容認できるエラー
08*xxx*		接続例外
	08003	接続が存在しません。
	08004	アプリケーション・サーバーが、接続の確立を拒否しました。
	08506	クライアント・リルートの例外です。
0A*xxx*		機能がサポートされていない
	0A502	このデータベース・インスタンスではアクションまたは操作ができません。
	0A504	この機能はドライバーによってサポートされません
22*xxx*		データ例外
	22007	日時値のストリング表記が無効です。
	22021	文字がコード化文字セットにありません。
23*xxx*		制約違反
	23502	列に挿入されるかまたは列を更新する値が NULL ですが、列に NULL 値を入れることはできません。
24*xxx*		カーソル状態が無効
	24501	識別されたカーソルがオープンしていません。
28*xxx*		許可例外
	28000	許可名が無効です。
2D*xxx*		トランザクション終了が無効
	2D521	SQL COMMIT または ROLLBACK が、現在の操作環境では無効です。
34*xxx*		カーソル名が無効
	34000	カーソル名が無効です。
3B*xxx*		SAVEPOINT が無効
	3B503	SAVEPOINT、RELEASE SAVEPOINT、または ROLLBACK TO SAVEPOINT は、トリガーまたはグローバル・トランザクションでは許可されていません。
40*xxx*		トランザクションのロールバック
42*xxx*		構文エラーまたはアクセス規則違反
	42601	文字、トークン、または節が、無効もしくは欠けています。
	42734	重複するパラメーター名、SQL 変数名、カーソル名、条件名、またはラベルが検出されました。
	42807	INSERT、UPDATE、または DELETE は、このオブジェクトでは許可されません。
	42808	INSERT または UPDATE 操作で識別された列は更新できません。

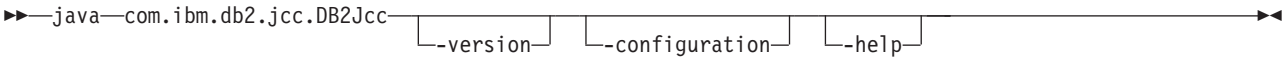
表 105. IBM Data Server Driver for JDBC and SQLJ により戻される SQLSTATE (続き)

SQLSTATE クラス	SQLSTATE	説明
	42815	データ・タイプ、長さ、位取り、値、または CCSID が無効です。
	42820	数値定数が長すぎるか、またはそのデータ・タイプの範囲内にない値を持っています。
	42968	現行ソフトウェア・ライセンスがないため、接続が失敗しました。
57*xxx*		リソースを使用できない、またはオペレーター介入
	57033	自動ロールバックなしで、デッドロックまたはタイムアウトが発生しました
58*xxx*		システム・エラー
	58008	分散プロトコル・エラーのため、実行が失敗しました。このエラーは、後続の DDM コマンドまたは SQL ステートメントの正常な実行には影響しません。
	58009	会話の割り振り解除の原因となる分散プロトコル・エラーのため、実行が失敗しました。
	58012	指定したパッケージ名と整合性トークンを持つバインド処理がアクティブではありません。
	58014	DDM コマンドはサポートされていません。
	58015	DDM オブジェクトはサポートされていません。
	58016	DDM パラメーターはサポートされていません。
	58017	DDM パラメーターの値がサポートされていません。

IBM Data Server Driver for JDBC and SQLJ のバージョンおよび環境の情報を確認する方法

IBM Data Server Driver for JDBC and SQLJ のバージョンや、ドライバーが実行される環境に関する情報を判別するには 、 コマンド行で DB2Jcc ユーティリティーを実行します。

DB2Jcc の構文



DB2Jcc オプションの説明

-version

IBM Data Server Driver for JDBC and SQLJ がその名前とバージョンを表示することを指定します。

-configuration

IBM Data Server Driver for JDBC and SQLJ がその名前とバージョン、その環境に関する情報 (Java ランタイム環境、オペレーティング・システム、パス情報、およびライセンス制限など) を表示することを指定します。

-help

DB2Jcc ユーティリティーがサポートする各オプションを記述することを指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

DB2Jcc の出力例

以下の出力は、-configuration パラメーターを指定して DB2Jcc を呼び出した結果です。

図 57. サンプル DB2Jcc 出力

```
(myid@mymachine) /home/myusrid $ java com.ibm.db2.jcc.DB2Jcc -version
[jcc] Driver: IBM DB2 JDBC Universal Driver Architecture 3.50.137

(myid@mymachine) /home/myusrid $ java com.ibm.db2.jcc.DB2Jcc -configuration
[jcc] BEGIN TRACE_DRIVER_CONFIGURATION
[jcc] Driver: IBM DB2 JDBC Universal Driver Architecture 3.50.137
[jcc] Compatible JRE versions: { 1.4, 1.5 }
[jcc] Target server licensing restrictions: { z/OS: enabled; SQLDS: enabled; iSe
ries: enabled; DB2 for Unix/Windows: enabled; Cloudscape: enabled; Informix: ena
bled }
[jcc] Range checking enabled: true
[jcc] Bug check level: 0xff
[jcc] Default fetch size: 64
[jcc] Default isolation: 2
[jcc] Collect performance statistics: false
[jcc] No security manager detected.
[jcc] Detected local client host: lead.svl.ibm.com/9.30.10.102
[jcc] Access to package sun.io is permitted by security manager.
[jcc] JDBC 1 system property jdbc.drivers = null
[jcc] Java Runtime Environment version 1.4.2
[jcc] Java Runtime Environment vendor = IBM Corporation
[jcc] Java vendor URL = http://www.ibm.com/
[jcc] Java installation directory = /wsdb/v91/bldsupp/AIX5L64/jdk1.4.2_sr1/sh/..
/jre
[jcc] Java Virtual Machine specification version = 1.0
[jcc] Java Virtual Machine specification vendor = Sun Microsystems Inc.
[jcc] Java Virtual Machine specification name = Java Virtual Machine Specificati
on
[jcc] Java Virtual Machine implementation version = 1.4.2
[jcc] Java Virtual Machine implementation vendor = IBM Corporation
[jcc] Java Virtual Machine implementation name = Classic VM
[jcc] Java Runtime Environment specification version = 1.4
[jcc] Java Runtime Environment specification vendor = Sun Microsystems Inc.
[jcc] Java Runtime Environment specification name = Java Platform API Specificat
ion
[jcc] Java class format version number = 48.0
[jcc] Java class path = :.:/home2/myusrid/sqllib/java/db2java.zip:/lib/classes.z
ip:/home2/myusrid/sqllib/java/sqlj.zip:./test:/home2/myusrid/sqllib/java/db2jcc.
jar:/home2/myusrid/sqllib/java/db2jcc_license_cisuz.jar:...
[jcc] Java native library path = /wsdb/v91/bldsupp/AIX5L64/jdk1.4.2_sr1/sh/../jr
e/bin:/wsdb/v91/bldsupp/AIX5L64/jdk1.4.2_sr1/jre/bin/classic:/wsdb/v91/bldsupp/A
IX5L64/jdk1.4.2_sr1/jre/bin:/home2/myusrid/sqllib/lib:/local/cobol:/home2/myusri
d/sqllib/samples/c:/usr/lib
[jcc] Path of extension directory or directories = /wsdb/v91/bldsupp/AIX5L64/jdk
1.4.2_sr1/sh/../jre/lib/ext
[jcc] Operating system name = AIX
[jcc] Operating system architecture = ppc64
[jcc] Operating system version = 5.3
[jcc] File separator ("/" on UNIX) = /
[jcc] Path separator (":" on UNIX) = :
[jcc] User's account name = myusrid
[jcc] User's home directory = /home2/myusrid
```

```
[jcc] User's current working directory = /home2/myusrid
[jcc] Dumping all system properties: { java.assistive=ON, java.runtime.name=Java
(TM) 2 Runtime Environment, Standard Edition, sun.boot.library.path=/wsdb/v91/bl
dsupp/AIX5L64/jdk1.4.2_sr1/sh/../jre/bin, java.vm.version=1.4.2, java.vm.vendor=
IBM Corporation, java.vendor.url=http://www.ibm.com/, path.separator=:, java.vm.
name=Classic VM, file.encoding.pkg=sun.io, user.country=US, sun.os.patch.level=u
nknown, ... }
[jcc] Dumping all file properties: {  }
[jcc] END TRACE_DRIVER_CONFIGURATION
```

SQLJ プログラムの準備のためのコマンド

SQLJ プログラムを実行用に準備するには、コマンドを使用して、SQLJ ソース・コードを Java ソース・コードに変換し、その Java ソース・コードをコンパイルし、SQLJ シリアライズド・プロファイルを作成してカスタマイズし、DB2 パッケージをバインドします。

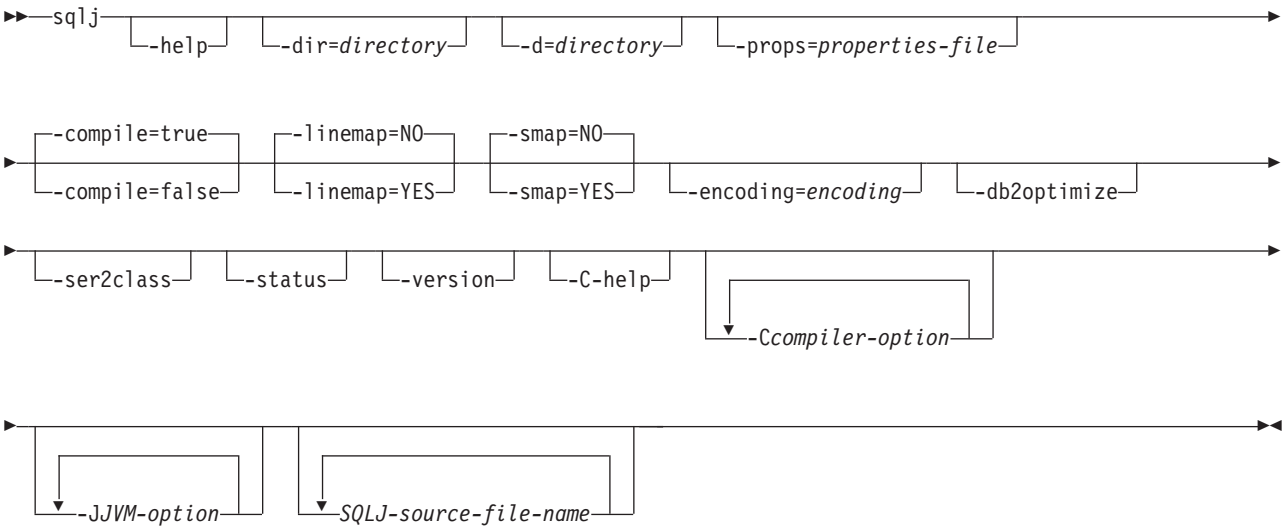
sqlj - SQLJ 変換プログラム

sqlj コマンドは SQLJ ソース・ファイルを Java ソース・ファイルおよび 0 個以上の SQLJ シリアライズド・プロファイルに変換します。デフォルトでは、sqlj コマンドは Java ソース・ファイルのコンパイルも行います。

許可

なし

コマンド構文



コマンド・パラメーター

-help

変換プログラムがサポートする各オプションが、SQLJ 変換プログラムによって説明されることを指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

-dir=*directory*

SQLJ が変換プログラムによって生成される .java ファイル、およびコンパイラーによって生成される .class ファイルを入れるディレクトリーの名前を指定します。デフォルトは、SQLJ ソース・ファイルが入っているディレクトリーです。

変換プログラムは、生成ファイルをディレクトリーに配置するときに、SQLJ ソース・ファイルのディレクトリー構造を使用します。例えば、変換プログラムが次の 2 つのファイルを処理するとします。

- Java パッケージにない file1.sqlj
- Java パッケージ sqlj.test にある file2.sqlj

また、変換プログラムを呼び出すときにパラメーター -dir=/src を指定するとします。変換プログラムは file1.sqlj 用の Java ソース・ファイルをディレクトリー /src に入れ、file2.sqlj 用の Java ソース・ファイルをディレクトリー /src/sqlj/test に入れます。

-d=*directory*

SQLJ が変換プログラムおよびコンパイラーによって生成されるバイナリー・ファイルを入れるディレクトリーの名前を指定します。これらのファイルには、.ser ファイル、*name*_SJProfileKeys.class ファイル、およびコンパイラーによって生成される .class ファイルが含まれます。

デフォルトは、SQLJ ソース・ファイルが入っているディレクトリーです。

変換プログラムは、生成ファイルをディレクトリーに配置するときに、SQLJ ソース・ファイルのディレクトリー構造を使用します。例えば、変換プログラムが次の 2 つのファイルを処理するとします。

- Java パッケージにない file1.sqlj
- Java パッケージ sqlj.test にある file2.sqlj

また、変換プログラムを呼び出すときにパラメーター -d=/src を指定するとします。変換プログラムは file1.sqlj 用のシリアライズド・プロファイルをディレクトリー /src に入れ、file2.sqlj 用のシリアライズド・プロファイルをディレクトリー /src/sqlj/test に入れます。

-compile=true|false

SQLJ 変換プログラムが生成済みの Java ソースをバイトコードにコンパイルするかどうかを指定します。

true
変換プログラムは生成済みの Java ソース・コードをコンパイルします。これはデフォルトです。

false
変換プログラムは生成済みの Java ソース・コードをコンパイルしません。

-linemap=no|yes

Java 例外の行番号が、SQLJ ソース・ファイル (.sqlj ファイル) の行番号に一致するか、SQLJ 変換プログラムによって生成される Java ソース・ファイル (.java ファイル) の行番号に一致するかを指定します。

no Java 例外の行番号は Java ソース・ファイルの行番号に一致します。これはデフォルトです。

yes
Java 例外の行番号は SQLJ ソース・ファイルの行番号に一致します。

-smap=no|yes
SQLJ 変換プログラムが SQLJ ソース・ファイルごとにソース・マップ (SMAP) ファイルを生成するかどうかを指定します。 SMAP ファイルはいくつかの Java 言語デバッグ・ツールで使用されます。このファイルは SQLJ ソース・ファイルの行を、SQLJ 変換プログラムによって生成される Java ソース・ファイルの行にマップします。ファイルは Unicode UTF-8 コード化スキームです。その形式は、以下の Web サイトの Original Java Specification Request (JSR) 45 に説明されています。

```
http://www.jcp.org
```

no SMAP ファイルを生成しません。これはデフォルトです。

yes
SMAP ファイルを生成します。SMAP ファイルの名前は *SQLJ-source-file-name*.java.smap です。 SQLJ 変換プログラムは SMAP ファイルを、生成された Java ソース・ファイルと同じディレクトリーに置きます。

-encoding=*encoding-name*
ソース・ファイルのエンコードを指定します。例として JIS または EUC があります。このオプションを指定しない場合、オペレーティング・システムのデフォルト・コンバーターが使用されます。

-db2optimize
SQLJ 変換プログラムが DB2 に合わせて最適化された接続コンテキスト・クラスのコードを生成することを指定します。 -db2optimize は、デフォルト・コンテキストのコードではなく、ユーザー定義コンテキストのコードを最適化します。

SQLJ 変換プログラムを -db2optimize オプションを指定して実行する際に、アプリケーションが JDBC 3.0 またはそれ以前の関数を使用する場合、生成された Java アプリケーションをコンパイルするには IBM Data Server Driver for JDBC and SQLJ ファイル db2jcc.jar が CLASSPATH に入っていなければなりません。アプリケーションが JDBC 4.0 またはそれ以前の関数を使用する場合、生成された Java アプリケーションをコンパイルするには IBM Data Server Driver for JDBC and SQLJ ファイル db2jcc4.jar が CLASSPATH に入っていなければなりません。

-ser2class
SQLJ 変換プログラムが .ser ファイルを .class ファイルに変換することを指定します。

-status
SQLJ 変換プログラムが実行中に状況メッセージを表示することを指定します。

-version
SQLJ 変換プログラムが IBM Data Server Driver for JDBC and SQLJ のバージョンを表示することを指定します。情報の形式は次のとおりです。

```
IBM SQLJ xxxx.xxxx.xx
```

-C-help
SQLJ 変換プログラムが Java コンパイラーのヘルプ情報を表示することを指定します。

-C*compiler-option*
ダッシュ (-) で始まる有効な Java コンパイラー・オプションを指定します。-C とコンパイラー・オプションの間にはスペースを入れないでください。複数のコンパイラー・オプションを指定する必要がある場合、各コンパイラー・オプションの前に -C を付けてください。以下に例を示します。

```
-C-g -C-verbose
```

すべてのオプションは Java コンパイラーに渡され、SQLJ 変換プログラムには使用されません。ただし、以下のオプションを**除きます**。

-classpath
SQLJ 変換プログラムおよび Java コンパイラーによって使用されるユーザー・クラス・パスを指定します。この値は CLASSPATH 環境変数をオーバーライドします。

-sourcepath
SQLJ 変換プログラムおよび Java コンパイラーがクラスまたはインターフェース定義を検索するソース・コード・パスを指定します。 SQLJ 変換プログラムは、JAR または zip ファイル内ではなく、ディレクトリー内の .sqlj および .java ファイルのみ検索します。

-J*JVM-option*
sqlj コマンドの実行場所である Java 仮想マシン (JVM) に渡されるオプションを指定します。オプションは、ダッシュ (-) で始まる有効な JVM オプションでなければなりません。 -J と JVM オプションの間にはスペースを入れないでください。複数の JVM オプションを指定する必要がある場合、各コンパイラー・オプションの前に -J を付けてください。以下に例を示します。

```
-J-Xmx128m -J-Xmine2M
```

SQLJ-source-file-name
変換される SQLJ ソース・ファイルのリストを指定します。これは必須パラメーターです。すべての SQLJ ソース・ファイル名には拡張子 .sqlj が付いていなければなりません。

出力

各ソース・ファイル *program-name*.sqlj ごとに、SQLJ 変換プログラムは以下のファイルを作成します。

- 生成済みソース・プログラム

  生成済みソース・ファイルには *program-name*.java という名前が付けられます。
- SQLJ 実行可能節で使用される接続コンテキスト・クラスごとのシリアライズド・プロファイル・ファイル

  直列化されたプロファイル名の形式は、以下のようになります。

  *program-name*_SJProfile*IDNumber*.ser
- SQLJ 変換プログラムが Java コンパイラーを呼び出す場合、コンパイラーが生成するクラス・ファイル

例

```
sqlj -encoding=UTF8 -C-O MyApp.sqlj
```

db2sqljcustomize - SQLJ プロファイル・カスタマイザー

db2sqljcustomize は、組み込み SQL ステートメントを含む SQLJ プロファイルを処理します。

デフォルトでは、それぞれの分離レベルごとに 1 つずつ、4 つの DB2 パッケージが db2sqljcustomize によって作成されます。db2sqljcustomize では、実行時に使用できるように、 DB2 の固有情報を使ってプロファイルが拡大されます。

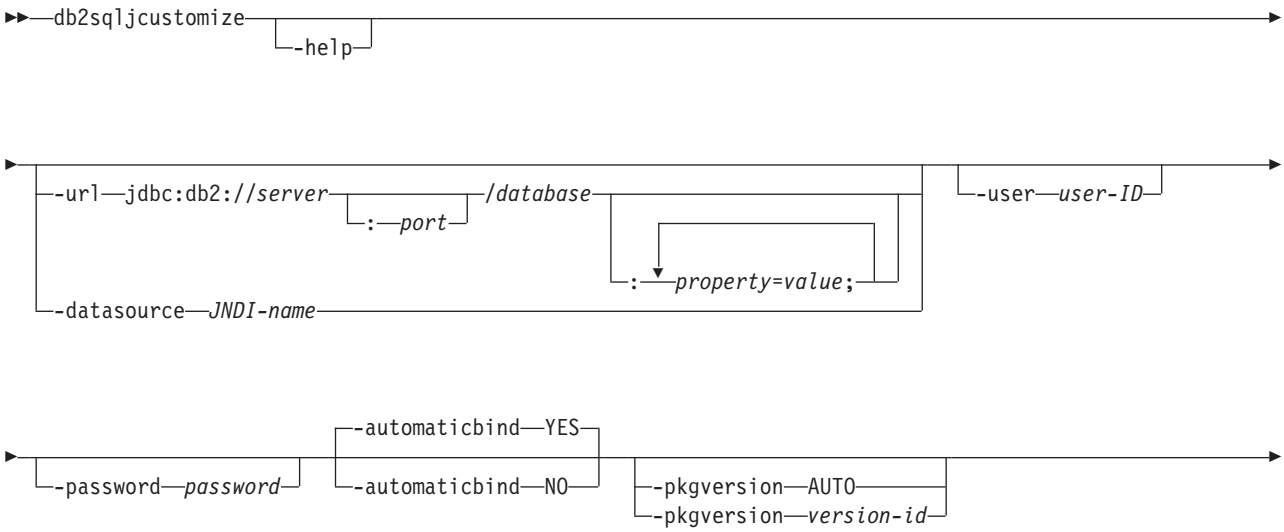
許可

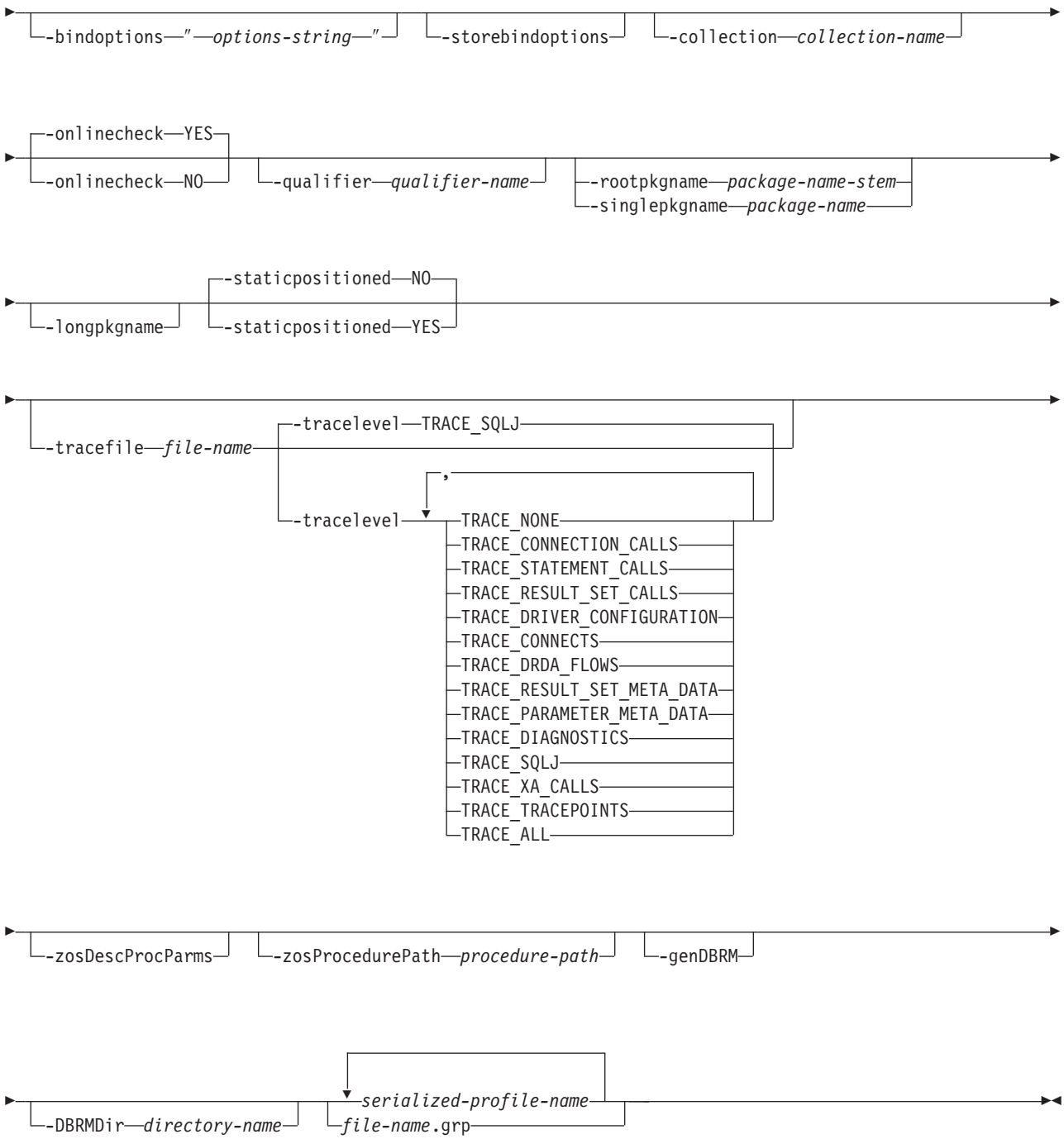
プロセスの特権セットには、以下の権限のうちの 1 つを組み込む必要があります。

- DBADM 権限
- パッケージが存在しない場合の BINDADD 特権と、以下のいずれかの特権
  - CREATEIN 特権
  - パッケージのスキーマ名が存在しない場合、データベースに対する IMPLICIT_SCHEMA 権限
- パッケージが存在する場合に必要な特権:
  - スキーマに対する ALTERIN 特権
  - パッケージに対する BIND 特権

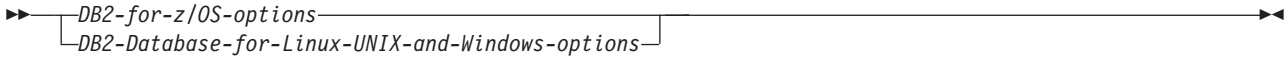
ユーザーには、アプリケーションで静的 SQL ステートメントをコンパイルするのに必要なすべての特権も必要です。グループに認可された特権が、静的ステートメントの許可の検査に使用されることはありません。

コマンド構文

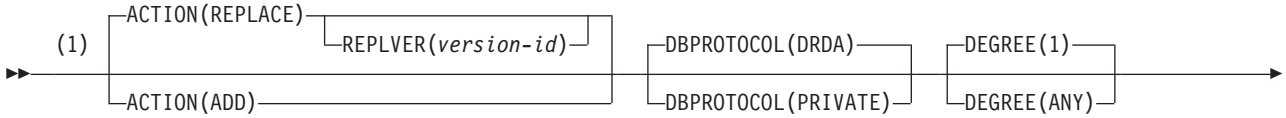


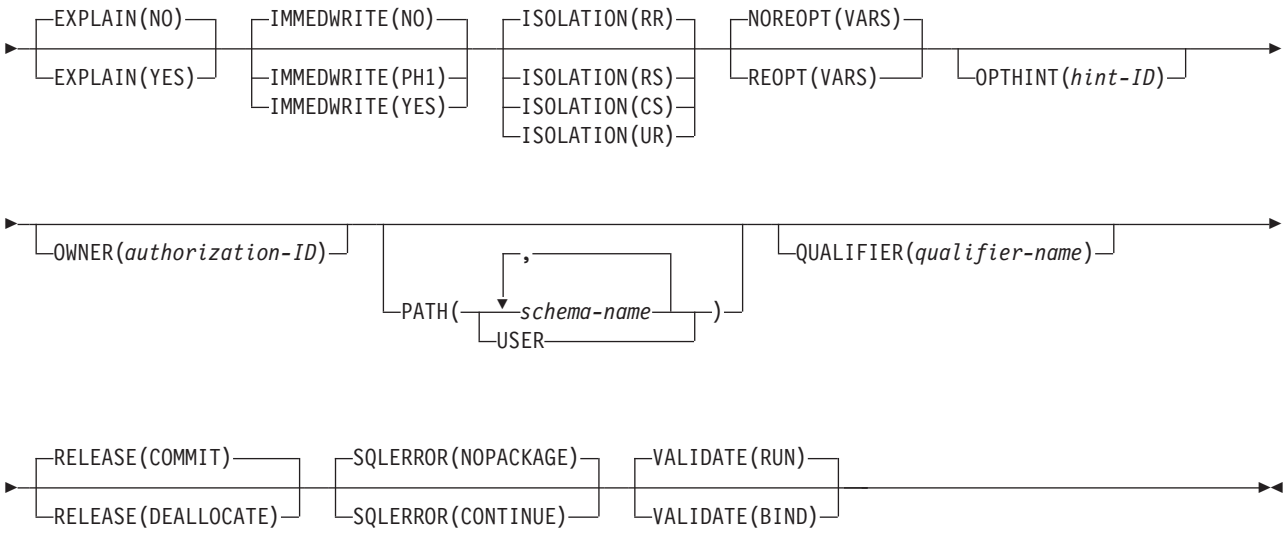


options-string:



DB2 for z/OS options:

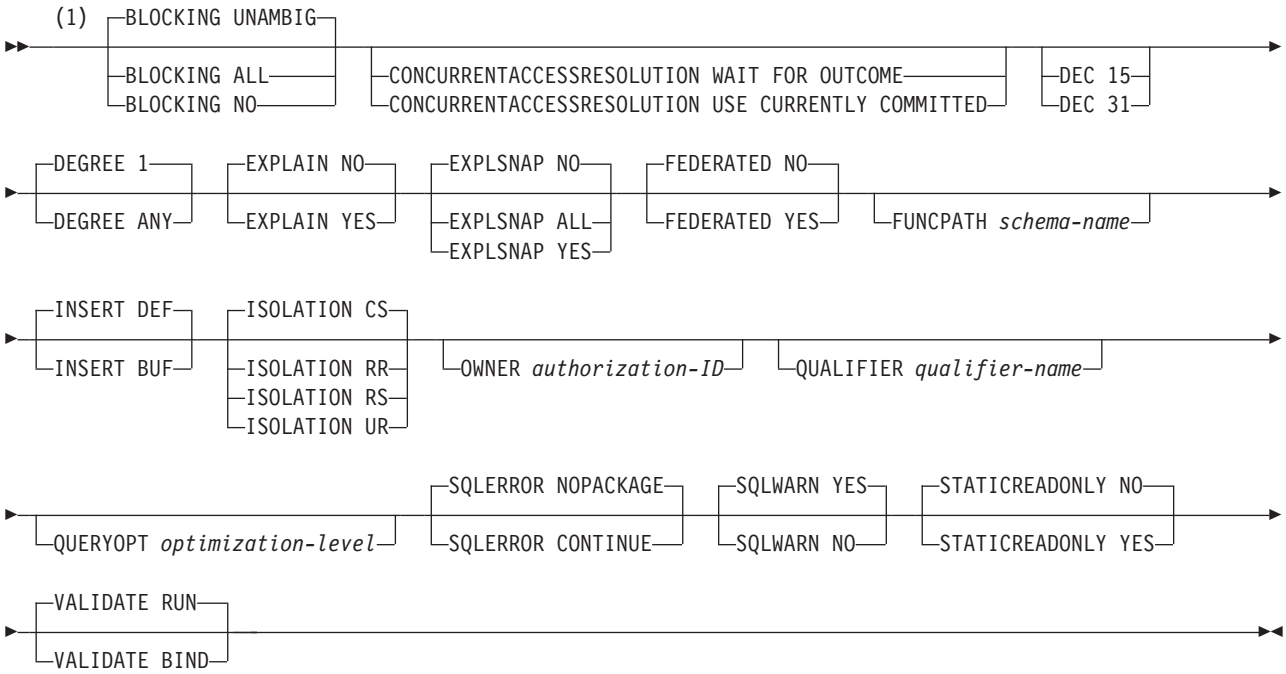




注:

1 これらのオプションは、どのような順序で指定してもかまいません。

DB2 Database for Linux, UNIX, and Windows options



注:

1 これらのオプションは、どのような順序で指定してもかまいません。

コマンド・パラメーター

-help

SQLJ カスタマイザーがサポートする各オプションが、このカスタマイザーによって説明されることを指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

-url

プロファイルがカスタマイズされるデータ・ソースの URL を指定します。-automaticbind オプションまたは -onlinecheck オプションが YES に指定されているか、デフォルトで YES になった場合、この URL が表すデータ・ソースへの接続が確立されます。`-url` 値の可変部は、以下のとおりです。

server

DB2 サブシステムが存在する z/OS システムのドメイン名または IP アドレス。

port

DB2 サブシステムに割り当てられている TCP/IP サーバーのポート番号。デフォルトは 446 です。

-url

プロファイルがカスタマイズされるデータ・ソースの URL を指定します。-automaticbind オプションまたは -onlinecheck オプションが YES に指定されているか、デフォルトで YES になった場合、この URL が表すデータ・ソースへの接続が確立されます。`-url` 値の可変部は、以下のとおりです。

server

データベース・サーバーが存在するオペレーティング・システムのドメイン名または IP アドレス。

port

データベース・サーバーに割り当てられている TCP/IP サーバーのポート番号。デフォルトは 446 です。

database

プロファイルがカスタマイズされるデータベース・サーバーの名前。

接続を DB2 for z/OS サーバーに対して行う場合、*database* は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

接続を DB2 Database for Linux, UNIX, and Windows サーバーに対して行う場合、*database* は、インストール中に定義されるデータベース名です。

接続を IBM Cloudscape サーバーに対して行う場合、*database* は、データベースを含むファイルの完全修飾名です。この名前は、二重引用符 (″) で囲まなければなりません。以下に例を示します。

```
"c:/databases/testdb"
```

property=*value*;
JDBC 接続のプロパティー。

property=*value*;
JDBC 接続のプロパティー。

-datasource *JNDI-name*
JNDI に登録された DataSource オブジェクトの論理名を指定します。DataSource オブジェクトは、プロファイルがカスタマイズされるデータ・ソースを表します。-automaticbind オプションまたは -onlinecheck オプションが YES に指定されているか、デフォルトで YES になった場合、このデータ・ソースへの接続が確立されます。-datasource は、-url を指定する代わりに指定できます。DataSource オブジェクトは、 IBM Data Server Driver for JDBC and SQLJ Type 4 接続 を使用する接続を表す必要があります。

-user *user-ID*
オンライン検査またはパッケージのバインドのためにデータ・ソースに接続するときに使うユーザー ID を指定します。 -url を指定する場合は、-user を指定しなければなりません。-datasource を指定し、*JNDI-name* が表す DataSource オブジェクトにユーザー ID が含まれない場合、-user を指定しなければなりません。

-password *password*
オンライン検査またはパッケージのバインドのためにデータ・ソースに接続するときに使うパスワードを指定します。 -url を指定する場合は、-password を指定しなければなりません。-datasource を指定し、*JNDI-name* が表す DataSource オブジェクトにパスワードが含まれない場合、-password を指定しなければなりません。

-automaticbind YES|NO
-url パラメーターで指定されているデータ・ソースにおいてカスタマイザーが DB2 パッケージをバインドするかどうかを指定します。

デフォルトは YES です。

パッケージ数と、それらのパッケージの分離レベルは、-rootpkgname および -singlepkgname オプションで制御します。

バインド操作が功を奏するには、以下の条件が満たされる必要があります。

- TCP/IP と DRDA がターゲットのデータ・ソースにインストールされていなければなりません。
- -url、-username、および -password にそれぞれ有効な値を指定する必要があります。
- -username の値には、ターゲット・データ・ソースにおいてパッケージをバインドする許可がなければなりません。

-pkgversion AUTO|*version-id*
カスタマイズしようとしているシリアライズド・プロファイル用のパッケージがサーバー側でバインドされるときに使用するパッケージ・バージョンを指定します。 db2sqljcustomize は、シリアライズド・プロファイル内と DB2 パッケージ内にバージョン ID を保管します。実行時のバージョン検査では、バージョン名ではなく、整合性トークンがベースになります。整合性トークンに基づいたバージョン名を自動的に生成するには、-pkgversion AUTO を指定します。

デフォルトでは、バージョンはありません。

-bindoptions *options-string*
スペースで区切ったオプションのリストを指定します。それらのオプションには、同名の DB2 プリコンパイル・オプションおよびバインド・オプションと同じ機能があります。 DB2 for z/OS システム上で実行されるプログラムを準備する場合、 DB2 for z/OS のオプションを指定します。 DB2 Database for Linux, UNIX, and Windows システム上で実行されるプログラムを準備する場合、 DB2 Database for Linux, UNIX, and Windows のオプションを指定します。

バインド・オプションに関する注意事項:

- ISOLATION を指定するのは、-singlepkgname オプションも指定する場合だけです。
- STATICREADONLY をサポートするサーバーの場合は、STATICREADONLY の値は YES、その他のサーバーの場合は、NO となります。 STATICREADONLY YES を指定すると、 DB2 は未確定カーソルを読み取り専用カーソルであるものとして処理します。イテレーター宣言のエラーのトラブルシューティングの場合、STATICREADONLY NO を明示的に指定する必要があります。あるいは、確定済みになるようにイテレーターを宣言します。例えば、イテレーターを確定的に更新可能にする場合は、sqlj.runtime.ForUpdate をインプリメントするようにイテレーターを宣言します。イテレーターを読み取り専用にする場合、イテレーターを使用する SELECT ステートメントに FOR READ ONLY 節を付け加えます。

重要: パッケージのバインディングを行う場所であるデータ・ソースに適したプログラム準備オプションのみを指定してください。 IBM Data Server Driver for JDBC and SQLJ の一部の値およびデフォルト値は、 DB2 用の値およびデフォルト値とは異なります。

-storebindoptions
-bindoptions および -staticpositioned パラメーターの値をシリアライズド・プロファイル内に保管することを指定します。 -bindoptions または -staticpositioned パラメーターを使用しないで db2sqljbind を呼び出すと、シリアライズド・プロファイルに保管されている値がバインド操作で使用されます。 1 回の db2sqljcustomize の呼び出しで複数のシリアライズド・プロファイルを指定すると、パラメーター値は、それぞれのシリアライズド・プロファイル内に保管されます。保管値は、db2sqljprint ユーティリティーからの出力に表示されます。

-collection *collection-name*
`db2sqljcustomize` がバインドするパッケージの修飾子。 `db2sqljcustomize` は、カスタマイズされたシリアライズド・プロファイル内にその値を保管します。その値は、関連パッケージのバインディング時に使用されます。このパラメーターを指定しないと、`db2sqljcustomize` では NULLID のコレクション ID が使用されます。

-onlinecheck YES|NO
SQLJ プログラム内のデータ・タイプのオンライン検査を実行するかどうかを指定します。 -url または -datasource オプションは、オンライン検査で使用され

るデータ・ソースを決定します。 -url または -datasource パラメーターを指定した場合のデフォルトは YES です。それ以外の場合のデフォルトは NO です。

-qualifier *qualifier-name*
オンライン検査で使用するための、SQLJ プログラム内の非修飾オブジェクトの修飾子を指定します。パッケージのバインディング時には、この値は修飾子として使用されません。

-rootpkgname|-singlepkgname
プログラムに関連したパッケージの名前を指定します。 -automaticbind が NO である場合、これらのパッケージ名が db2sqljbind 実行時に使用されます。各パラメーターの意味は、以下のとおりです。

-rootpkgname *package-name-stem*
カスタマイザーが、4 つの DB2 分離レベルのそれぞれに対して 1 つずつ、4 つのパッケージを作成することを指定します。 4 つのパッケージの名前は以下のとおりです。

*package-name-stem***1**
分離レベル UR 用

*package-name-stem***2**
分離レベル CS 用

*package-name-stem***3**
分離レベル RS 用

*package-name-stem***4**
分離レベル RR 用

-longpkgname が指定されていない場合、*package-name-stem* は 7 バイト以下の英数字ストリングでなければなりません。

-longpkgname が指定されている場合、*package-name-stem* は 127 バイト以下の英数字ストリングでなければなりません。

-singlepkgname *package-name*
カスタマイザーが *package-name* という名前の 1 つのパッケージを作成することを指定します。このオプションを指定した場合、プログラムはただ 1 つの分離レベルで実行されます。パッケージの分離レベルの指定は、-bindoptions オプション・ストリングの ISOLATION オプションで指定します。

-longpkgname が指定されていない場合、*package-name* は 8 バイト以下の英数字ストリングでなければなりません。

-longpkgname が指定されている場合、*package-name* は 128 バイト以下の英数字ストリングでなければなりません。

-singlepkgname オプションの使用は推奨されていません。

推奨: ターゲット・データ・ソースが DB2 for z/OS である場合は、*package-name-stem* または *package-name* の値には大文字を使用してください。特定の CCSID 値で定義された DB2 for z/OS システムでは、パッケージ名またはコレクション名内の小文字が許容されません。

-rootpkgname または -singlepkgname を指定しない場合、db2sqljcustomize は直列化されたプロファイル名に基づいた 4 つのパッケージ名を生成します。直列化されたプロファイル名の形式は、以下のようになります。

*program-name*_SJProfile*IDNumber*.ser

生成される 4 つのパッケージ名は、以下の形式になります。

*Bytes-from-program-name**IDNumber**PkgIsolation*

表 106生成されるパッケージ名の部分およびそれぞれの部分のバイト数を示します。

パッケージ名の最大長は、*maxlen* です。-longpkgname が指定されない場合、*maxlen* は 8 です。-longpkgname が指定される場合は、*maxlen* は 128 です。

表 106. db2sqljcustomize によって生成されるパッケージ名の部分

パッケージ名の部分	バイト数	値
Bytes-from-program-name	*m*=min(Length(*program-name*), *maxlen*–1–Length(*IDNumber*))	*program-name* の最初の *m* バイト。大文字。
IDNumber	Length(*IDNumber*)	*IDNumber*
PkgIsolation	1	1、2、3、または 4。この値は、パッケージのトランザクション分離レベルを表します。表 107を参照してください。

表 107は、db2sqljcustomize によって生成されるパッケージ名の *PkgIsolation* 部分の値を示します。

表 107. PkgIsolation 値および関連した分離レベル

***PkgNumber* 値**	**パッケージの分離レベル**
1	非コミット読み取り (UR)
2	カーソル固定 (CS)
3	読み取り固定 (RS)
4	反復可能読み取り (RR)

例: プロファイル名が ThisIsMyProg_SJProfile111.ser であるとします。db2sqljcustomize オプション -longpkgname が指定されていません。したがって、*Bytes-from-program-name* は、ThisIsMyProg の大文字に変換された最初の 4 バイト、THIS です。*IDNumber* は 111 です。4 つのパッケージ名は以下のとおりです。

```
THIS1111
THIS1112
THIS1113
THIS1114
```

例: プロファイル名が ThisIsMyProg_SJProfile111.ser であるとします。db2sqljcustomize オプション -longpkgname が指定されています。したがって、*Bytes-from-program-name* は、ThisIsMyProg が大文字に変換されたもの、つまり THISISMYPROG です。*IDNumber* は 111 です。4 つのパッケージ名は以下のとおりです。

```
THISISMYPROG1111
THISISMYPROG1112
THISISMYPROG1113
THISISMYPROG1114
```

例: プロファイル名が A_SJProfile0.ser であるとします。*Bytes-from-program-name* は A で、*IDNumber* は 0 です。したがって、4 つのパッケージ名は以下のとおりです。

```
A01
A02
A03
A04
```

db2sqljcustomize にパッケージ名を生成させることは、推奨されていません。生成されたパッケージ名に既存のパッケージの名前と同じものがある場合、db2sqljcustomize は既存のパッケージを上書きします。パッケージ名が必ず固有になるようにするためには、-rootpkgname を指定します。

-longpkgname

128 バイトまでの DB2 パッケージ名を db2sqljcustomize で生成できることを指定します。このオプションは、長いパッケージ名をサポートするサーバーでパッケージをバインドする場合にのみ使用してください。-singlepkgname または -rootpkgname を指定する場合、以下の条件では、-longpkgname も指定しなければなりません。

- -singlepkgname の引数が 8 バイトよりも長い。
- -rootpkgname の引数が 7 バイトよりも長い。

-staticpositioned NO|YES

イテレーターを使用する位置指定 UPDATE ステートメントと同じソース・ファイルに宣言されているイテレーターの場合、静的にバインドされたステートメントとしてその位置指定 UPDATE を実行するかどうかを指定します。デフォルトは NO です。NO は、動的に準備されるステートメントとして位置指定 UPDATE を実行することを意味します。

-zosDescProcParms

DB2 for z/OS は、ストアード・プロシージャー・パラメーターに対する DESCRIBE 操作の実行を指定します。

-zosDescProcParms は、 DB2 for z/OS データベース・サーバーで実行されるプログラムにのみ適用されます。

DESCRIBE 情報が入手可能な場合、SQLJ には INOUT および OUT パラメーターの長さおよび精度に関する情報が含まれるため、これらのパラメーターに必要なメモリー量のみが割り振られます。 DESCRIBE 情報が入手できることによって最も大きな影響を受けるのは、INOUT、LOB OUT、および decimal OUT 文字パラメーターのためのストレージの使用です。

-zosDescProcParms を指定すると、 DB2 データベース・サーバーは、指定された値または -zosProcedurePath のデフォルト値を使用して、DESCRIBE 情報が要求されるストアード・プロシージャーの非修飾名を解決します。

-zosProcedurePath *procedure-path*

SQLJ プログラムのオンライン・チェック時に DB2 for z/OS が非修飾ストアード・プロシージャー名の解決に使用する、スキーマ名のリストを指定します。

-zosProcedurePath は、 DB2 for z/OS データベース・サーバーで実行されるプログラムにのみ適用されます。

リストは、スキーマ名のコンマ区切りのリストを二重引用符で囲んだストリング値になります。 DB2 データベース・サーバーは、そのリストを非修飾ストアード・プロシージャー名を解決するための SQL パスに挿入します。 SQL パスは次のとおりです。

```
SYSIBM, SYSFUN, SYSPROC, procedure-path, qualifier-name, user-ID
```

qualifier-name は -qualifier パラメーターの値であり、*user-ID* は -user パラメーターの値です。

DB2 データベース・サーバーは、データベース・サーバーに存在するストアード・プロシージャーの名前と一致するスキーマ名が見つかるまで、SQL パス内のスキーマ名を左から右に確認します。一致するものがあると、 DB2 データベース・サーバーはそのストアード・プロシージャーのパラメーターに関する情報を DB2 カタログから取得します。 DB2 データベース・サーバーが一致するものを検出しなかった場合、SQLJ は DB2 カタログ情報を使用せずにパラメーター・データを設定します。

-zosProcedurePath を指定しない場合、 DB2 データベース・サーバーは次の SQL パスを使用します。

```
SYSIBM, SYSFUN, SYSPROC, qualifier-name, user-ID
```

-qualifier パラメーターを指定しない場合、SQL パスには *qualifier-name* は含まれません。

-genDBRM

db2sqljcustomize がデータベース要求モジュール (DBRM) を生成することを指定します。これらの DBRM は、 DB2 for z/OS プランおよびパッケージを作成するのに使用できます。

-genDBRM は、 DB2 for z/OS データベース・サーバーで実行されるプログラムにのみ適用されます。

-genDBRM および -automaticbind NO が指定されていると、db2sqljcustomize は DBRM を作成しますが、それらを DB2 パッケージにバインドすることはしません。 -genDBRM および -automaticbind YES が指定されていると、db2sqljcustomize は DBRM を作成し、それらを DB2 パッケージにバインドします。

各 DB2 分離レベルごとに 1 つの DBRM が作成されます。生成された DBRM ファイルの命名規則は、パッケージの命名規則と同じものです。例えば、-rootpkgname SQLJSA0 が指定されており、-genDBRM も指定されている場合、4 つの DBRM ファイルの名前は次のようになります。

- SQLJSA01
- SQLJSA02
- SQLJSA03
- SQLJSA04

-DBRMDir *directory-name*

-genDBRM を指定する場合、-DBRMDir は生成された DBRM ファイルを db2sqljcustomize が配置するローカル・ディレクトリーを指定します。デフォルトは現行ディレクトリーです。

-DBRMdir は、 DB2 for z/OS データベース・サーバーで実行されるプログラムにのみ適用されます。

-tracefile *file-name*

トレースを使用可能にし、トレース情報のための出力ファイルを示します。このオプションを指定できるのは、IBM ソフトウェア・サポートから指示された場合のみです。

-tracelevel

-tracefile を指定する場合、db2sqljcustomize の実行中に何をトレースするかを指示します。デフォルトは TRACE_SQLJ です。このオプションを指定できるのは、IBM ソフトウェア・サポートから指示された場合のみです。

serialized-profile-name|*file-name***.grp**

カスタマイズしようとしている 1 つ以上のシリアライズド・プロファイルの名前を指定します。指定されたシリアライズド・プロファイルは、CLASSPATH 環境変数で指定されたディレクトリーになければなりません。

直列化されたプロファイル名の形式は、以下のようになります。

*program-name*_SJProfile*IDNumber*.ser

シリアライズド・プロファイルの名前を指定するときは、.ser 拡張子を付けても付けなくてもかまいません。

program-name は、拡張子 .sqlj の付いていない SQLJ ソース・プログラムの名前です。*n* は、0 から *m-1* までの整数です。ただし、*m* は、SQLJ ソース・プログラムから SQLJ 変換プログラムが生成したシリアライズド・プロファイルの数です。

以下のいずれかの方法で、シリアライズド・プロファイルの名前を指定することができます。

- db2sqljcustomize コマンドで名前の一覧を表示します。複数のシリアライズド・プロファイル名は、スペースを使用して区切る必要があります。
- *file-name*.grp という名前のファイルに、シリアライズド・プロファイル名を 1 行に 1 つずつ指定し、db2sqljcustomize コマンドに *file-name*.grp を指定します。

複数のシリアライズド・プロファイル名を指定し、しかもデフォルト値の -automaticbind YES を指定または使用した場合、db2sqljcustomize によってそれらのプロファイルから 1 つの DB2 パッケージがバインドされます。db2sqljcustomize を使用して、複数のシリアライズド・プロファイルから 1 つの DB2 パッケージを作成するには、-rootpkgname または -singlepkgname オプションも一緒に指定する必要があります。

複数のシリアライズド・プロファイル名を指定し、さらに -automaticbind NO も指定した場合に、db2sqljbind の実行時にそのシリアライズド・プロファイルを 1 つの DB2 パッケージにバインドするには、シリアライズド・プロファイル名が同じ順序で並んだ同じリストを db2sqljcustomize および db2sqljbind に指定する必要があります。

出力

db2sqljcustomize を実行すると、カスタマイズされたシリアライズド・プロファイルが作成されます。また、automaticbind 値が YES の場合は、 DB2 パッケージも作成されます。

例

```
db2sqljcustomize -user richler -password mordecai
  -url jdbc:db2:/server:50000/sample -collection duddy
  -bindoptions "EXPLAIN YES" pgmname_SJProfile0.ser
```

使用上の注意

常にオンライン検査をお勧めします。 シリアライズド・プロファイルをカスタマイズする際には、オンライン検査を使用するよう強くお勧めします。オンライン検査によって、 DB2 ホスト変数のデータ・タイプおよび長さに関する情報が確認されます。これは、以下の項目の場合特に重要です。

- java.lang.String ホスト変数および CHAR 列を備えた述部

  他のホスト言語の文字変数とは違って、Java の String ホスト変数の宣言では、長さ属性は使用されません。文字ホスト変数が含まれる照会を正しく最適化するには、そのホスト変数の長さが DB2 で必要です。例えば、ある照会が述部をもっていて、そこで、String ホスト変数が CHAR 列と比較され、CHAR 列上で索引が定義されると仮定します。 DB2 がホスト変数の長さを判別できない場合、索引スキャンではなく表スペース・スキャンを行うことがあります。オンライン検査は、対応する文字カラムの長さを提示することによって、このような問題が起きないようにします。

- java.lang.String ホスト変数および GRAPHIC 列を備えた述部

  オンライン検査を行わないと、String ホスト変数を GRAPHIC 列と比較する述部が検出された場合、バインド・エラー (SQLCODE -134) が DB2 から発行されることがあります。

- リモート・サーバーでの SQLJ SELECT ステートメントの結果表内の列名:

  オンライン検査を行わないと、リモートの SELECT の結果表の列名をドライバーが判別できません。

複数のシリアライズド・プロファイルの一括カスタマイズ: 複数のシリアライズド・プロファイルをまとめてカスタマイズし、1 つの DB2 パッケージを作成することができます。そのように作成する場合に、-staticpostioned YES を指定すると、カーソルの宣言とは異なるソース・ファイル内に UPDATE または DELETE ステートメントがあっても、パッケージ内で*事前に* 宣言されているカーソルを参照する位置指定の UPDATE または DELETE ステートメントは静的に実行されます。複数のソース・ファイルでプログラムが構成されている場合に、-staticpositioned YES の動作が望ましい場合、パッケージ内で位置指定の UPDATE または DELETE ステートメントより前にカーソル宣言が出現するように、db2sqljcustomize コマンド内でプロファイルを配列する必要があります。そのためには、イテレーターを参照する位置指定の UPDATE または DELETE ステートメントを収めたプロファイルより*前*に、結果表をイテレーターに割り当てる SELECT ステートメントを収めたプロファイルをリストします。

データ・ソースでカスタマイズされたカスタマイズ済みシリアライズド・プロファイルの、別のデータ・ソースでの使用: db2sqljcustomize を実行して、ある 1 つのデータ・ソースで、SQLJ プログラム用にカスタマイズされたシリアライズド・プロファイルを作成してから、別のデータ・ソースでそのシリアライズド・プロファイルを使用することができます。それには、db2sqljcustomize を 1 回実行して作成したカスタマイズ済みシリアライズド・プロファイルに対して、db2sqljbind を複数回実行します。それらのデータ・ソースでプログラムを実行するときは、そのプログラムからアクセスする DB2 オブジェクトは、どちらのデータ・ソースでも同じでなければなりません。例えば、どのデータ・ソースの表も、同じコード化スキームおよび同じデータ・タイプの同じ列をもっていなければなりません。

-collection パラメーターの使用: db2sqljcustomize は、作成した各カスタマイズ済みシリアライズド・プロファイル内に DB2 コレクション名を保管します。 SQLJ プログラムが実行されると、ドライバーは、カスタマイズ済みシリアライズド・プロファイル内に保管されているコレクション名を使用して、実行するパッケージを検索します。カスタマイズ済みシリアライズド・プロファイルに保管されている名前は、-collection パラメーターの値によって判別されます。シリアライズド・プロファイルに保管できるコレクション ID は 1 つだけです。ただし、-bindoptions パラメーター内で COLLECTION オプションを指定すれば、1 つのシリアライズド・プロファイルを複数のパッケージ・コレクションにバインドすることができます。シリアライズド・プロファイルに指定されているもの以外のコレクション内にあるパッケージを実行するには、プログラム内で SET CURRENT PACKAGESET ステートメントを使用します。

VERSION パラメーターの使用: 1 つの SQLJ プログラムの複数バージョンのパッケージを 1 つのコレクションにバインドするときは、VERSION パラメーターを使用します。これを実行するのは、SQLJ ソース・プログラムを変更した後で、新旧バージョンのプログラムを実行することになった場合です。

2 つのバージョンのパッケージを保守するには、次のステップを行います。

1. ソース・プログラム内のコードを変更します。
2. 新規のシリアライズド・プロファイルを作成するように、ソース・プログラムを変換します。オリジナルのシリアライズド・プロファイルを上書きしないよう気を付けてください。
3. db2sqljcustomize を実行し、シリアライズド・プロファイルをカスタマイズしてから、オリジナル・パッケージと同じパッケージ名を付けて同じコレクション内に DB2 パッケージを作成します。それには、オリジナル・パッケージの作成時に使用したのと同じ -rootpkgname および -collection の値を、新規のパッケージのバインド時に使用します。新規のカスタマイズ済みシリアライズド・プロファイル内と新規のパッケージ内にバージョン ID を書き込むには、-bindoptions パラメーターに VERSION オプションを指定します。

   このステップを実行するときは、VERSION オプションを必ず指定してください。指定しないと、オリジナル・パッケージが上書きされてしまいます。

旧バージョンのプログラムを実行すると、旧バージョンのパッケージが DB2 でロードされます。新バージョンのプログラムを実行すると、新バージョンのパッケージが DB2 でロードされます。

DB2 for z/OS でのパッケージおよびプランのバインディング: db2sqljcustomize -genDBRM パラメーターを使用してローカル・システムに DBRM を作成することができます。続いて、それらの DBRM を DB2 for z/OS システムに転送して、それらをパッケージまたはプランにバインドすることができます。この手法を使用する場合、DBRM ファイルを**バイナリー**・ファイルとして z/OS システムに、つまりレコード形式 FB およびレコード長 80 のパーティション・データ・セットに転送する必要があります。パッケージまたはプランをバインドするときには、以下のバインド・オプションの値を指定する必要があります。

ENCODING(EBCDIC)
DB2 for z/OS 上の IBM Data Server Driver for JDBC and SQLJ では、パッケージおよびプランには EBCDIC エンコードが必要になります。

DYNAMICRULES(BIND)
このオプションにより、SQLJ が動的 SQL を使用するときの許可規則が一貫性のあるものになります。 SQLJ は、複数の SQLJ プログラムが関係する位置指定 UPDATE または DELETE 操作に対して動的 SQL を使用します。

DBPROTOCOL(DRDA)
専用プロトコルは推奨されていません。DBPROTOCOL(DRDA) を全アプリケーションで使用してください。特に、3 部から構成されるリモート表名を使用する SQLJ アプリケーションの場合は、必ず DBPROTOCOL(DRDA) を使用してください。そうしないと、それらのアプリケーションは失敗する可能性があります。

db2sqljbind - SQLJ プロファイル・バインダー

db2sqljbind は、以前に db2sqljcustomize コマンドでカスタマイズされたシリアライズド・プロファイルの DB2 パッケージをバインドします。

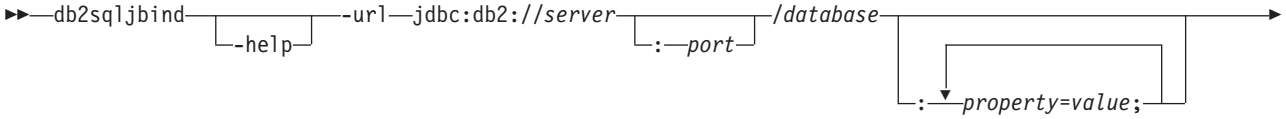
IBM Data Server Driver for JDBC and SQLJ と共に実行するアプリケーションは、パッケージを必要としますが、プランは必要としません。 db2sqljcustomize -automaticbind オプションが YES に指定されているか、デフォルトで YES になっている場合、db2sqljcustomize は -url パラメーターで指定されたデータ・ソースでパッケージのバインドを行います。しかし、-automaticbind が NO になっている場合、db2sqljcustomize の実行時にバインドが失敗した場合、あるいは、複数のロケーションで同じシリアライズド・プロファイルのための同一のパッケージを作成したい場合は、db2sqljbind コマンドを使用してパッケージをバインドできます。

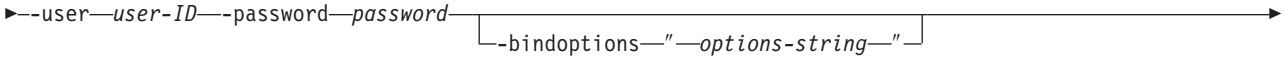
許可

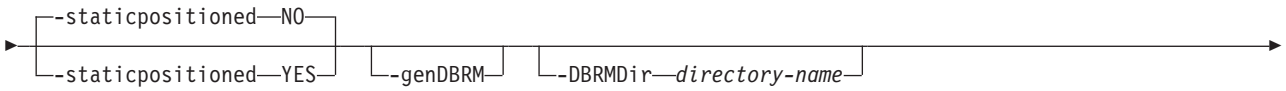
プロセスの特権セットには、以下の権限のうちの 1 つを組み込む必要があります。
- DBADM 権限
- パッケージが存在しない場合の BINDADD 特権と、以下のいずれかの特権
  - CREATEIN 特権
  - パッケージのスキーマ名が存在しない場合、データベースに対する IMPLICIT_SCHEMA 権限
- パッケージが存在する場合に必要な特権:
  - スキーマに対する ALTERIN 特権
  - パッケージに対する BIND 特権

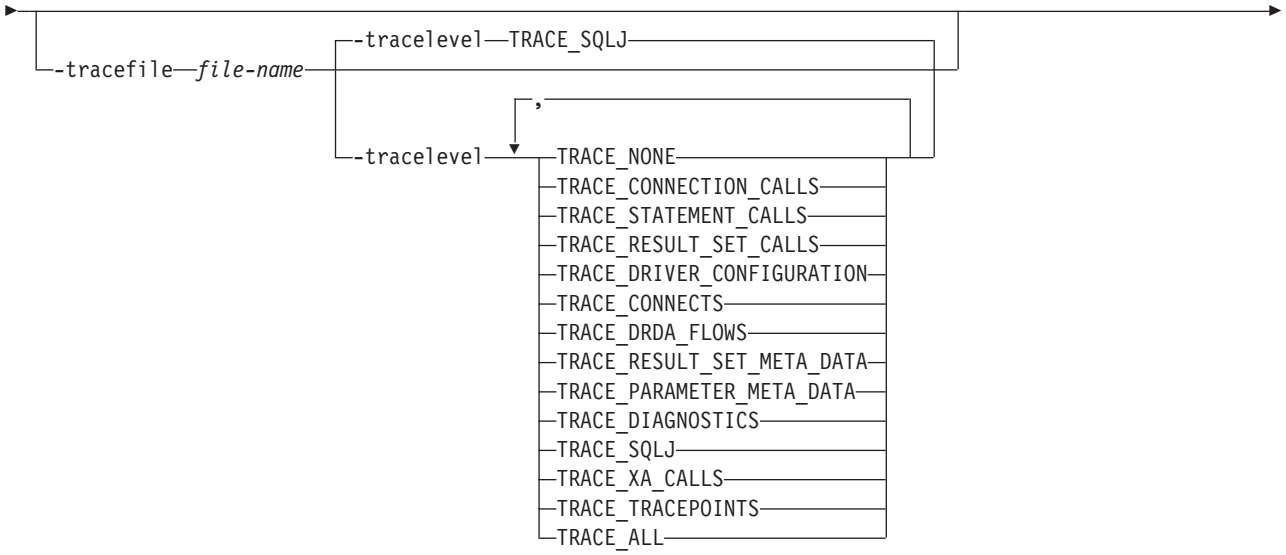
ユーザーには、アプリケーションで静的 SQL ステートメントをコンパイルするのに必要なすべての特権も必要です。グループに認可された特権が、静的ステートメントの許可の検査に使用されることはありません。

コマンド構文

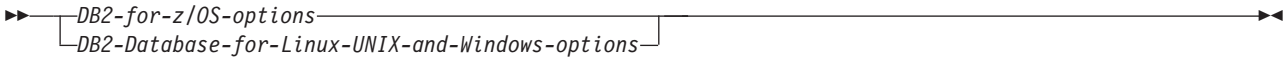




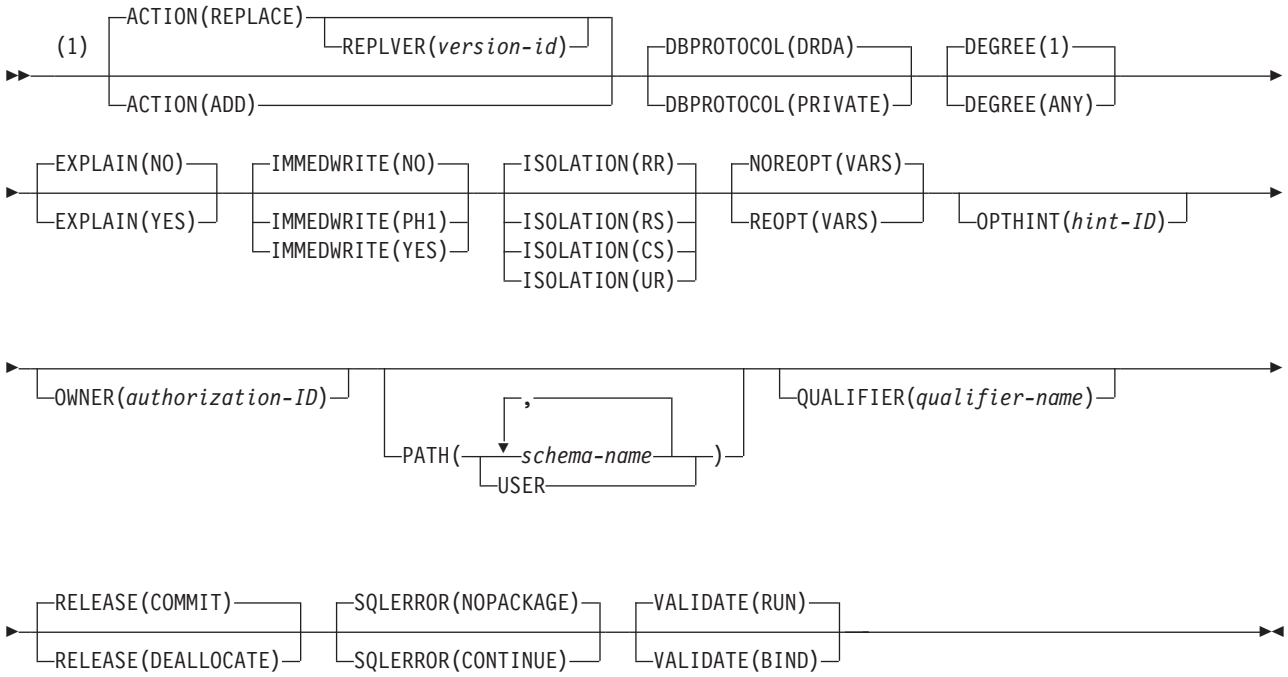




options-string:



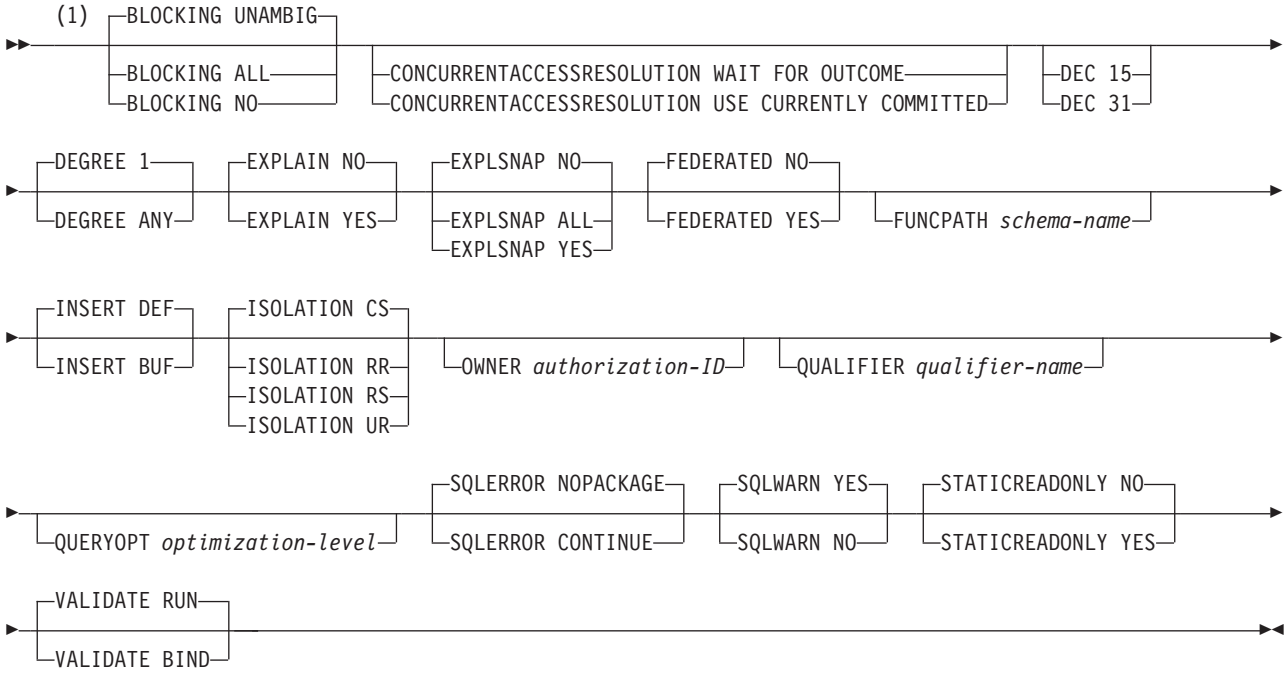
DB2 for z/OS options:



注:

1 これらのオプションは、どのような順序で指定してもかまいません。

DB2 Database for Linux, UNIX, and Windows options



注:

1 これらのオプションは、どのような順序で指定してもかまいません。

コマンド・パラメーター

-help
db2sqljbind はそれがサポートする各オプションについて記述することを指定します。 -help と共に他のオプションが指定されても、それらは無視されます。

-url
プロファイルがカスタマイズされるデータ・ソースの URL を指定します。-automaticbind オプションまたは -onlinecheck オプションが YES に指定されているか、デフォルトで YES になった場合、この URL が表すデータ・ソースへの接続が確立されます。-url 値の可変部は、以下のとおりです。

server
データベース・サーバーが存在するオペレーティング・システムのドメイン名または IP アドレス。

port
データベース・サーバーに割り当てられている TCP/IP サーバーのポート番号。デフォルトは 446 です。

database
プロファイルがカスタマイズされるデータベース・サーバーの名前。

接続を DB2 for z/OS サーバーに対して行う場合、*database* は、インストール中に定義される DB2 ロケーション名です。この値の中の文字はすべて大文字でなければなりません。サーバーで以下の SQL ステートメントを実行することにより、ロケーション名を判別できます。

```
SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
```

接続を DB2 Database for Linux, UNIX, and Windows サーバーに対して行う場合、*database* は、インストール中に定義されるデータベース名です。

接続を IBM Cloudscape サーバーに対して行う場合、*database* は、データベースを含むファイルの完全修飾名です。この名前は、二重引用符 (″) で囲まなければなりません。以下に例を示します。

```
"c:/databases/testdb"
```

property=*value*;
JDBC 接続のプロパティー。

-user *user-ID*
パッケージのバインドのためにデータ・ソースに接続するときに使うユーザー ID を指定します。

-password *password*
パッケージのバインドのためにデータ・ソースに接続するときに使うパスワードを指定します。

-bindoptions *options-string*
スペースで区切ったオプションのリストを指定します。それらのオプションには、同名の DB2 プリコンパイル・オプションおよびバインド・オプションと同じ機能があります。 DB2 for z/OS システム上で実行されるプログラムを準備する場合、 DB2 for z/OS のオプションを指定します。 DB2 Database for

Linux, UNIX, and Windows システム上で実行されるプログラムを準備する場合、 DB2 Database for Linux, UNIX, and Windows のオプションを指定します。

バインド・オプションに関する注意事項:

- 以下の条件が該当する場合にのみ VERSION を指定してください。
  - パッケージを DB2 Database for Linux, UNIX, and Windows システムでバインドしている場合、システムがバージョン 8 以降である。
  - 関連したパッケージを新規の VERSION 値にバインドする前にプログラム上で変換プログラムを再実行する。
- STATICREADONLY をサポートするサーバーの場合は、STATICREADONLY の値は YES、その他のサーバーの場合は、NO となります。
  STATICREADONLY YES を指定すると、 DB2 は未確定カーソルを読み取り専用カーソルであるものとして処理します。イテレーター宣言のエラーのトラブルシューティングの場合、STATICREADONLY NO を明示的に指定する必要があります。あるいは、確定済みになるようにイテレーターを宣言します。例えば、イテレーターを確定的に更新可能にする場合は、sqlj.runtime.ForUpdate をインプリメントするようにイテレーターを宣言します。イテレーターを読み取り専用にする場合、イテレーターを使用する SELECT ステートメントに FOR READ ONLY 節を付け加えます。

重要: パッケージのバインディングを行う場所であるデータ・ソースに適したプログラム準備オプションのみを指定してください。 IBM Data Server Driver for JDBC and SQLJ の一部の値およびデフォルト値は、 DB2 用の値およびデフォルト値とは異なります。

-staticpositioned NO|YES
イテレーターを使用する位置指定 UPDATE ステートメントと同じソース・ファイルに宣言されているイテレーターの場合、静的にバインドされたステートメントとしてその位置指定 UPDATE を実行するかどうかを指定します。デフォルトは NO です。NO は、動的に準備されるステートメントとして位置指定 UPDATE を実行することを意味します。この値は、シリアライズド・プロファイルの以前の db2sqljcustomize 呼び出しのための -staticpositioned 値と同じでなければなりません。

-genDBRM
db2sqljbind がシリアライズド・プロファイルからデータベース要求モジュール (DBRM) を生成すること、および db2sqljbind がリモート・バインド操作を行わないことを指定します。

-genDBRM は、 DB2 for z/OS データベース・サーバーで実行されるプログラムにのみ適用されます。

-DBRMDir *directory-name*
-genDBRM を指定する場合、-DBRMDir は生成された DBRM ファイルを db2sqljbind が配置するローカル・ディレクトリーを指定します。デフォルトは現行ディレクトリーです。

-DBRMdir は、 DB2 for z/OS データベース・サーバーで実行されるプログラムにのみ適用されます。

-tracefile *file-name*

トレースを使用可能にし、トレース情報のための出力ファイルを示します。このオプションを指定できるのは、IBM ソフトウェア・サポートから指示された場合のみです。

-tracelevel

-tracefile を指定する場合、db2sqljcustomize の実行中に何をトレースするかを指示します。デフォルトは TRACE_SQLJ です。このオプションを指定できるのは、IBM ソフトウェア・サポートから指示された場合のみです。

serialized-profile-name

パッケージのバインド元である 1 つ以上のシリアライズド・プロファイルの名前を指定します。直列化されたプロファイル名の形式は、以下のようになります。

*program-name*_SJProfile*IDNumber*.ser

program-name は、拡張子 .sqlj の付いていない SQLJ ソース・プログラムの名前です。*n* は、0 から *m-1* までの整数です。ただし、*m* は、SQLJ ソース・プログラムから SQLJ 変換プログラムが生成したシリアライズド・プロファイルの数です。

単一の DB2 パッケージを複数のシリアライズド・プロファイルからバインドするために複数のシリアライズド・プロファイル名を指定する場合、db2sqljcustomize を実行したときに、同じシリアライズド・プロファイル名を同じ順序で指定している必要があります。

例

```
db2sqljbind -user richler -password mordecai
  -url jdbc:db2://server:50000/sample -bindoptions "EXPLAIN YES"
  pgmname_SJProfile0.ser
```

使用上の注意

db2sqljbind によって作成されるパッケージ名: db2sqljbind によって作成されるパッケージの名前は、db2sqljcustomize を実行したときに -rootpkgname または -singlepkgname パラメーターを使用して指定された名前です。 -rootpkgname または -singlepkgname を指定しなかった場合、パッケージ名は分離レベル文字を付加したプロファイル名の最初の 7 バイトです。

db2sqljbind の DYNAMICRULES 値: DYNAMICRULES バインド・オプションによって、 DB2 パッケージの実行時の属性数が決まります。この属性のうちの 2 つは、許可を検査するときに使用される許可 ID と、非修飾オブジェクト用に使用される修飾子です。 SQLJ プログラムで動的に実行される位置指定 UPDATE および DELETE ステートメントの許可を正しいものにするために、db2sqljbind は常に DB2 パッケージを DYNAMICRULES(BIND) オプションにバインドします。このオプションは変更できません。 DYNAMICRULES(BIND) オプションを指定すると、SET CURRENT SQLID ステートメントおよび SET CURRENT SCHEMA ステートメントは SQLJ プログラムに対して無効になります。なぜなら、これらのステートメントは BIND 以外の DYNAMICRULES 値を使ってバインドされている動的ステートメントにのみ影響を与えるからです。

DYNAMICRULES(BIND) を指定する場合、動的 SQL ステートメント内の非修飾の表、ビュー、索引、および別名はバインド・オプション QUALIFIER の値を使って暗黙的に修飾されます。QUALIFIER を指定しない場合、 DB2 はパッケージ所有者の許可 ID を暗黙的な修飾子として使用します。この動作がご使用のプログラムに適していない場合、以下の手法のいずれかを使用して正しい修飾子を設定できます。

- 位置指定 UPDATE および DELETE ステートメントを静的に強制実行します。位置指定 UPDATE または DELETE ステートメントのカーソル (イテレーター) が位置指定 UPDATE または DELETE ステートメントと同じパッケージ内にある場合、db2sqljcustomize または db2sqljbind の -staticpositioned YES オプションを使用してこの強制実行を行えます。
- 位置指定 UPDATE および位置指定 DELETE ステートメント内の DB2 表名を完全修飾します。

db2sqljprint - SQLJ プロファイル・プリンター

db2sqljprint は、プロファイルのカスタマイズ済みバージョンの内容をプレーン・テキストとして印刷します。

許可

なし

コマンド構文

コマンド・パラメーター

profilename

SQLJ プロファイル・ファイルの相対名または絶対名を指定します。 SQLJ ファイルが Java ソース・ファイルに変換されると、そこに含まれる SQL 操作に関する情報は、プロファイルと呼ばれる SQLJ が生成したリソース・ファイルに保管されます。プロファイルは、オリジナルの入力ファイルの名前の後に続く接尾部 _SJProfileN (N は整数) で識別されます。その拡張子は .ser です。プロファイル名を指定する際には、.ser 拡張子を付けても付けなくてもかまいません。

例

```
db2sqljprint pgmname_SJProfile0.ser
```

付録 A. DB2 技術情報の概説

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2 インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、DB2 ライブラリーについて説明しています。DB2 ライブラリーに関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。英語および翻訳された DB2 バージョン 9.7 のマニュアル (PDF 形式) は、 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: *DB2 インフォメーション・センター*は、PDF やハードコピー資料よりも頻繁に更新されます。

表 108. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
管理 API リファレンス	SC88-5883-00	入手可能	2009 年 8 月
管理ルーチンおよびビュー	SC88-5880-00	入手不可	2009 年 8 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SC88-5885-00	入手可能	2009 年 8 月
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SC88-5886-00	入手可能	2009 年 8 月
コマンド・リファレンス	SC88-5884-00	入手可能	2009 年 8 月
データ移動ユーティリティー ガイドおよびリファレンス	SC88-5903-00	入手可能	2009 年 8 月
データ・リカバリーと高可用性 ガイドおよびリファレンス	SC88-5904-00	入手可能	2009 年 8 月
データベース: 管理の概念および構成リファレンス	SC88-5870-00	入手可能	2009 年 8 月
データベースのモニタリング ガイドおよびリファレンス	SC88-5872-00	入手可能	2009 年 8 月
データベース・セキュリティー・ガイド	SC88-5905-00	入手可能	2009 年 8 月

表 108. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Text Search ガイド	SC88-5902-00	入手可能	2009 年 8 月
ADO.NET および OLE DB アプリケーションの開発	SC88-5874-00	入手可能	2009 年 8 月
組み込み SQL アプリケーションの開発	SC88-5875-00	入手可能	2009 年 8 月
Java アプリケーションの開発	SC88-5878-00	入手可能	2009 年 8 月
Perl、PHP、Python および Ruby on Rails アプリケーションの開発	SC88-5879-00	入手不可	2009 年 8 月
SQL および外部ルーチンの開発	SC88-5876-00	入手可能	2009 年 8 月
データベース・アプリケーション開発の基礎	GI88-4201-00	入手可能	2009 年 8 月
DB2 インストールおよび管理 概説 (Linux および Windows 版)	GI88-4202-00	入手可能	2009 年 8 月
グローバリゼーション・ガイド	SC88-5906-00	入手可能	2009 年 8 月
DB2 サーバー機能 インストール	GC88-5888-00	入手可能	2009 年 8 月
IBM データ・サーバー・クライアント機能 インストール	GC88-5889-00	入手不可	2009 年 8 月
メッセージ・リファレンス 第 1 巻	SC88-5897-00	入手不可	2009 年 8 月
メッセージ・リファレンス 第 2 巻	SC88-5898-00	入手不可	2009 年 8 月
Net Search Extender 管理およびユーザーズ・ガイド	SC88-5901-00	入手不可	2009 年 8 月
パーティションおよびクラスタリングのガイド	SC88-5907-00	入手可能	2009 年 8 月
pureXML ガイド	SC88-5895-00	入手可能	2009 年 8 月
Query Patroller 管理およびユーザーズ・ガイド	SC88-5908-00	入手不可	2009 年 8 月

表 108. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
Spatial Extender および Geodetic Data Management Feature ユーザーズ・ガイドおよびリファレンス	SC88-5900-00	入手不可	2009 年 8 月
SQL プロシージャー言語: アプリケーションのイネーブルメントおよびサポート	SC88-5877-00	入手可能	2009 年 8 月
SQL リファレンス 第 1 巻	SC88-5881-00	入手可能	2009 年 8 月
SQL リファレンス 第 2 巻	SC88-5882-00	入手可能	2009 年 8 月
問題判別およびデータベース・パフォーマンスのチューニング	SC88-5871-00	入手可能	2009 年 8 月
DB2 バージョン 9.7 へのアップグレード	SC88-5887-00	入手可能	2009 年 8 月
Visual Explain チュートリアル	SC88-5899-00	入手不可	2009 年 8 月
DB2 バージョン 9.7 の新機能	SC88-5893-00	入手可能	2009 年 8 月
ワークロード・マネージャー ガイドおよびリファレンス	SC88-5894-00	入手可能	2009 年 8 月
XQuery リファレンス	SC88-5896-00	入手不可	2009 年 8 月

表 109. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
DB2 Connect Personal Edition インストールおよび構成	SC88-5891-00	入手可能	2009 年 8 月
DB2 Connect サーバー機能 インストールおよび構成	SC88-5892-00	入手可能	2009 年 8 月
DB2 Connect ユーザーズ・ガイド	SC88-5890-00	入手可能	2009 年 8 月

表 110. Information Integration の技術情報

資料名	資料番号	印刷資料が入手可能かどうか	最終更新
Information Integration: Administration Guide for Federated Systems	SC19-1020-02	入手可能	2009 年 8 月
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-04	入手可能	2009 年 8 月
Information Integration: Configuration Guide for Federated Data Sources	SC19-1034-02	入手不可	2009 年 8 月
Information Integration: SQL Replication Guide and Reference	SC19-1030-02	入手可能	2009 年 8 月
Information Integration: Introduction to Replication and Event Publishing	GC19-1028-02	入手可能	2009 年 8 月

DB2 の印刷資料の注文方法

DB2 の印刷資料が必要な場合、オンラインで購入することができますが、すべての国および地域で購入できるわけではありません。 DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 *DB2 PDF ドキュメンテーション* DVD の一部のソフトコピー・ブックは、印刷資料では入手できないことに留意してください。例えば、「*DB2 メッセージ・リファレンス*」はどちらの巻も印刷資料としては入手できません。

DB2 PDF ドキュメンテーション DVD で利用できる DB2 の印刷資料の大半は、IBM に有償で注文することができます。国または地域によっては、資料を IBM Publications Center からオンラインで注文することもできます。お客様の国または地域でオンライン注文が利用できない場合、DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 DB2 PDF ドキュメンテーション DVD に収録されている資料の中には、印刷資料として提供されていないものもあります。

注: 最新で完全な DB2 資料は、DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v9r7) で参照することができます。

DB2 の印刷資料は以下の方法で注文することができます。

- 日本 IBM 発行のマニュアルはインターネット経由でご購入いただけます。詳しくは http://www.ibm.com/shop/publications/order をご覧ください。資料の注文情報にアクセスするには、お客様の国、地域、または言語を選択してください。その後、各ロケーションにおける注文についての指示に従ってください。
- DB2 の印刷資料を IBM 営業担当員に注文するには、以下のようにします。

1. 以下の Web サイトのいずれかから、営業担当員の連絡先情報を見つけてください。
   - IBM Directory of world wide contacts (www.ibm.com/planetwide)
   - IBM Publications Web サイト (http://www.ibm.com/shop/publications/order)。国、地域、または言語を選択し、お客様の所在地に該当する Publications ホーム・ページにアクセスしてください。このページから、「このサイトについて」のリンクにアクセスしてください。
2. 電話をご利用の場合は、DB2 資料の注文であることをご指定ください。
3. 担当者に、注文する資料のタイトルと資料番号をお伝えください。タイトルと資料番号は、552 ページの『DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)』でご確認いただけます。

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 製品は、SQL ステートメントの結果の原因になったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

SQL 状態ヘルプを開始するには、コマンド行プロセッサーを開いて以下のように入力します。

```
? sqlstate or ? class code
```

ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。
例えば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

DB2 バージョン 9.7 のトピックを扱っている DB2 インフォメーション・センターの URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/です。

DB2 バージョン 9.5 のトピックを扱っている DB2 インフォメーション・センターの URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/です。

DB2 バージョン 9 のトピックを扱っている DB2 インフォメーション・センターの URL は http://publib.boulder.ibm.com/infocenter/db2luw/v9/です。

DB2 バージョン 8 のトピックについては、バージョン 8 のインフォメーション・センターの URL http://publib.boulder.ibm.com/infocenter/db2luw/v8/にアクセスしてください。

DB2 インフォメーション・センターでの希望する言語でのトピックの表示

DB2 インフォメーション・センターでは、ブラウザーの設定で指定した言語でのトピックの表示が試みられます。トピックがその指定言語に翻訳されていない場合は、 DB2 インフォメーション・センターでは英語でトピックが表示されます。

- Internet Explorer Web ブラウザーで、指定どおりの言語でトピックを表示するには、以下のようにします。
  1. Internet Explorer の「**ツール**」 -> 「**インターネット オプション**」 -> 「**言語...**」ボタンをクリックします。「言語の優先順位」ウィンドウがオープンします。
  2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
     - リストに新しい言語を追加するには、「**追加...**」ボタンをクリックします。

       注: 言語を追加しても、特定の言語でトピックを表示するのに必要なフォントがコンピューターに備えられているとはかぎりません。
     - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで「**上に移動**」ボタンをクリックします。
  3. ブラウザー・キャッシュを消去してから、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。
- Firefox または Mozilla Web ブラウザーの場合に、希望する言語でトピックを表示するには、以下のようにします。
  1. 「**ツール**」 -> 「**オプション**」 -> 「**詳細**」 ダイアログの「**言語**」セクションにあるボタンを選択します。「設定」ウィンドウに「言語」パネルが表示されます。
  2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
     - リストに新しい言語を追加するには、「**追加...**」ボタンをクリックしてから、「言語を追加」ウィンドウで言語を選択します。
     - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで「**上に移動**」ボタンをクリックします。
  3. ブラウザー・キャッシュを消去してから、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。

ブラウザーとオペレーティング・システムの組み合わせによっては、オペレーティング・システムの地域の設定も希望のロケールと言語に変更しなければなりません。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

ローカルにインストールされた DB2 インフォメーション・センターは、定期的に更新する必要があります。

始める前に

DB2 バージョン 9.7 インフォメーション・センターが既にインストールされている必要があります。詳しくは、「*DB2 サーバー機能 インストール*」の『DB2 セットアップ・ウィザードによる DB2 インフォメーション・センターのインストール』

のトピックを参照してください。インフォメーション・センターのインストールに適用されるすべての前提条件と制約事項は、インフォメーション・センターの更新にも適用されます。

このタスクについて

既存の DB2 インフォメーション・センターは、自動で更新することも。手動で更新することもできます。

- 自動更新 - 既存のインフォメーション・センターのフィーチャーと言語を更新します。自動更新を使用すると、更新中にインフォメーション・センターが使用できなくなる時間が最小限で済むというメリットもあります。さらに、自動更新は、定期的に実行する他のバッチ・ジョブの一部として実行されるように設定することができます。
- 手動更新 - 更新処理中にフィーチャーまたは言語を追加する場合に使用する必要があります。例えば、ローカルのインフォメーション・センターが最初は英語とフランス語でインストールされており、その後ドイツ語もインストールすることにした場合、手動更新でドイツ語をインストールし、同時に、既存のインフォメーション・センターのフィーチャーおよび言語を更新できます。しかし、手動更新ではインフォメーション・センターを手動で停止、更新、再始動する必要があります。更新処理の間はずっと、インフォメーション・センターは使用できなくなります。

手順

このトピックでは、自動更新のプロセスを詳しく説明しています。手動更新の手順については、『コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新』のトピックを参照してください。

コンピューターまたはイントラネット・サーバーにインストールされている DB2 インフォメーション・センターを自動で更新するには、次のようにします。

1. Linux オペレーティング・システムの場合、次のようにします。
   a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、/opt/ibm/db2ic/V9.7 ディレクトリーにインストールされています。
   b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
   c. 次のように ic-update スクリプトを実行します。

      ```
      ic-update
      ```
2. Windows オペレーティング・システムの場合、次のようにします。
   a. コマンド・ウィンドウを開きます。
   b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、<Program Files>¥IBM¥DB2 Information Center¥Version 9.7 ディレクトリーにインストールされています (<Program Files> は「Program Files」ディレクトリーのロケーション)。

c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。

d. 次のように ic-update.bat ファイルを実行します。

```
ic-update.bat
```

結果

DB2 インフォメーション・センターが自動的に再始動します。更新が入手可能な場合、インフォメーション・センターに、更新された新しいトピックが表示されます。インフォメーション・センターの更新が入手可能でなかった場合、メッセージがログに追加されます。ログ・ファイルは、doc¥eclipse¥configuration ディレクトリーにあります。ログ・ファイル名はランダムに生成された名前です。例えば、1239053440785.log のようになります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの手動更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から資料の更新を入手してインストールすることができます。

ローカルにインストールされた DB2 インフォメーション・センターを手動で更新するには、以下のことを行う必要があります。

1. コンピューター上の DB2 インフォメーション・センターを停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新を適用できるようになります。DB2 インフォメーション・センターのワークステーション・バージョンは、常にスタンドアロン・モードで実行されます。を参照してください。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールしなければならない更新がある場合は、「更新」機能を使用してそれを入手およびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに DB2 インフォメーション・センターの更新をインストールする必要がある場合、インターネットに接続されていて DB2 インフォメーション・センターがインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングしてください。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージを入手します。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の DB2 インフォメーション・センターを再開します。

注: Windows 2008、Windows Vista (およびそれ以上) では、このセクションの後の部分でリストされているコマンドは管理者として実行する必要があります。完全な

管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを開くには、ショートカットを右クリックしてから、 「**管理者として実行**」を選択します。

コンピューターまたはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターを更新するには、以下のようにします。

1. DB2 インフォメーション・センターを停止します。
   - Windows では、「**スタート**」 → 「**コントロール パネル**」 → 「**管理ツール**」 → 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**停止**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv97 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。
     b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、<Program Files>¥IBM¥DB2 Information Center¥Version 9.7 ディレクトリーにインストールされています (<Program Files> は「Program Files」ディレクトリーのロケーション)。
     c. インストール・ディレクトリーから doc¥bin ディレクトリーにナビゲートします。
     d. 次のように help_start.bat ファイルを実行します。

        ```
        help_start.bat
        ```
   - Linux の場合:
     a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは、/opt/ibm/db2ic/V9.7 ディレクトリーにインストールされています。
     b. インストール・ディレクトリーから doc/bin ディレクトリーにナビゲートします。
     c. 次のように help_start スクリプトを実行します。

        ```
        help_start
        ```

   システムのデフォルト Web ブラウザーが開き、スタンドアロンのインフォメーション・センターが表示されます。
3. 「**更新**」ボタン ( ) をクリックします。(ブラウザーで JavaScript™ が有効になっている必要があります。) インフォメーション・センターの右側のパネルで、「**更新の検索 (Find Updates)**」をクリックします。 既存の文書に対する更新のリストが表示されます。
4. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. インストール・プロセスが完了したら、「**完了**」をクリックします。
6. 次のようにして、スタンドアロンのインフォメーション・センターを停止します。

- Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

  ```
  help_end.bat
  ```

  注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。help_start.bat は、Ctrl-C や他の方法を使用して停止しないでください。
- Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

  ```
  help_end
  ```

  注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に停止するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを停止しないでください。

7. DB2 インフォメーション・センターを再開します。
   - Windows では、「**スタート**」 → 「**コントロール パネル**」 → 「**管理ツール**」 → 「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv97 start
     ```

更新された DB2 インフォメーション・センターに、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 製品のさまざまな機能について学習するのを支援します。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド*」の『**pureXML™**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

「*Visual Explain チュートリアル*」の『**Visual Explain**』
: Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2 データベース製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 ドキュメンテーション

トラブルシューティング情報は、「*DB2 問題判別ガイド*」、または*DB2 インフォメーション・センター*の『データベースの基本』セクションにあります。ここには、DB2 診断ツールおよびユーティリティーを使用して、問題を切り分けて識別する方法、最も頻繁に起こる幾つかの問題に対するソリューションについての情報、および DB2 データベース製品を使用する際に発生する可能性のある問題の解決方法についての他のアドバイスがあります。

DB2 Technical Support の Web サイト

現在問題が発生していて、考えられる原因とソリューションを検索したい場合は、DB2 Technical Support の Web サイトを参照してください。Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

DB2 Technical Support の Web サイト (http://www.ibm.com/software/data/db2/support/db2_9/) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

付録 B. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。IBM 以外の製品に関する情報は、本書の最初の発行時点で入手可能な情報に基づいており、変更される場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを

経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。サンプル・プログラムは、現存するままの状態で提供されるものであり、いかなる種類の保証も提供されません。 IBM は、これらのサンプル・プログラムの使用から生ずるいかなる損害に対しても責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (*お客様の会社名*) (*西暦年*). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _*年を入れる*_. All rights reserved.

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtml の「Copyright and trademark information」をご覧ください。

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。
- UNIX は、The Open Group の米国およびその他の国における登録商標です。
- Intel®、Intel ロゴ、Intel Inside®、 Intel Inside ロゴ、Intel® Centrino®、Intel Centrino ロゴ、Celeron®、Intel® Xeon®、Intel SpeedStep®、Itanium®、Pentium® は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft®、 Windows、 Windows NT®、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

G

H

I

J

K

L

P

R

S

U

V

W

X

Printed in Japan

SC88-5878-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

IBM DB2 9.7 for Linux, UNIX, and Windows

Java アプリケーションの開発