Hewlett Packard Enterprise

SQL/MX リリース 3.6 用 NonStop JDBC タイプ 2 ドライバー プログラマ用リファレンス

部品番号: 691127-199
発行: 2018 年 3 月
版数: L18.02 およびそれ以降のすべての L シリーズの RVU

ご注意

本書の内容は、将来予告なしに変更されることがあります。Hewlett Packard Enterprise 製品およびサービスに対する保証については、当該製品およびサービスの保証規定書に記載されています。本書のいかなる内容も、新たな保証を追加するものではありません。本書の内容につきましては万全を期しておりますが、本書中の技術的あるいは校正上の誤り、脱落に対して、責任を負いかねますのでご了承ください。

本書で取り扱っているコンピューターソフトウェアは秘密情報であり、その保有、使用、または複製には、Hewlett Packard Enterprise から使用許諾を得る必要があります。米国政府の連邦調達規則である FAR 12.211 および 12.212 の規定に従って、コマーシャルコンピューターソフトウェア、コンピューターソフトウェアドキュメンテーションおよびコマーシャルアイテムのテクニカルデータ（Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items）は、ベンダーが提供する標準使用許諾規定に基づいて米国政府に使用許諾が付与されます。

他社の Web サイトへのリンクは、Hewlett Packard Enterprise の Web サイトの外に移動します。Hewlett Packard Enterprise は、Hewlett Packard Enterprise の Web サイト以外の情報を管理する権限を持たず、また責任を負いません。

商標

Microsoft®および Windows®は、米国および/またはその他の国における Microsoft Corporation の登録商標または商標です。

Intel®、インテル、Itanium®、Pentium®、Intel Inside®、および Intel Inside ロゴは、インテルコーポレーションまたはその子会社のアメリカ合衆国およびその他の国における商標または登録商標です。

Adobe®および Acrobat®は、米国 Adobe Systems Incorporated の登録商標です。

UNIX®は、The Open Group の登録商標です。

Java®および Oracle®は、Oracle および/またはその関連会社の登録商標です。

Open Software Foundation、OSF、OSF ロゴ、OSF/1、OSF/Motif、および Motif は、Open Software Foundation, Inc.の商標です。

保証

OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. This documentation and the software to which it relates are derived in part from materials supplied by the following:

© 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation. © 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991 Hewlett Packard Enterprise. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991, 1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informations system AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989, 1990, 1991 Transarc Corporation.

This software and documentation are based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. OSF acknowledges the following individuals and institutions for their role in its development: Kenneth C.R.C. Arnold, Gregory S. Couch, Conrad C. Huang, Ed

目次

このマニュアルについて

このマニュアルでは、HPE NonStop SQL/MX 用 HPE NonStop JDBC/MX ドライバー、タイプ 2 ドライバーの使用方法、HPE Integrity NonStop サーバーについて説明します。JDBC/MX ドライバーは、Java アプリケーション用 HPE NonStop サーバーに SQL/MX に対する JDBC アクセスを提供します。該当する場合、JDBC/MX ドライバー は、Oracle からの JDBC 3.0 API に準拠しています。

対象となるリリースバージョンアップデート（RVU）

本書は、改訂版で別途明示されるまで、L18.02 およびそれ以降のすべての L シリーズの RVU を対象とします。

対象読者

このマニュアルは、JDBC API を使用して、SQL NonStop/MX を持つ SQL データベースにアクセスする経験豊富な Java プログラマ用です。

読者が、NonStop Server for Java5 ― NonStop サーバー上でエンタープライズ Java アプリケーションにおいて使用するための Java インプリメンテーションに精通していることを仮定しています。NonStop Server for Java5 は、Oracle からヒューレット・パッカード エンタープライズにライセンスされる Solaris 用参照 Java インプリメンテーションに基づいています。NonStop Server for Java は、「NonStop Server for Java Programmer's Reference」 で説明されるように、Oracle JDK の準拠バージョンです。

読者が、フィールド内に文献を読み、JDBC API に精通していることを仮定してます。

参考資料

入門ガイド

SQL/MX Comparison Guide for SQL/MP Users

NonStop SQL/MP と NonStop SQL/MX の SQL の違いについて説明しています。

SQL/MX Quick Start

SQL/MX 会話型インターフェイス（MXCI）で SQL を使用するための基本的な方法について説明しています。サンプルデータベースのインストールについても説明しています。

リファレンスマニュアル

SQL/MX リファレンスマニュアル

SQL/MX 文、MXCI コマンド、関数、その他の SQL/MX 言語要素の構文について説明しています。

SQL/MX Messages Manual

SQL/MX のメッセージについて説明しています。

SQL/MX データベースサービスマニュアル

マルチテナント環境でのユーザーデータベースのプロビジョニングについて説明しています。mxdbs コマンドラインユーティリティを使用してユーザーデータベースを作成および管理する方法についても説明しています。

SQL/MX 3.5 SQL/MX 手続き型言語（PL/MX）リファレンスマニュアル

Oracle PL/SQL や ANSI SQL/PSM と多くの点で類似した NonStop SQL/MX の手続き型言語である、PL/MX について説明しています。これは NonStop SQL/MX でのユーザー定義ルーチン（UDR）の実装言語です。

MXDM User Guide for SQL/MX

HPE NonStop SQL/MX Database Manager を使用して SQL/MX データベースを監視および管理する方法について説明しています。

SQL/MX Workload Management Services (WMS) Reference Manual

SQL/MX 3.6 ワークロード管理サービス（WMS）の設定と構成、および NonStop システムのワークロードの監視方法について説明しています。

SQL/MX Glossary

SQL/MX の用語を定義しています。

インストールおよび移行ガイド

SQL/MX Installation and Upgrade Guide

SQL/MX データベースのインストールとアップグレードの計画方法について説明しています。

NonStop NS-Series Database Migration Guide

NonStop SQL/MX、NonStop SQL/MP、Enscribe のデータベースとアプリケーションを HPE Integrity NonStop NS シリーズのシステムに移行する方法について説明しています。

NonStop SQL/MP to SQL/MX Database and Application Migration Guide

データベースとアプリケーションを SQL/MP から SQL/MX に移行する方法について説明しています。

接続に関するマニュアル

SQL/MX 接続サービス マニュアル

Microsoft Open Database Connectivity (ODBC) アプリケーションプログラミングインターフェイス (API) およびその他の接続 API 用に開発されたアプリケーションで、NonStop SQL/MX を使用できるようにする、HPE NonStop SQL/MX 接続サービス (MXCS) をインストールおよび管理する方法について説明します。

SQL/MX 接続サービス管理コマンド リファレンス

SQL/MX 会話型インターフェイス (MXCI) と共に使用できる SQL/MX 管理コマンド ライブラリ (MACL) について説明します。

ODBC/MX Driver for Windows

Microsoft Windows 用 HPE NonStop ODBC/MX をインストールして構成する方法について説明します。これにより、ODBC API 向けに開発されたアプリケーションを有効化し、NonStop SQL/MX を使用できるようになります。

ODBC/MX Client Drivers User Guide for SQL/MX

ODBC/MX クライアント ドライバーをインストール、構成、使用する方法について説明します。

SQL/MX 用 JDBC タイプ 4 ドライバー プログラマ リファレンス

NonStop SQL/MX 用 JDBC タイプ 4 ドライバーを使用する方法について説明します。

データ管理ガイド

SQL/MX Management Manual

SQL/MX データベースの管理方法について説明しています。

SQL/MX Data Mining Guide

SQL/MX のデータ構造と、ナレッジ発見のプロセスを実行するための操作について説明しています。

SQL/MX Report Writer Guide

SQL/MX データベースからのデータを使用して書式設定されたレポートを作成する方法について説明しています。

DataLoader/MX Reference Manual

SQL/MX データベースをロードするためのツールである DataLoader/MX 製品の特徴と機能について説明しています。

アプリケーション開発ガイド

SQL/MX プログラミングマニュアル C および COBOL 言語用

SQL/MX 文を ANSI C および COBOL プログラムに埋め込む方法について説明しています。

SQL/MX Query Guide

クエリ実行プランを理解し、SQL/MX データベースに最適なクエリを記述する方法について説明しています。

SQL/MX Queuing and Publish/Subscribe Services

NonStop SQL/MX で、トランザクションキューイングおよびパブリッシュ/サブスクライブサービスをそのデータベースインフラストラクチャに統合するしくみについて説明しています。

SQL/MX Guide to Stored Procedures in Java

Java によって記述されるストアドプロシージャーを NonStop SQL/MX 内で使用する方法について説明しています。

オンラインヘルプ

SQL/MX Messages Online Help

SQL/MX Messages Manual の個々のメッセージをソース別に分類しています。

SQL/MX Glossary Online Help

SQL/MX Glossary の用語および定義で構成されています。

SQL/MX Database Manager Help

MXDM User Guide for SQL/MX のオンラインヘルプバージョンです。

MXCI Online Help

`SQL/MX Reference Manual` の SQL/MX 文と MXCI コマンドの構文について説明しています。

関連する SQL/MP マニュアル

これらのマニュアルは SQL/MP マニュアルライブラリの一部であり、重要なリファレンスです。SQL/MP データ定義言語（DDL）と、SQL/MP のインストールおよび管理について詳しくは、以下の SQL/MP マニュアルを参照してください。

SQL/MP Reference Manual

SQL/MP の言語要素、式、述語、関数、文について説明しています。

SQL/MP Installation and Management Guide

SQL/MP データベースを計画、インストール、作成、管理する方法について説明しています。インストールおよび管理コマンドと、SQL/MP のカタログおよびファイルについて説明しています。

新しい情報と変更された情報

691127-009 マニュアルにおける変更:

インストール要件における留意事項を更新しました。

発行履歴

本ガイドは、その置換出版物により指摘されるまで、NonStop SQL/MX リリース 3.6 をサポートします。出版日付および部品番号は、ドキュメントの現在の版を示します。

部品番号	製品バージョン	発行
691127-009	NonStop SQL/MX 3.6 用 JDBC/MX ドライバー	2018 年 3 月

JDBC/MX ドライバーの概要

HPE NonStop SQL/MX 用 HPE NonStop/MX JDBC ドライバーは、標準 JDBC 3.0 データ アクセス API に準拠している JDBC テクノロジーを実装します。JDBC/MX ドライバーにより、Java アプリケーションは、NonStop SQL/MX を使用して、NonStop SQL データベースにアクセスできます。

JDBC/MX インプリメンテーションに関連付けられている JDBC API に関する詳細情報については、Oracle ドキュメントを参照してください。標準 JDBC API に関する詳細情報を取得するには、Oracle によって提供される JDBC API のドキュメントをダウンロードしてください (**http://www.oracle.com/technetwork/java/download-141179.html**)。

HPE NonStop Server for Java を伴う JDBC/MX ドライバーは、エンタープライズ サーバー用のコンパクト、同時、動的、移植可能なプログラムをサポートしている Java 環境です。JDBC/MX ドライバーは、HPE NonStop オープンシステムサービス (OSS) 環境を必要とする NonStop Server for Java および SQL/MX を必要とします。NonStop Server for Java は、HPE NonStop オペレーティングシステムを使用して、Java 環境に NonStop システムの基礎である拡張性およびプログラム持続性を追加します。

本セクションでは、以下について説明します。

JDBC/MX アーキテクチャー

JDBC/MX ドライバーは、タイプ 2 ドライバーです。独自のネイティブ API を採用して、SQL/MX を使用し、NonStop SQL データベースにアクセスしています。SQL/MX のネイティブ API は、クライアントシステムからコールできません。このため、JDBC/MX ドライバーは、NonStop サーバーでのみ実行されます。

JDBC/MX ドライバーは、3 層モデルに最適です。3 層モデルでは 、コマンドは、データソースにコマンドを送信するサービスの中央層に送信されます。データソースは、コマンドを処理し、結果を中央層に送信し、その後、ユーザーに送信されます。中央層により、アクセス時に制御を維持し、企業データに対する一種の更新を行えます。別の利点は、アプリケーションの展開を簡単にすることです。最後に、多くの場合、3 層アーキテクチャーは、性能上の利点を提供できます。

次の図は、データベース アクセス用の 3 層アーキテクチャーを示します。

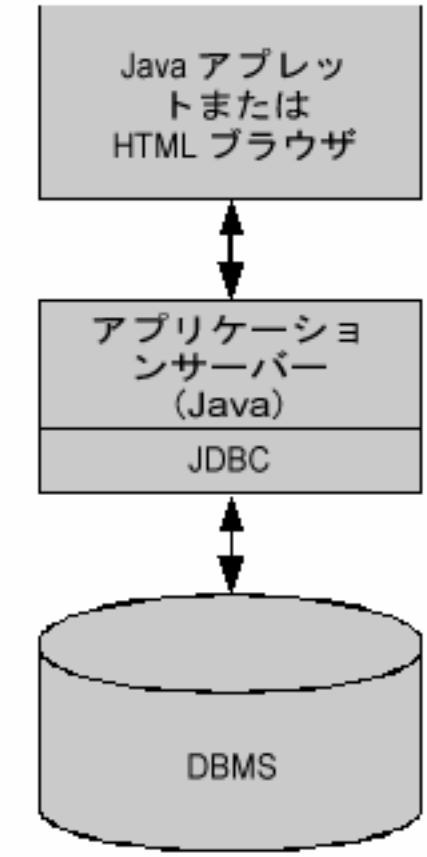

図 1: JDBC/MX ドライバーのアーキテクチャー

JDBC API パッケージ

JDBC/MX API パッケージは、JDBC/MX ドライバー ソフトウェアに付属しています。API ドキュメントについては、**HPE NonStop API リファレンス ページ**を参照してください。

`java.sql` および `javax.sql` パッケージは、Java 2 Standard Edition (J2SE) SDK 1.5 の一部として含まれています。したがって、NonStop Server for Java 5 製品と共に提供されるコア API と共に利用できます。

見本プログラムのサマリー

JDBC/MX ドライバー製品には、製品の機能を示すサンプル Java プログラムが含まれています。次の表で、プログラムについて説明します。

表 1: 見本プログラムのサマリー

見本プログラム	コメント
sampleJdbcMx.java	JDBC/MX ドライバーをロードし、DriverManager インターフェイスを使用して、JDBC 接続を取得する方法を示します。
CreateDataSource.java および TestDataSource.java	DataSource インターフェイスを使用して、接続し、Java プログラムにドライバー固有コードを回避する方法を示します。 CreateDataSource.java は、SQLMXDataSource オブジェクトを作成し、オブジェクトを Java 命名およびディレクトリインターフェイス (JNDI) に登録します。
MultiThreadTest.java	非ブロッキング JDBC/MX ドライバーの機能を示します。デフォルトでは、このプログラムは、2 つのスレッドを作成します。非ブロックモードで、これらの 2 つのスレッドを同時に実行します。このプログラムは、各操作の前後におけるスレッド ID と SQL 操作のステータスを表示します。ブロックモードでプログラムを実行すると、トランザクション開始操作が、SQL 非待機モードで開始されるため、1 つのみのスレッド スイッチを観察します。プログラムを非ブロックモードで実行すると、多くのスレッドスイッチ を観察できます。
holdJdbcMx.java	JDBC/MX ドライバーにおける保持可能カーソル サポートを示します。プログラムは、メッセージ待ち行列をサブスクライブするサブスクライバー スレッドを作成します。メッセージ待ち行列内のすべての行を読み、サブスクライバーは、5 秒後にタイム アウトします。
TestConnectionPool.java	接続プーリングおよびステートメント プーリングの利点を示します。このプログラムは、JDBC 接続を行い、いくつかの SQL 文を実行し、接続を閉じるループを 100 回実行します。このプログラムで OSS time() コマンドを使用して、接続プーリングとステートメント プーリングの性能上の利点を測定します。
CreateTraceDS.javaTest および TraceDS.Java	トレースするドライバー固有データソースの周りにラッパーを作成することによって、トレースを示します。これらのデモ プログラムは、製品インスール ディレクトリの /demo にあります。
JdbcRowSetSample.java	SQLMXJdbcRowSet オブジェクトを作成し、いくつかの JdbcRowSet メソッドを呼び出す方法を示します。
LobSample.java	JDBC/MX ドライバーの LOB 機能を示します。
TransactionMode.java	内部、外部、および混合トランザクション モードを示します。
ISO88591Sample.java	**ISO88591 プロパティ**(31 ページ) を示します。

これらの見本プログラムを実行する方法については、JDBC/MX ドライバー ソフトウェアに付属の README ファイルを参照してください。

JDBC/MX のインストールおよび検証

Softdoc ファイルには、JDBC/MX タイプ 2 ドライバーを NonStop SQL/MX リリース 3.6 インストールする方法に関する説明が含まれます。

本セクションでは、以下について説明します。

インストール要件

NonStop SQL/MX 用 JDBC/MX T2 ドライバーのハードウェアおよびソフトウェア要件は、JDBC/MX ドライバーに含まれる Softdoc ファイルで説明されています。

JDBC/MX ドライバーのバージョンは、NonStop SQL/MX リリース 3.6 用 JDBC/MX ドライバー 3.6 (製品 T1275 として識別される) です。

JDBC/MX ドライバーでは、次のソフトウェアが必要です。

- NonStop SQL/MX リリース 3.0 以降
- Java 2 Platform Standard Edition 5.0 (T2766H50) またはそれ以降の製品アップデートに基づく NonStop Server for Java。
- NonStop Server for Java 7.0 またはそれ以降の製品アップデータ (64 ビット サポート用)

  留意事項

  JDBC T2 ドライバーには､Java ネイティブ ライブラリ用に最小の 128 MB が必要です。Java プロセスのヒープ領域用メモリを予約するときに、1 つのプロセスあたり少なくとも 128 MB 以上が必要です (ヒープ領域用に使用できる最大メモリ制限)。

注記: ソフトウェアの要件の最新の文、および、必須ソフトウェアの最新サポートバージョンのリストについては、Softdoc ファイルを参照してください。一般に、同じ製品の新しいバージョンを代わりに用いることができます。しかし、Softdoc で説明される SPR 要件が満たされていることを確認しなければなりません。

JDBC/MX T2 ドライバーのインストール

JDBC T2 ドライバーをインストールするには:

手順

1. `super.super` として、NonStop システムにログオンします。
2. TACL プロンプトに変更します。
3. JDBC T2 ドライバーは、`$SYSTEM.ZOSSUTL` で利用可能です。`$SYSTEM.ZOSSUTL` subvolume に変更するには、次を入力します。

```
$SYSTEM STARTUP 1> volume $SYSTEM.ZOSSUTL
```

4. 次のいずれかのコマンドを実行します。

   a. ドライバーを標準の場所 (/usr/tandem/jdbcMx/< *バージョン* >) に解凍するには、次を入力します。

   ```
   $SYSTEM ZOSSUTL > PINSTALL -pp -rvf T1275PAX
   ```

   PINSTALL は、T1275PAX の内容を /usr/tandem/jdbcMx/< *バージョン* > OSS ディレクトリに解凍します。

   b. ドライバーを非標準の場所に解凍するには、次を入力します。

   ```
   $SYSTEM ZOSSUTL > PINSTALL -pp-s:/usr/tandem: <jdbc_install_dir>: - rvf T1275PAX
   ```

   例えば、内容を /home/test に解凍するには:

   ```
   $SYSTEM ZOSSUTL > PINSTALL -pp -s:/usr/tandem:/home/test: -rvf T1275PAX
   ```

   PINSTALL は、T1275PAX の内容を /home/test/jdbcMx/<*version*> OSS ディレクトリに提供します。

注記: <*jdbc_install_dir*> の標準の位置は、次のとおりです。 /usr/tandem/jdbcMx/T1275< *シリーズ* >< *バージョン* >

ここで、

- < *シリーズ* > - RVU シリーズ、H (J-シリーズ RVU) または L (L-シリーズ RVU) のいずれかを示します。
- < *バージョン* > - 製品バージョンを示します。例えば、3.4 リリースの場合は 34 です。

---

5. OSS シェル セッションを開始するには、次のように入力します。

   ```
   $SYSTEM ZOSSUTL> osh
   ```

6. 作業ディレクトリに変更するには、以下を入力します。

   ```
   /G/SYSTEM/SYSTEM: cd <jdbc_install_dir>/jdbcMx/<version>/bin
   ```

7. インストールスクリプトを実行するには、以下を入力します。

   <*jdbc_install_dir*>/jdbcMx/< *バージョン* >/bin:./jdbcMxInstall

---

注記: JDBC/MX 製品ディレクトリに対する一貫性のあるパス名を提供するために、シンボリックリンク <*jdbc_install_dir*>/jdbcMx/current 例えば、JDBC T2 ドライバーを標準インストール場所にインストールすると、/usr/tandem/jdbcMx/current が、/usr/tandem/jdbcMx/< *バージョン* > ディレクトリに対するシンボリックリンクになります。

---

8. 次のプロプンプトに、**Enter** をクリックします。

   Please specify the full path to the NonStop Java Installation directory [for example /usr/tandem/nssjava/jdk123]: [/usr/tandem/java]: （NonStop Java インストール ディレクトリの [例/usr/tandem/nssjava/jdk123] へのフルパスを指定してください: [/usr/tandem/java]。）

9. 次のプロプンプトに、**y** を入力します。

   Are the above locations correct [y] ?（上記の位置正しい [y] ですか。）

次のメッセージが表示されます。

*** Please Wait.（しばらくお待ちください。） Checking for the correct Java version.（Java の正しいバージョンを確認しています。） *** Correct version of Java Found.（適切なバージョンの Java を検出しました。） The install is now finished.（インストールが終了しました。） Make sure that the CLASSPATH environment variable contains the full path to the jdbcMx.jar file in the JDBC/MX library directory and the _RLD_LIB_PATH environment variable contains the full path to the JDBC/MX library directory.（CLASSPATH 環境変数には、JDBC/MX ライブラリ ディレクトリに jdbcMx.jar ファイルへのフルパスが含まれていること、_RLD_LIB_PATH 環境変数には、JDBC/MX ライブラリ ディレクトリへのフルパスが含まれていることを確認してください。） 注記： 特定の製品バージョン名の代わりに「current」のシンボリックリンクをパスに使用できます。

10. <*jdbc_install_dir*>/jdbcMx/< *バージョン* >/lib/jdbcMx.jar JDBC/MX JAR ファイル名を CLASSPATH 環境変数に追加するには、以下を入力します。

    <*jdbc_install_dir*>/jdbcMx/< *バージョン* >/bin: export CLASSPATH = <*jdbc_install_dir*>/jdbcMx/< *バージョン* >/lib/jdbcMx.jar:$CLASSPATH

11. <*jdbc_install_dir*>/jdbcMx/<*version*>/lib JDBC/MX TNS/E PIC オブジェクト形式ライブラリからのパスを _RLD_LIB_PATH 環境変数に追加するには、以下を入力します。

    export _RLD_LIB_PATH = <*jdbc_install_dir*>/jdbcMx/< *バージョン* >/lib/

注記: JDBC プログラムを実行またはコンパイルする前に、ログオンするたびに手順 10 および 11 を実行してください。また、T1275R34 を current で置き換えることができます。

JDBC/MX ドライバーファイルの位置

JDBC/MX ドライバー ソフトウェアの JDBC/MX ドライバーのインストールディレクトリの位置は、次のとおりです、

<*jdbc_install_dir*>/jdbcMx/<*バージョン*>

デフォルトのインストールディレクトリの位置は、次のとおりです。

/usr/tandem/jdbcMx/<*バージョン* >

インストールされるファイルは、次のとおりです。

.../demo

デモプログラム

.../lib/libjdbcMx.so

JDBC/MX ドライバー ライブラリ

.../lib/jdbcMx.jar

JDBC/MX Java アーカイブ ファイルには、JDBC トレース機能が含まれています

.../lib/libjdbcMx64.so

JDBC/MX ドライバーの 64 ビット ライブラリ

.../bin/jdbcMxInstall

JDBC/MX インストールスクリプト

.../bin/jdbcMxUninstall

JDBC/MX ドライバーの確認

JDBC/MX ドライバーのバージョンを確認するには、これらのコマンドを使用します。

- java コマンド、JDBC/MX ドライバーの Java コード部分のバージョンを表示します
- vproc コマンド、JDBC/MX ドライバーの C コード部分のバージョンを表示します

java コマンドを使用するには、以下を OSS プロンプトで入力します。

```
java JdbcMx -version
```

このコマンドの出力は次のようになります。

```
JDBC driver for NonStop(TM) SQL/MX Version T1275R32_30AUG2012_JDBCMX_0419
```

出力を Java 5 ディストリビューション CD の NonStop サーバーの Softdoc ファイルの製品番号と比較します。

vproc コマンドを使用して、正しいライブラリであるかどうかを確認します。次のコマンドを OSS プロンプトで実行します。

32 ビットの場合:

```
vproc /jdbcmx-installation-directory/T1275R32/lib/libjdbcMx.so
...
Binder timestamp:  29OCT2012 09:07:46 Version procedure:T1275R32_20FEB2013_JDBCMXAMU_1029 TNS/E Native
Mode:  runnable file
```

64 ビットの場合:

```
vproc /jdbcmx-installation-directory/T1275R32/lib/libjdbcMx64.so
...
Binder timestamp:29OCT2012 09:06:59 Version procedure:  T1275R32_20FEB2013_JDBCMXAMU_1029 TNS/E Native
Mode:  runnable file
```

JDBC/MX ドライバー (T1275) 製品に対応するバージョンプロシージャは、java および vproc コマンドの両方の出力で一致する必要があります。

CLASSPATH の設定

実行中の JDBC アプリケーションについて、CLASSPATH 環境変数に jdbcMx.jar ファイルが含まれていることを確認します。デフォルトインストールを仮定すると、パスは

/usr/tandem/jdbcMx/current/lib/jdbcMx.jar になります

_RLD_LIB_PATH の設定

JDBC アプリケーションを実行する前に、_RLD_LIB_PATH 環境変数パスが、libjdbcMx.so JDBC/MX DLL ファイルを含むディレクトリに設定されていることを確認します。デフォルト インストールの場合、パスは次のようになります。

/usr/tandem/jdbcMx/T1275 < シリーズ >< バージョン >/< シリーズ >< バージョン >/lib

ここで、

- < *シリーズ* > - RVU シリーズ、H (J-シリーズ RVU) または L (L-シリーズ RVU) のいずれかを示します。
- < *バージョン* > - 製品バージョンを示します。例えば、3.4 リリースの場合は 34 です。

例えば、リリース 3.4 のデフォルト インストールパスは、`/usr/tandem/jdbcMx/T1275L34/L34/lib` です。

SQL/MX を用いる SQL データベースへのアクセス

本セクションでは、次のトピックについて説明します。

T2 JDBC ドライバーにおけるユーザー認証

デフォルトでは、ドライバーは、接続要求中に提供されるユーザー認証情報を認証せず、ユーザー認証情報を無視します。

JDBC T2 ドライバーのリリース 3.4 から、ユーザー認証情報を認証するドライバーを構成できます。

認証の有効化

デフォルトでは、JDBC T2 ドライバー認証は無効です。JDBC T2 ドライバーは、デフォルトの非ブロッキング モードでのみ、認証をサポートします。認証を有効化するには、以下の方法のいずれかを使用します。

- `=_JDBCMX_AUTHENTICATION` 定義を ON に設定します。 `add_define =_JDBCMX_AUTHENTICATION CLASS=MAP FILE=ON` ユーザー認証情報は、Java アプリケーションにおいて、すべての JDBC 接続に対して認証されます。
- `authenticationMode` コマンドライン プロパティを ON に設定します。 `-Djdbcmx.authenticationMode=ON` ユーザー認証情報は、

- DriverManager クラス API を使用するために取得済みのすべての JDBC 接続に対して認証されます。
- authenticationMode が、これらのオブジェクト上に設定されていない場合、DataSource および ConnectionPoolDataSource クラス API

- authenticationMode プロパティを DataSource または ConnectionPoolDataSource オブジェクト上に設定します。アプリケーションは、認証有効および認証無効データソースの両方を作成できます。

認証に関する留意事項

JDBC T2 ドライバーでの認証を有効にする前に、以下を考慮してください。

- 認証に成功すると、認証済みユーザーに対するプロセス クリエーター アクセス ID および プロセスアクセス ID が変更されます。
- JDBC T2 ドライバーは、Guardian API USER_AUTHENTICATE_ を使用して、接続要求において提供されたユーザー認証情報を検証します。無効なユーザー認証情報が接続要求時に渡されると、アプリケーション プロセスがサスペンドされる可能性があります。Safeguard が使用されていない場合、無効なパスワードを USER_AUTHENTICATE_ API に引き渡すと、NonStop システム上のユーザーが 3 回連続してログオン試行時に、プロセスを 60 秒間 (タイムアウト時間) サスペンドする場合があります。

  注記: 最初の 2 つの失敗したログオン試行は、別のプロセスから実行される場合があります。

  Safeguard を使用しているときに、プロセスをサスペンドさせる前の失敗したログオン試行回数、および、プロセスをサスペンドするタイムアウト期間を設定できます。

  AUTHENTICATE-MAXIMUM-ATTEMPTS 属性を使用して、連続する最大ログイン失敗回数を設定します。Safeguard ソフトウェアは、最大ログイン失敗回数に到達した後、ユーザー ID をフリーズするまたはタイムアウトします。AUTHENTICATE-FAIL-TIMEOUT 属性を使用して、タイムアウト時間を設定します。ログオンの失敗の最大数を超過すると、このタイムアウト時間後、プロセスがサスペンドされます。

  △ **注意:** 認証を有効にする前に、Java アプリケーションのサスペンドの影響を検討してください。Safeguard 構成属性を設定することによって影響を軽減することができます。

- JDBC T2 ドライバーにより、ユーザーは、異なるデータソースに接続できます。アプリケーションが、複数のユーザーが所有する接続を使用すると、頻繁にプロセスの所有権のスイッチが切り替わり、接続において SQL 文を実行する際に、プロセスは、接続を所有しているユーザーによって所有されます。残りのアプリケーションを実行する際に、プロセスは、アプリケーションを起動したユーザーの所有権の下にあります。**例を参照してください**
- Java 用 ストアド プロシージャー (SPJ) の手順で認証を有効にするには、ドライバーは非ブロックモードでのみ認証をサポートするので、非ブロックモード用に SQL/MX UDR サーバーを構成します。JVM を起動するには、–Djdbcmx.sqlmx_nowait プロパティを ON に設定します。

  注記: マルチスレッド アプリケーションの CALL 文は、非ブロックモードでのみ実行できます。しかし、これらの CALL 文に対応する SPJ メソッドは、与えられた SPJ 環境内でシリアルに実行します。SPJ 環境は、シングル スレッドであり、ブロックモードは、CPU パスの長さを減少させ、SPJ 環境での性能を改善します。デフォルトでは、SQL/MX UDR サーバーは、ブロックモードで動作します。

SQL 文を実行する際のユーザーの切り替え

userA が所有するアプリケーション プロセスは、userB が所有する connection1 と userC が所有する connection2 の 2 つの接続を作成します。プロセスアクセス ID は、connection1 で SQL 文を実行する場合は、userA から userB に、connection2 で SQL 文を実行する場合は、userA から userC に切り替わります。

認証モードの設定

認証用に設定される DEFINE = _JDBCMX_AUTHENTICATION は、authenticationMode プロパティよりも高い優先順位を持ちます。次の表は、さまざまな定義およびプロパティ設定に対する認証動作を列挙します。

DEFINE 設定	プロパティ設定	接続に対する認証動作
未設定	未設定	OFF
未設定	ON	ON
未設定	OFF	OFF
ON	未設定	ON
ON	ON	ON
ON	OFF	ON
OFF	未設定	OFF
OFF	ON	OFF
OFF	OFF	OFF

authenticationMode プロパティ設定の優先順位は、次のとおりです。

1. setAuthenticationMode メソッドを使用して、アプリケーションで指定されます。例えば、ds.setAuthenticationMode です。
2. JNDI ConnectionPoolDataSource オブジェクトで設定されます。
3. JNDI SQLMXDataSource オブジェクトで設定されます。
4. コマンドライン プロパティ。

詳細情報

- **接続プール認証条件**
- **デフォルトのユーザーおよびパスワード**
- **JdbcRowSet のインプリメンテーション**
- **認証モードの設定**

SQL/MX への接続

Java アプリケーションは、SQL/MX への JDBC 接続を 2 つの方法で入手できます。

- DriverManager クラスを使用する方法
- DataSource インターフェイスを使用する方法

DriverManager クラスを使用する接続

これは、データベースへの接続を確立する従来の方法です。DriverManager クラスは、Driver インターフェイスのと共に動作して、ロードされた一連のドライバーを管理します。アプリケーションが、DriverManager.getConnection メソッドを使用して、接続用要求を発行し、URL を提供するとき、DriverManager は、この URL を認識する適切なドライバーを検索し、このドライバーを使用して、データベース接続を取得する責任を負います。

com.tandem.sqlmx.SQLMXDriver は、Driver インターフェイスを実装する JDBC/MX ドライバー クラスです。アプリケーションは、**DriverManager クラスと共に使用される JDBC/MX ドライバー プロパティ**の表に記載される場合を除き、次の方法いずれかで、JDBC/MX ドライバーをロードできます。

- コマンドラインの -Djdbc.drivers オプションで、MX/JDBC ドライバー クラスを指定します。
- アプリケーション内、Class.ForName メソッドを使用します。
- JDBC/MX ドライバー クラスをアプリケーション内の jdbc.drivers プロパティに追加します。

DriverManager.getConnection メソッドが、JDBC URL を含む文字列を受け取ります。JDBC/MX ドライバー用 JDBC URL は、jdbc:sqlmx: です。

DriverManager クラス使用して接続するときは、次のトピックの情報を使用します。

- **DriverManager クラスと共に使用する JDBC/MX ドライバー プロパティ** (23 ページ)
- **DriverManager クラスとの接続を使用するためのガイドライン** (25 ページ)

DriverManager クラスと共に使用する JDBC/MX ドライバー プロパティ

JDBC/MX ドライバーは、ドライバーの構成に使用できる次の一連のプロパティを定義します。

表 2: DriverManager クラスと共に使用する JDBC/MX ドライバー プロパティ

プロパティ名	タイプ	値	説明
authenticationMode	文字列	オンまたはオフ	認証を有効にするには、このプロパティを ON に設定します。デフォルト値は OFF です。
contBatchOnError	文字列	オンまたはオフ	任意の BatchUpdateExceptions した後でも、バッチで残りのジョブを続行するために JDBC ドライバーと通信します。**contBatchOnError プロパティ**(47 ページ) を参照してください。
catalog（カタログ）	文字列	**デフォルトのカタログおよびスキーマ** (30 ページ) を参照してください。	デフォルト カタログおよびスキーマを指定しない場合、JDBC/MX ドライバー により、SQL/MX は、デフォルトの独自のルールに従うことができます。

表は続く

プロパティ名	タイプ	値	説明
schema	文字列	**デフォルトのカタログおよびスキーマ** (30 ページ) を参照してください。	上記の catalog を参照してください。
mploc	文字列	**mploc プロパティ** (31 ページ) を参照してください。	SQL/MP テーブルが作成される位置 ($volume.subvolume (デフォルト位置は、ログオンしているユーザーのデフォルト サブボリュームです)。
enableLog	boolean	オンまたはオフ	SQL 文の ID および対応する JDBC SQL 文のロギングを有効化します。**enableLog プロパティ**(95 ページ) を参照してください。
idMapFile	文字列	有効な OSS ファイル名	トレース機能が、SQL 文 ID および対応する JDBC SQL 文を記録するファイルを指定します。**idMapFile プロパティ**(95 ページ) を参照してください。
ISO88591	文字列	**ISO88591 プロパティ** (31 ページ) を参照してください。	ISO88591 列にアクセスおよび書き込む際に使用する Java エンコーディングを指定します。
maxStatements	int	**maxStatements プロパティ**(32 ページ) を参照してください。	接続プールがキャッシュを行う必要がある PreparedStatement オブジェクトの総数。**maxStatements プロパティ**(32 ページ) を参照してください。
minPoolSize	int	**minPoolSize プロパティ**(32 ページ)を参照してください。	フリー接続プールに存在できる物理的な接続の数を制限します。minPoolSize のプロパティを参照してください。
maxPoolSize	int	**maxPoolSize プロパティ**(32 ページ) を参照してください。	プールが含む必要がある物理的な接続の最大数を設定します。この数には、フリー接続および使用中の接続の両方が含まれています。**maxPoolSize プロパティ**(32 ページ) を参照してください。
initialPoolSize	int	**initialPoolSize プロパティ**(32 ページ) を参照してください。	JDBC/MX ドライバーと共に接続プーリングを使用する場合、初期接続プーリング サイズを設定します。**initialPoolSize プロパティ**(32 ページ) を参照してください。
maxIdleTime	int	**maxIdletime プロパティ**(33 ページ) を参照してください。	接続が閉じられる前に、プールにおいて、未使用の物理的な接続の残り秒数を表します。**maxIdletime プロパティ**(33 ページ) を参照してください。
language	文字列	**言語プロパティ**(33 ページ) を参照してください。	エラーメッセージで使用される言語を設定します。**言語プロパティ**(33 ページ) を参照してください。

表は続く

プロパティ名	タイプ	値	説明
blobTableName	文字列	**LOB テーブル名のプロパティ**(30 ページ) を参照してください。	BLOB 列に使用する、LOB テーブルを指定します。
clobTableName	文字列	**LOB テーブル名のプロパティ**(30 ページ) を参照してください。	CLOB 列に使用する、LOB テーブルを指定します。
transactionMode	文字列	**transactionMode プロパティ**(34 ページ) を参照してください。	トランザクションが実行される方法と制御を提供するトランザクション モードを設定します。**transactionMode プロパティ**(34 ページ) を参照してください。
user	文字列	有効なユーザー名。	認証用ユーザーを設定します。
password	文字列	user に対する有効なパスワード。	認証用パスワードを設定します。

注記: プロパティがパラメーターとして接続メソッドに与えられるとき、データソースを使用するときは、jdbcmx プレフィックスをプロパティ名に追加しないでください。プレフィックスは、プロパティが JDBC/MX ドライバー オブジェクトに渡されるために、プロパティ タイプを識別するために必要はありません。**コマンドラインでプロパティの設定**(36 ページ) で説明するように、コマンドラインでのみ、jdbcmx プレフィックスを使用します。

DriverManager クラスとの接続を使用するためのガイドライン

- Java アプリケーションでは、次の方法でプロパティを指定できます。
  - JDBC/MX プロパティをコマンドラインで、-D オプション共に使用します。使用する場合、このオプションは、Java アプリケーション内の DriverManager を使用するすべての JDBC 接続に適用されます。コマンドラインで入力する形式は、次のとおりです。

    -Djdbcmx.**property_name=property_value**

    例えば、コマンドライン行で、-Djdbcmx.maxStatements=1024
  - DriverManager の getConnection メソッドで、java.util.properties パラメーターを使用します。
- java.util.properties パラメーターを通して渡されるプロパティは、コマンドライン プロパティよりも高い優先順位を持ちます。
- 接続プーリング機能は、Java アプリケーションが、DriverManager クラスを使用して、JDBC 接続を取得するときに利用可能です。接続プール サイズは、maxPoolSize プロパティの値および minPoolSize プロパティの値により決定されます。
- JDBC/MX ドライバーが、カタログおよびスキーマの組み合わせについての接続プール マネージャーを含むため、同じカタログおよびスキーマの組み合わせを持つ接続に一緒にプールされます。特定のスキーマおよびカタログの組み合わせについての最初の接続の取得時に使用される接続プーリング プロパティ値

は、プロセスのライフサイクル全体にわたって有効です。アプリケーションは、特定のカタログおよびスキーマの組み合わせについての最初の接続以降のこれらのプロパティ値を変更できません。

- 同様に、基本的なデータソースオブジェクト インプリメンテーションとして、Java アプリケーションは、プロパティを 0 以外の正の値に設定することにより、ステートメント プーリングを有効化できます。

DataSource インターフェイスを使用する接続

JDBC 2.0 オプション パッケージに導入された DataSource インターフェイスは、アプリケーションのポータビリティを強化するため、データベースに対する接続を確立するための好ましい方法です。JDBC/MX ドライバーは、アプリケーションが、DataSource インターフェイスの getConnection メソッドを使用して、接続を要求するとき、DataSource インターフェイスを実装し、接続オブジェクトを返します。

DataSource オブジェクトを使用し、アプリケーションにドライバー固有の情報を提供する代わりに、データソースの論理名の使用を許可することにより、アプリケーションのポータビリティが増加します。論理名は、Java ネーミングおよびディレクトリ インターフェイス (JNDI) を使用するネームサービスの手段により、DataSource オブジェクトにマッピングされます。

新しい情報と変更された情報(10 ページ) は、JDBC/MX データ ソース オブジェクトの識別に使用できるプロパティについて説明します。

表 3: DataSource オブジェクト プロパティ

プロパティ名	タイプ	値	説明
authenticationMode	文字列	オンまたはオフ	認証を有効にするには、このプロパティを ON に設定します。デフォルト値は OFF です。
contBatchOnError	文字列	オンまたはオフ	任意の BatchUpdateExceptions した後でも、バッチで残りのジョブを続行するために JDBC ドライバーと通信します。**contBatchOnError プロパティ**(47 ページ) を参照してください。
catalog （カタログ）	文字列	**デフォルトのカタログおよびスキーマ** (30 ページ) を参照してください。	デフォルト カタログおよびスキーマを指定しない場合、JDBC/MX ドライバー により、SQL/MX は、デフォルトの独自のルールに従うことができます。
schema	文字列	**デフォルトのカタログおよびスキーマ** (30 ページ) を参照してください。	上記の catalog を参照してください。
dataSourceName	文字列		登録された ConnectionPoolDataSource 名。この文字列が空の場合、基本 DataSource オブジェクトの maxPoolSize プロパティおよび minPoolSize プロパティにより決定されるプール サイズを持つ接続プーリングがデフォルトで使用されます。詳細情報については、**Basic DataSource API を使用する接続プーリング**(43 ページ) を参照してください。

表は続く

プロパティ名	タイプ	値	説明
説明	文字列	任意の有効な識別子	データソースの説明。
enableLog	boolean	オンまたはオフ	SQL 文の ID および対応する JDBC SQL 文のロギングを有効化します。**enableLog プロパティ**(95 ページ) を参照してください。
idMapFile	文字列	有効な OSS ファイル名	トレース機能が、SQL 文 ID および対応する JDBC SQL 文を記録するファイルを指定します。**idMapFile プロパティ**(95 ページ) を参照してください。
ISO88591	文字列	**ISO88591 プロパティ**(31 ページ) を参照してください。	ISO88591 列にアクセスおよび書き込む際に使用する Java エンコーディングを指定します。
maxPoolSize	int	**maxPoolSize プロパティ**(32 ページ) を参照してください。	プールが含む必要がある物理的な接続の最大数を設定します。この数には、フリー接続および使用中の接続の両方が含まれています。**maxPoolSize プロパティ**(32 ページ) を参照してください。
initialPoolSize	int	**initialPoolSize プロパティ**(32 ページ) を参照してください。	JDBC/MX ドライバーと共に接続プーリングを使用する場合、初期接続プーリング サイズを設定します。**initialPoolSize プロパティ** (32 ページ) を参照してください。
maxIdleTime	int	**maxIdletime プロパティ**(33 ページ) を参照してください。	接続が閉じられる前に、プールにおいて、未使用の物理的な接続の残り秒数を表します。**maxIdletime プロパティ**(33 ページ) を参照してください。
language	文字列	**言語プロパティ**(33 ページ) を参照してください。	エラーメッセージで使用される言語を設定します。**言語プロパティ**(33 ページ) を参照してください。
maxStatements	int	**maxStatements プロパティ**(32 ページ) を参照してください。	接続プールがキャッシュを行う必要がある PreparedStatement オブジェクトの総数。**maxStatements プロパティ**(32 ページ) を参照してください。
minPoolSize	int	**minPoolSize プロパティ**(32 ページ) を参照してください。	フリー接続プールに存在できる物理的な接続の数を制限します。**minPoolSize プロパティ** (32 ページ) を参照してください。
mploc	文字列	**mploc プロパティ** (31 ページ) を参照してください。	SQL/MP テーブルが作成される位置 ($volume.subvolume 形式) (デフォルト位置は、ログオン ユーザーのデフォルト サブボリュームです。)

表は続く

プロパティ名	タイプ	値	説明
blobTableName	文字列	**LOB テーブル名のプロパティ**(30 ページ) を参照してください。	BLOB 列に使用する、LOB テーブルを指定します。
clobTableName	文字列	**LOB テーブル名のプロパティ**(30 ページ) を参照してください。	CLOB 列に使用する、LOB テーブルを指定します。
transactionMode	文字列	**transactionMode プロパティ**(34 ページ) を参照してください。 .	トランザクションが実行される方法と制御を提供するトランザクション モードを設定します。**transactionMode プロパティ**(34 ページ) を参照してください。
user	文字列	有効なユーザー名。	認証用ユーザーを設定します。
password	文字列	user に対する有効なパスワード。	認証用パスワードを設定します。

注記: プロパティがパラメーターとして接続メソッドに与えられるとき、データソースを使用するときは、jdbcmx プレフィックスをプロパティ名に追加しないでください。プレフィックスは、プロパティが JDBC/MX ドライバー オブジェクトに渡されるために、プロパティ タイプを識別するために必要はありません。**コマンドラインでプロパティの設定**(36 ページ) で説明するように、コマンドラインでのみ、jdbcmx プレフィックスを使用します。

JdbcRowSet のインプリメンテーション

JdbcRowSet インターフェイス、SQLMXJdbcRowSet のインプリメンテーションは、com.tandem.sqlmx パッケージ内で提供されます。ResultSet オブジェクトに類似する JdbcRowSet オブジェクトは、データベースに対する接続を維持します。しかし、JdbcRowSet オブジェクトは、それを JavaBeans™ コンポーネントにする一連のプロパティおよびリスナー通知メカニズムを維持します。

SQLMXJdbcRowSet オブジェクトは、これらの SQLMXJdbcRowSet コンストラクターを使用して作成できます。

- デフォルト コンストラクターは、パラメーターを必要としません。

  JDBC T2 ドライバーが認証用に有効化される場合、execute メソッドを呼び出す前に、次のメソッドを使用して、ユーザーおよびパスワードを設定してください。

  - public void setUsername(String user)
  - public void setPassword(String password)

注記: コンストラクターは、`execute` メソッドが呼び出されるまで、データベースへの接続を試行しません、

- コンストラクターは、`Connection` オブジェクトを取得します。
- コンストラクターは、`ResultSet` オブジェクトを取得します。

コンストラクターは、URL、ユーザー名、パスワードを取得します。

注記: ユーザー名およびパスワードの属性は、SQL/MX 3.4 リリースからサポートされます。

`JdbcRowSetSample.java` デモプログラムを `SQLMXJdbcRowSet` オブジェクトのインスタンス化および操作の例として参照してください。また、`SQLMXJdbcRowSet` オブジェクト固有のインプリメンテーション詳細については、「未サポートおよび逸脱」のセクションを参照してください。

追加の詳細として、以下で、`JdbcRowSet` インターフェイスの仕様を参照してください

http://docs.oracle.com/javase/1.5.0/docs/api/javax/sql/rowset/JdbcRowSet.html

JDBC/MX プロパティ

本セクションでは、`DriverManager` クラス (**DriverManager クラス** テーブルと共に使用される JDBC/MX ドライバーのプロパティ) および DataSource オブジェクト プロパティ (**DataSource オブジェクト プロパティ** テーブル) に含まれる JDBC/MX プロパティを説明します。

- **デフォルトのカタログおよびスキーマ**(30 ページ)
- **デフォルトのユーザーおよびパスワード**(30 ページ)
- **LOB テーブル名のプロパティ**(30 ページ)
- **ISO88591 プロパティ**(31 ページ)
- **mploc プロパティ**(31 ページ)
- **maxStatements プロパティ**(32 ページ)
- **minPoolSize プロパティ**(32 ページ)
- **maxPoolSize プロパティ**(32 ページ)
- **initialPoolSize プロパティ**(32 ページ)
- **maxIdletime プロパティ**(33 ページ)
- **言語プロパティ**(33 ページ)
- **transactionMode プロパティ**(34 ページ)
- **queryExecuteTime プロパティ**(35 ページ)
- **T2QueryExecuteLogFile プロパティ**(35 ページ)
- **コマンドラインでプロパティの設定**(36 ページ)

これらのプロパティおよび追加プロパティは、**コマンドラインでプロパティの設定**(36 ページ) で説明するように、コマンドラインで指定できます。

さまざまな JDBC/MX プロパティにより提供される機能の使用に関する情報については、トピック **追加 JDBC/MX プロパティの使用**(47 ページ) を参照してください。

デフォルトのカタログおよびスキーマ

SQL オブジェクトの完全修飾されていない場合、SQL 文で参照されている SQL オブジェクトにアクセスするために、デフォルトのカタログおよびスキーマが使用されます。SQL/MX オブジェクトの 3 つの部分の完全修飾名の形式は次のとおりです。

[[**catalog.**]**schema.**]**object-name**

カタログおよびスキーマ名は、SQL 識別子に準拠して、任意の文字列にすることができます。これらの名前は、ANSI SQL:99 スキーマおよびカタログ名に準拠します。

例えば、CAT.SCH.TABLE として参照されるテーブルに対して、デフォルトのカタログおよスキーマ プロパティを使用するオプションは、次のとおりです。

```
-Djdbcmx.catalog=CAT -Djdbcmx.schema=SCH
```

詳細情報については、最新の **HPE NonStop SQL/MX リリース リファレンスマニュアル**を参照してください。

デフォルトのユーザーおよびパスワード

認証が有効な場合、接続を取得するために、ユーザーおよびパスワードを提供しなければなりません。ユーザーおよびパスワードが、接続 API で提供されていなかったり、SQLMXDataSource オブジェクトで設定されてない場合、ドライバーは、認証例外を返します。この例外を回避するために、デフォルトのユーザーおよびパスワードを設定できます。ユーザー名またはユーザー ID やユーザー エイリアス名を指定できます。例えば、SQL.USER1 または 125,12 です。コマンドラインから、デフォルトのユーザーおよびパスワードを設定します。例えば、-Djdbcmx.user=super.super -Djdbcmx.password=superuser ユーザーおよびパスワードに対する JDBC T2 ドライバーの優先順位は以下のとおりです。

1. 接続要求で指定されています
2. SQLMXDataSource オブジェクトで設定されています
3. コマンドラインで指定されたデフォルトのユーザーおよびパスワード

LOB テーブル名のプロパティ

LOB テーブルは、LOB 列用データを格納します。BLOB 列または CLOB 列を使用するたの LOB テーブルを指定するために使用するプロパティは、次のとおりです。

BLOB 列の場合

blobTableName

CLOB 列の場合

clobTableName

プロパティ値の形式は、次のとおりです。

catalog_name.schema_name.lob_table_name

次の方法でプロパティを使用して、LOB テーブルの名前を指定できます。

- Java コマンドラインで、-Djdbcmx.property_name=property_value オプションを使用します 例:

  -Djdbcmx.clobTableName=mycat.myschema.myLobTable

- DriverManager クラスの getConnection メソッドで、java.util.Properties パラメーターを使用します プロパティ パラメーターで渡されるプロパティは、コマンドライン プロパティよりも高い優先順位を持ちます。
- DataSource で、これらのプロパティのいずれかを設定します。**DataSource インターフェイスを使用する接続**(26 ページ) を参照してください。

ISO88591 プロパティ

ISO88591 キャラクターセット マッピング プロパティは、SQL/MX ISO88591 キャラクターセットに対応します。これは、キャラクター データタイプに対するシングル-バイトの 8 ビット キャラクターセットです。ISO88591 キャラクターセットは、英語およびその他の西ヨーロッパ言語をサポートします。ISO88591 プロパティを以下のように指定します

文字列

デフォルト値は DEFAULT です。これは、ISO88591 列にアクセス・書き込みをするとき、デフォルト Java エンコーディングを使用します。値は、Oracle ドキュメントの「Canonical Name for java.io および java.lang API 用正式名称」列に列挙される有効な Java 正式名称にすることができます。詳細情報については、**サポートされるエンコーディング**を参照してください。

例えば、漢字データが、列キャラクターセットを使用して、SQL/MP テーブル (SQL/MX を通してアクセス) の ISO88591 列に格納されており、データベースに読み書きされている場合、次のプロパティを指定して、正しいエンコーディングを確認できます。

-Djdbcmx.ISO88591=SJIS

mploc プロパティ

プロパティ mploc は、SQL テーブルが作成される Guardian 位置を指定します。mploc の形式は、次のとおりです。

[**node**].$**volume.subvolume**

JDBC/MX ドライバーを使用する Java アプリケーションは、コマンドラインで、-D を用いて、システムプロパティ mploc を使用することにより、mploc を指定できます。

-Djdbcmx.mploc=**mploc**

例えば、OSS 環境における DriverManager オブジェクトの場合、次の形式のいずれかで、mploc プロパティを指定します。

-Djdbcmx.mploc=[**node.**]\$**volume.subvolume**

または

-Djdbcmx.mploc='[**node.**]$**volume.subvolume**'

詳細情報については、最新の **HPE NonStop SQL/MX リリース リファレンスマニュアル**を参照してください。

maxStatements プロパティ

接続プールがキャッシュする PreparedStatement オブジェクトの合計数を設定します。この総数には、フリー オブジェクトのおよび使用中のオブジェクトが含まれます。以下のように maxStatements プロパティを指定します。

int

整数は、0 ~ 2147483647 にすることができます。負の値は、0 のように扱われます。デフォルトでは 0 で、ステートメント プーリングを無効にします。ヒューレット・パッカード エンタープライズでは、このプーリングは、多くのアプリケーションの性能を劇的に改善するため、JDBC アプリケーションに対して、ステートメント プーリングを有効化することをお勧めします。

minPoolSize プロパティ

フリー接続プールに存在できる物理的な接続の数を制限します。minPoolSize プロパティを以下のように指定します。

int

整数は 0 ~ 2147483647、ただし、maxPoolSize より小さい値にすることができます。デフォルトは 0 です。負の値は、0 のように扱われます。maxPoolSize より大きい値は、maxPoolSize 値に変更されます。この値は、maxPoolSize が -1 であるときは無視されます。値は、次のとおり接続プールの使用を決定します。

- フリー プール内の物理的な接続の数が、minPoolSize 値に到達すると、JDBC/MX ドライバーは、フリー プールに追加することではなく、物理的に閉じることにより、その後の接続を閉じます。
- 0 は、接続が物理的に閉じられていないことを意味します。接続は、接続がクローズされるとき、常に、フリー プールに追加されます。

maxPoolSize プロパティ

プールが含むことができる物理的な接続の最大数を設定します。これらの接続には、フリー接続と使用中の接続の両方が含まれます。物理的な接続の最大数に達すると、JDBC/MX ドライバーは、SQLException をメッセージ「最大プールサイズに到達しました」と共にスローします。maxPoolSize プロパティを以下のように指定します。

int

整数は、-1、0 ～ 2147483647、ただし、minPoolSize より大きい値にすることができます。デフォルトは -1 (接続プーリングを無効にする) です。値は、次のとおり接続プールの使用を決定します。

- 負の値は -1 と見なされます。
- 0 は、最大プール サイズがないことを示します。
- -1 の値は、接続プーリングを無効にします。

initialPoolSize プロパティ

InitialPoolSize プロパティは、接続プーリングを JDBC/MX ドライバーと共に使用するときに、初期接続プールサイズを設定します。アプリケーションが初めて接続を要求するとき、ドライバーが、各接続プールに対して、initialPoolSize プロパティの値を割り当てられるようにします。ドライバーは、割り当てられた接続の数を作成およびプールしようとします。例えば、initialPoolSize が、データソースに対して

5 に設定されている場合、ドライバーは、アプリケーションが、データソースの getConnection() メソッドを初めてコールするとき、5 つの接続を作成およびプールしようとします。

JDBC T2 ドライバーが、認証に対して有効化されている場合、ユーザーが作成する初めての接続は、プールにおける initialpoolsize 接続数を所有します。

- データタイプ - int
- 単位 - 物理的な接続の数
- デフォルト - -1 (最初の接続プールが作成されていない場合)
- 範囲 - -1〜 maxPoolSize
- 負の値は、-1 として扱われます。

minPoolSize 以下で、maxPoolSize を超えない値を割り当てられます。指定された値が maxPoolSize を超える場合、maxPoolSize プロパティの値が使用されます。

maxIdletime プロパティ

maxIdleTime プロパティは、接続が閉じられる前に、プールにおいて、未使用の物理的な接続の残り秒数を表します。0 (ゼロ) は、制限なしを示します。

次の条件が適用されます。

- maxPoolSize >= 0 かつ minPoolSize = 0 の場合、maxIdleTime は適用されません。
- maxPoolSize > 0、minPoolSize > 0、
  - maxIdleTime プロパティを使用して指定された秒数の後、プール内の接続数は、minPoolSize と等しくなり、余分な接続は、ハードクローズドされます。
  - maxIdleTime = 0 (デフォルト)、次に、接続は、プール サイズが minPoolSize に等しくなるまで、直ちにフリー プールに送られ、接続は閉じられ、その後、ハードクローズドされます。

- データタイプ - int
- 単位 - 秒
- デフォルト - 0 (タイムアウトなし。)
- 範囲 - 0 〜 2147483647
- 負の値は、0 として扱われます。

300 を maxIdletime プロパティに割り当てるには、以下を入力します。

-Djdbcmx.maxIdleTime=300

言語プロパティ

言語プロパティは、エラーメッセージで使用する言語を設定します。このプロパティに関する詳細情報については、**国際化機能**(82 ページ) を参照してください。

- データタイプ — 文字列
- デフォルト — なし
- 値 — `java.io` および `java.lang` について、 正規名で説明される有効な Java の正式名

  Oracle のドキュメントの API 列: **http://download.oracle.com/javase/1.5.0/docs/guide/intl/encoding.doc.html**

言語を shift-JIS に設定するには、以下を入力します。

```
Japanese: language=SJIS
```

transactionMode プロパティ

`transactionMode` プロパティは、トランザクションが実行される方法と制御を提供します。`transactionMode` プロパティは、以下のように指定されます。

文字列

デフォルトは `mixed` です。指定できる値は、次のとおりです。

`internal`

トランザクションが常に JDBC/MX ドライバーで管理されるトランザクション内で実行されることを指定します。内部トランザクションモードが有効な場合に、外部トランザクションが存在する場合、外部トランザクションがサスペンドされ、JDBC/MX ドライバーで管理されるトランザクション内で SQL 文が実行されます。ドライバーの内部で管理されるトランザクションが完了すると、既存の外部トランザクションが再開されます。内部トランザクションモードの時には、Connection 自動コミットフラグは真の値を保持します。

注記: 外部トランザクションで実行される SELECT 文に `internal` transactionMode を使用すると、JDBC/MX は、「無効なトランザクションの状態」例外をスローします。したがって、これらの状況下では、`internal` transactionMode を指定しないでください。

`mixed` (混在)

ドライバーが、現在のスレッドで任意のアクティブなトランザクションを継承することを指定します。トランザクションの自動コミット設定は無視されます。アプリケーションは、このモードで、トランザクションをコミットまたはロールバックしなければなりません。アクティブなトランザクションがない場合、ドライバーは、トランザクションを作成し、トランザクションを開始します。または、SQL エラーがある場合、トランザクションを中止します。このモードでは、ドライバーは、自動コミットと非自動コミットの両方をサポートします。アプリケーションは、非自動コミットモードでは、トランザクションを終了します。

`external` (外部)

外部トランザクションが存在する場合、トランザクションを外部トランザクション内で実行することを指定します。外部トランザクションが存在しない場合は、SQL 文は、トランザクションなしで実行されます。これにより、実行される既存のトランザクションを必要としない SQL 文を実行し、性能改善を行うことができます。SQL コマンドがトランザクションを必要とし、外部トランザクションが存在しない場合、SQL 例外がスローされます。

注記: SQL 文に対して、`external` transactionMode を使用すると、SQL 例外が発生します。したがって、これらの状況下では、`external` transactionMode を指定しないでください。

留意事項:

- その他の任意の文字列がトランザクション モードの値に指定された場合、`mixed` が使用されます。
- 外部または混合トランザクションモードを使用すると、性能が改善される場合があります。

- 負荷の高い TMF トランザクションのオーバーヘッドにより、内部トランザクション モードは、アプリケーションの性能に影響を与える可能性があります。
- このプロパティは、DataSource オブジェクト内で定義される、または、java コマンドラインを通して渡される JDBC/MX ドライバー プロパティ ファイル内で設定できます。
- トランザクションモードは、新しい接続に対してのみ変更できます。したがって、接続内で動的に変更できません。

このプロパティは、データソース、JDBC/MX プロパティファイルまたは java コマンドラインで指定できます。

queryExecuteTime プロパティ

queryExecuteTime プロパティを使用して、クエリ実行時間をミリ秒単位で指定できます。指定された時間を超えるすべてのクエリは、ログファイルに記録されます。このプロパティを構成する場合は、個別のログファイルが作成されます。このプロパティを DataSource オブジェクト、ConnectionPoolDataSource オブジェクト、DriverManager オブジェクト上に設定しなければなりません。。

queryExecuteTime プロパティを構成するときは、次の設定を考慮しなければなりません。

- データタイプ — long
- 単位 — ミリ秒
- デフォルト — 0
- 範囲 — 1 ～ 9,223,372,036,854,775,807 (2**63-1))

次の例では、queryExecuteTime の値は、10 ミリ秒に設定されます。

```
java -Djdbcmx.queryExecuteTime=10
```

ログファイルの構成に関する情報については、**T2QueryExecuteLogFile プロパティ**(35 ページ) を参照してください。

プロパティの設定に関する情報については、**コマンドラインでプロパティの設定**(36 ページ) を参照してください。

T2QueryExecuteLogFile プロパティ

T2QueryExecuteLogFile プロパティを使用して、ログファイルの名前を設定できます。DataSource オブジェクト、ConnectionPoolDataSource オブジェクト、DriverManager オブジェクトで、T2QueryExecuteLogFile プロパティを設定しなければなりません。

T2QueryExecuteLogFile プロパティを構成するときは、次の設定を考慮しなければなりません。。

- データタイプ - 文字列
- ファイル名 - OSS のファイル名。ログファイルのフルパスまたはログファイルの名前のみを割り当てることができます。以下は、ログファイル名の例です。
  - -Djdbcmx.T2QueryExecuteLogFile=/home/t2query.log
  - -Djdbcmx.T2QueryExecuteLogFile=t2query.log

- デフォルトのログ ファイル名 - t2sqlmxQueryExecuteTime.log
- ファイルの場所 - ログファイルが、指定した位置に作成されます。ログファイル名が現在の作業ディレクトリにない場合は、フル パスを指定する必要があります。例えば、作業ディレクトリが、/home/usr で、-Djdbcmx.T2QueryExecuteLogFile プロパティの値が、t2query.log である場合、t2query.log ファイルは、/home/usr に作成されます。しかし、-Djdbcmx.T2QueryExecuteLogFile プロパティの値が、/home/t2query.log である場合、t2query.log ファイルは、現在の作業ディレクトに関係なく、/home に作成されます。

ログファイルを構成する場合、そのファイル名を使って、FileHandler が確立するたびに上書きされます。

プロパティの設定に関する詳細情報については、**コマンドラインでプロパティの設定**(36 ページ) を参照してください。

以下は、T2QueryExecuteLogFile のサンプル内容です。

```
jdbcTrace:[05/02/12 01:05:24]:TRACING JDBC/MX VERSION: T1275R32_30AUG2012_JDBCMX_1209
STMT1:select c2 from tab1: TIME TAKEN 9 ms
STMT2:select c2 from tab1: TIME TAKEN 4 ms
STMT3:select * from tab1: TIME TAKEN 15 ms
```

注記: 実行されているパラメーターの値は表示されません。

コマンドラインでプロパティの設定

java- D オプションのコマンドラインで使用される JDBC/MX ドライバー プロパティ名には、プレフィックス

jdbcmx を含めなければなりません。

この表記法には、ピリオド (.) が含まれており、すべての JDBC/MX ドライバー プロパティ名は、Java アプリケーションに対してユニークであることが保証されます。例: maxStatements プロパティは、

jdbcmx.maxStatements になります

表 4: コマンドラインで許可される JDBC/MX ドライバー プロパティ

JDBC/MX プレフィックス	プロパティ名	説明
jdbcmx.	authenticationMode	認証を有効にするには、このプロパティを ON に設定します。デフォルト値は OFF です。
jdbcmx.	contBatchOnError	任意の BatchUpdateExceptions した後でも、バッチで残りのジョブを続行するために JDBC ドライバーと通信します。**contBatchOnError プロパティ**(47 ページ) を参照してください。
jdbcmx.	stmtatomicity	文のレベルでの SQL 文の原子性を有効化することができます。**stmtatomicity プロパティ**(48 ページ) を参照してください。
jdbcmx.	batchBinding	文とまとめて executeBatch() 操作でバッチ処理することを指定します。**Prepared Statement のバッチ処理設定**(49 ページ) を参照してください。

表は続く

JDBC/MX プレフィックス	プロパティ名	説明
jdbcmx.	blobTableName	BLOB 列に使用する、LOB テーブルを指定します。**LOB テーブル名のプロパティ**(30 ページ) を参照してください。
jdbcmx.	catalog（カタログ）	デフォルト カタログを設定します。**デフォルトのカタログおよびスキーマ**(30 ページ) を参照してください。
jdbcmx.	clobTableName	CLOB 列に使用する、LOB テーブルを指定します。**LOB テーブル名のプロパティ**(30 ページ) を参照してください。
jdbcmx.	enableLog	SQL 文の ID および対応する JDBC SQL 文のロギングを有効化します。**enableLog プロパティ**(95 ページ) を参照してください。
jdbcmx.	idMapFile	トレース機能が、SQL 文 ID および対応する JDBC SQL 文を記録するファイルを指定します。**idMapFile プロパティ**(95 ページ) を参照してください。
jdbcmx.	ISO88591	IS088591 列に格納されたデータにアクセスしたり、書き込んだりするときに使用するエンコーディングを指定します。**ISO88591 プロパティ**(31 ページ) を参照してください。
jdbcmx.	maxPoolSize	最大プールサイズを設定します。**maxPoolSize プロパティ**(32 ページ) を参照してください。
jdbcmx.	initialPoolSize	JDBC/MX ドライバーと共に接続プーリングを使用する場合、初期接続プーリング サイズを設定します。**initialPoolSize プロパティ**(32 ページ) を参照してください。
jdbcmx.	maxIdleTime	接続が閉じられる前に、プールにおいて、未使用の物理的な接続の残り秒数を表します。**maxIdletime プロパティ**(33 ページ) を参照してください。
jdbcmx.	language	エラーメッセージで使用される言語を設定します。**言語プロパティ**(33 ページ) を参照してください。
jdbcmx.	maxStatements	接続プールがキャッシュする PreparedStatement オブジェクトの合計数を設定します。**maxStatements プロパティ**(32 ページ) を参照してください。
jdbcmx.	minPoolSize	最小プールサイズを設定します。**minPoolSize プロパティ**(32 ページ) を参照してください。

表は続く

JDBC/MX プレフィックス	プロパティ名	説明
jdbcmx.	mploc	SQL/MP テーブルの位置を設定します。**mploc プロパティ**(31 ページ) を参照してください。
jdbcmx.	reserveDataLocators	予約されるデータ ロケーターの数を設定します。**reserveDataLocators プロパティの設定**(50 ページ) を参照してください。
jdbcmx.	schema	デフォルト スキーマを設定します。**デフォルトのカタログおよびスキーマ**(30 ページ) を参照してください。
jdbcmx.	sqlmx_nowait	**非ブロック JDBC/MX の管理**(48 ページ) を参照してください。
jdbcmx.	traceFile	ログ用トレースファイルを指定します。**アプリケーションサーバーに対するトレースの有効化**(92 ページ) を参照してください。
jdbcmx.	traceFlag	ログ用トレースフラグを設定します。**アプリケーションサーバーに対するトレースの有効化**(92 ページ) を参照してください。
jdbcmx.	transactionMode	トランザクションが実行される方法と制御を提供するトランザクション モードを設定します。**transactionMode プロパティ**(34 ページ) を参照してください。
jdbcmx.	user	認証用ユーザーを設定します。
jdbcmx.	password	認証用パスワードを設定します。

例えば、OSS 環境で mploc プロパティを使用して、次の形式のいずれかのプレフィックスを含む **mploc** プロパティを指定します。

```
-Djdbcmx.mploc=[\\node.]\$volume.subvolume
```

または

```
-Djdbcmx.mploc='[\node.]$volume.subvolume'
```

トランザクション

JDBC/MX ドライバーは、データの完全性および整合性を維持するためにトランザクション サポートを提供します。アプリケーションが SQL/MX オブジェクトと従来のファイルシステムの間のトランザクションをインターリーブすることができるように、JDBC/MX ドライバーは、SQL/MX との対話が必要なときに、トランザクションがアクティブであるかどうかを確認します。

transactionMode プロパティ(34 ページ) は、トランザクション処理の動作を決定します。通常の環境で、デフォルト値 `mixed` と共に transactionMode を使用する場合、

- アクティブなトランザクションが存在する場合は、自動コミット設定は無視され、JDBC/MX ドライバーにより、アプリケーションが、トランザクションを管理します。
- アクティブなトランザクションが存在しない場合、JDBC/MX ドライバーが、トランザクションを管理します。

このインプリメンテーションは、JDBC/MP とは異なります。JDBC/MP ドライバーでは、URL の 2 つの異なるタイプが、トランザクションを管理するコンポーネントを決定します。

BLOB および CLOB データにアクセスしている場合は、**Blob および Clob へのアクセスを含むトランザクション**(62 ページ) を参照してください。

自動コミット モードおよびトランザクション境界

JDBC/MX がトランザクションを管理するとき、ドライバーは、新しいトランザクションを開始することを決定します。トランザクションが Connection に関連付けられていない場合、新しいトランザクションが開始されます。Connection に関連付けられているトランザクションがある場合、トランザクションは再開されます。Connection 属性自動コミットは、トランザクションを終了するタイミングを指定します。自動コミットの有効化により、JDBC/MX ドライバーは、次のルールに従ってトランザクションを終了します。

- JDBC/MX ドライバーは、SELECT 文以外の SQL 文における SQL エラーの場合、トランザクションをロールバックします。
- 非 SELECT SQL 文の場合、JDBC/MX ドライバーは、この SQL 文の現在のトランザクションが開始された場合に、トランザクションをコミットします。
- SELECT 文の場合は、JDBC/MX ドライバーは、結果セットを閉じるときにトランザクションをコミットします。
- 同時複数 SELECT 文の場合、JDBC/MX ドライバーは、SELECT 文の結果セットまたはトランザクションを開始した文が閉じられるときのみ、トランザクションをコミットします。

自動コミット モードの無効化

自動コミット モードを無効化すると、アプリケーションは、Connection メソッドをコールし、それぞれ、コミットおよびロールバックすることにより、各トランザクションを明示的にコミットまたはロールバックしなければなりません。SELECT 文以外の SQL 文で SQL エラーが発生すると、SQL/MX は、トランザクションに中断を知らせます。このような場合、トランザクションは、アプリケーションが、トランザクションをコミットまたはロールバックするかどうかに関係なく、ロールバックされます。

自動コミット モードに対するデフォルトは、Connection オブジェクトが作成されるとき有効にします。トランザクションの途中で自動コミットの値が変更されると、現在のトランザクションがコミットされます。setAutoCommit がコールされ、自動コミットに対する値が、現在の値から変更されない場合には、NO-OP として処理されます。

ストアド プロシージャー

SQL/MX は、結果セットを含むストアド プロシージャーを提供しています。これは、Java で記述され、SQL/MX 実行環境下で実行されます。

CALL 文を使用して、SQL/MX で、ストアド プロシージャーを実行できます。JDBC/MX ドライバーにより、ストアド プロシージャー用の標準 JDBC API エスケープ構文を使用することにより、ストアド プロシージャーをコールできます。エスケープ SQL 構文は次のとおりです。

```
{call procedure-name([arg1,arg2, ...])}
```

ここで、**argn は、**1 番目のパラメーター **arg1 から、順番にパラメーターを参照します**。Call 文の非エスケープ構文に関する詳細情報については、最新の **HPE NonStop SQL/MX リリース リファレンスマニュアル**を参照してください。

Java アプリケーションは、JDBC 標準 `CallableStatement` インターフェイスを使用して、CALL 文を使用することにより、SQL/MX で、ストアド プロシージャーを実行できます。詳細情報については、**「SQL/MX Guide to Stored Procedures in Java」**を参照してください。

制限事項

Java (SPJs) におけるストアド プロシージャーの制限事項は次のとおりです。

- Java (SPJs) におけるストアド プロシージャーは、CLOB または BLOB データタイプを含む Java メソッドからの返される結果セットをサポートしていません。
- SPJs は、SHORTANSI 名をサポートしていません。

注記:

SHORTANSI 名タイプを SPJs と共に使用しないでください。

SQL コンテキスト管理

NonStop SQL/MX により、SQL/MX コンテキストを管理できます。SQL/MX エグゼキューターは、以下を含む独自の実行環境を含む SQL/MX エグゼキューターのインスタンスとして見なすことができます。

- CONTROL 文および SET 文による設定情報
- トランザクション
- SQL/MX コンパイラ プロセス (MXCMP)
- 一連の SQL/MX エグゼクティブ サーバープロセス (ESP)
- ユーザーが作成した SQL 文

JDBC/MX ドライバーは、JDBC 接続を SQL/MX コンテキストにマッピングします。したがって、マルチスレッド アプリケーションである JDBC アプリケーションでは、複数の SQL/MX コンパイラ プロセス (MXCMP プロセス) が、アプリケーションに関連付けられています。アプリケーションが、JDBC 接続を取得すると、SQL/MX コンテキストが作成されます。アプリケーションが、明示的または暗黙的に JDBC 接続を閉じると、SQL/MX コンテキストは破棄されます。

次の JDBC 接続属性は、対応する SQL 文を実行することにより、 JDBC/MX ドライバーによって、SQL/MX コンテキストに渡されます。

表 5: SQL/MX コンテキストに渡された接続属性

属性	SQL Statement
`catalog` (カタログ)	`SET CATALOG` **`default-catalog-name`**
`schema`	`SET SCHEMA` **`default-schema-name`**

表は続く

属性	SQL Statement
mploc	SET MPLOC **default-location**
transaction isolation	SET TRANSACTION **isolation-level**

プロセス (JVM プロセス) は、プロセス内で複数の SQL/MX コンテキストを持つことができます。

保持可能カーソル

JDBC/MX ドライバーは、ResultSet の保持性属性をサポートします。JDBC アプリケーションで保持可能カーソルを使用するには、これらのガイドラインに従ってください。

- 保持性属性の次の定数のいずれかを使用します。

  com.tandem.sqlmx.SQLMXResultSet.HOLD_CURSORS_OVER_COMMIT

  アプリケーションが、メソッド Connection.commit または Connection.rollback をコールするとき、HOLD_CURSORS_OVER_COMMIT 定数は、ResultSet オブジェクトが閉じられないことを示します。

  com.tandem.sqlmx.SQLMXResultSet.CLOSE_CURSORS_AT_COMMIT

  アプリケーションが、メソッド Connection.commit または Connection.rollback をコールするとき、CLOSE_CURSORS_AT_COMMIT 定数が、ResultSet オブジェクトが閉じられることを示します。

- コミット操作時に保持可能である ResultSet オブジェクトの場合、ResultSet を生成する SQL 文がストリーム アクセス モード、あるいは、テーブル参照に対する組込更新または削除を有していることを確認します。
- SQLMXConnection オブジェクトにおいて、次のメソッドのいずれかを使用して、コミット時に保持可能カーソルを持つ結果セットを作成します。

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

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

  見本プログラムのデモについては、holdJdbcMx.java プログラムの説明を参照してください。

connection pooling (接続プーリング)

JDBC/MX 接続プーリングのインプリメンテーションを提供します。ここでは、物理データベース接続のキャッシュが、クライアント接続セッションに割り当てられ、データベース アクティビティ用に再利用されます。クライアント セッションを閉じると、物理的な接続が、後続の使用のために、キャッシュに戻されます。このインプリメンテーションは、基本 DataSource オブジェクト インプリメンテーションに対比されます。ここでは、1 対 1 対応が、クライアントの Connection オブジェクトと物理的なデータベース接続の間に存在します。

ご使用のアプリケーションは、次の方法で、接続プーリングを使用できます。

- **アプリケーションサーバーによる接続プーリング**(42 ページ)
- **Basic DataSource API を使用する接続プーリング**(43 ページ)
- **DriverManager クラスを用いる接続プーリング**(44 ページ)

認証基準については、**接続プール認証条件**(44 ページ) を参照してください。

アプリケーションサーバーによる接続プーリング

通常、3 層環境では、アプリケーションサーバーは、接続プーリング コンポーネントを実装します。このコンポーネントを実装する方法は、これらのトピックで説明されています。

- **接続プーリングを使用するアプリケーション サーバーを実装するためのガイドライン**(42 ページ)
- **標準 ConnectionPoolDataSource オブジェクトのプロパティ**(43 ページ)

接続プーリングを使用するアプリケーション サーバーを実装するためのガイドライン

- アプリケーションサーバーがプーリングすることで、`ConnectionPoolDataSource` インターフェイスにより作成される `PooledConnection` オブジェクトのキャッシュを維持します。クライアントが、接続オブジェクトを要求するとき、アプリケーションは、適切な `PooledConnection` オブジェクトを検索します。検索条件および他の方法は、アプリケーションサーバーに固有です。
- アプリケーションサーバーは、`ConnectionEventListener` インターフェイスを実装し、リスナー オブジェクトを `PooledConnection` オブジェクトと共に登録します。JDBC/MX ドライバーは、アプリケーションが、Connection オブジェクトの使用を完了したとき、`connectionClosed` イベントを用いて、リスナー オブジェクトに通知します。その後、接続プーリング コンポーネントは、この `PooledConnection` オブジェクトを将来の要求に対して、再利用できます。また、JDBC/MX ドライバーは、`PooledConnection` オブジェクトが、接続の初期化に失敗したとき、`connectionErrorOccurred` イベントを用いて、リスナー オブジェクトを通知します。アプリケーションサーバーの接続プーリング コンポーネントは、このような接続エラー イベントが発生した場合、`PooledConnection` を廃棄する必要があります。
- アプリケーションサーバーは、`ConnectionPoolDataSource` インターフェイスを実装する `SQLMXConnectionPoolDataSource` を使用することにより、接続プールを管理します。JDBC/MX により提供される getter および setter メソッドを使用して、**標準 ConnectionPoolDataSource オブジェクトのプロパティ**(43 ページ) に列挙される接続プール構成プロパティを設定します。これらの標準プロパティに加え、`ConnectionPoolDataSource` には、**DataSource インターフェイスを使用する接続**(26 ページ) で説明される JDBC/MX ドライバー固有のプロパティが含まれます。認証が有効化されている場合、ユーザーおよびパスワード情報を `SQLMXConnectionPoolDataSource.getConnection` メソッドで指定しなければなりません。ユーザーおよびパスワード パラメーターなしで、`getConnection` メソッドを呼び出す場合は、ユーザーおよびパスワードを `ConnectionPoolDataSource` オブジェクトに設定するか、デフォルトのユーザーおよびパスワードをコマンドラインから設定してください。詳細情報については、**認証の有効化**(20 ページ) を参照してください。

サードパーティ製接続プールに対する認証基準

Apache DBCP または C3P0 のようなサードパーティ製接続プーリング メカニズムを使用している場合、接続要求中に提供される認証情報は、サードパーティ製接続プーリング メカニズムよって、無視されることがあります。あるいは、接続を取得する際、データソースのデフォルトのユーザーおよびパスワードを使用することがあります。JDBC T2 ドライバーは、最初の物理接続の作成要求に対してのみ認証情報を認証します。

接続要求中に提供される認証情報は、サードパーティ製接続プーリング メカニズムよって、無視されることがあります。あるいは、接続を取得する際、データソースのデフォルトのユーザーおよびパスワードを使用す

ることがあります。JDBC T2 ドライバーは、最初の物理接続の作成要求に対してのみ、サードパーティ製接続プールによって認証情報を認証します。

サードパーティ製接続プーリング メカニズムは、接続再利用要求用ユーザー認証情報を考慮または無視する場合があります。サードパーティ製プーリング メカニズムのインプリメンテーションにより、未認可ユーザーとしてデータベースにアクセスすることができる潜在的なセキュリティ上の影響が存在する場合があります。

注記:

JDBC T2 ドライバー認証を有効化する前に、サードパーティ製プーリング メカニズムに関するドキュメントを読み、セキュリティ上の影響を理解してください。

標準 ConnectionPoolDataSource オブジェクトのプロパティ

注記: アプリケーションサーバーでは、これらのプロパティの意味を定義します。

表 6: 標準 ConnectionPoolDataSource オブジェクトのプロパティ

プロパティ名	タイプ	説明
maxStatements	**int**	プールがキャッシュを行う必要がある PreparedStatement オブジェクトの総数。この総数には、フリー オブジェクトのおよび使用中のオブジェクトが含まれます。0 (ゼロ) は、ステートメント プーリングを無効化します。
initialPoolSize	**int**	作成されるとき、プールが含む必要がある物理的な接続の数。
minPoolSize	**int**	プールが常時使用可能に維持する必要がある物理的な接続の数。0 (ゼロ) は、最大サイズがないことを示します。
maxPoolSize	**int**	プールが含む必要がある物理的な接続の最大数。0 (ゼロ) は、最大サイズがないことを示します。
maxIdleTime	**int**	接続が閉じられる前に、プールにおいて、未使用の物理的な接続の残り秒数。0 (ゼロ) は、制限なしを示します。
propertyCycle	**int**	プールが上記の接続プール プロパティの値により定義されている現在のポリシーを強制する前に待機する秒単位の間隔。

Basic DataSource API を使用する接続プーリング

JDBC アプリケーションが接続プーリングを有効化するために、基本 DataSource インターフェイスを使用します。これには、接続プーリングを制御する次のプロパティが含まれます。

- maxPoolSize
- minPoolSize
- maxStatements

アプリケーションは、次の 2 つの方法で、接続プーリングを有効化できます。

- 基本 DataSource オブジェクトの dataSourceName プロパティを以前に登録した ConnectionPoolDataSource オブジェクトに設定します。接続プーリングが有効な場合、

ConnectionPoolDataSource オブジェクトの JDBC/MX ドライバー固有プロパティは有効です。そして、DataSource オブジェクトの JDBC/MX ドライバー固有プロパティは無視されます。JDBC/MX ドライバー固有プロパティを含む接続が初期化されるとき、PooledConnection が取得されます。

- dataSourceName プロパティが空の場合、DataSource オブジェクトのプロパティを用します。接続プーリングは、デフォルトでは無効です。maxPoolSize プロパティに対するデフォルト値が -1 であることに注意してください。これは、接続プーリングを無効化します。これらのプロパティの使用に関する詳細については、DataSource インターフェイスを参照してください。

接続プーリング用トラブルシューティング アプリケーションについて、機能を実装する方法に関する次の詳細に注意してください。JDBC/MX は、最初の使用可能な PooledConnection オブジェクトを検索し、クライアントの接続要求に対して、オブジェクトを割り当てます。JDBC/MX は、すべての接続の SQL/MX 実行環境およびコンパイル環境が、接続プーリング環境と同じであること、つまり、初期接続がクライアント セッションにより、プールまたは新しい物理接続から取得されたときと同じであることを確認します。

DriverManager クラスを用いる接続プーリング

接続プーリングは、JDBC アプリケーションが接続用に DriverManager クラスを使用する場合、デフォルトで使用可能です。DriverManager オブジェクトのプロパティ テーブルに列挙される次のプロパティおよび JDBC/MX プロパティで説明される次のプロパティを使用することにより、接続プーリングを管理できます。

- maxPoolSize
- minPoolSize
- maxStatements

2 つの方法のいずれかでこれらのプロパティを設定します。

- コマンドラインで、オプション -D**property_name=property_value** を使用します。
- DriverManager クラスの getConnection() メソッドの java.util.Properties パラメーターを使用します。

DriverManager クラスを含む接続プーリングのプロパティを設定する場合、これらのガイドラインを使用します。

- 接続プーリングを有効化するには、maxPoolSize プロパティを 0 (ゼロ) より大きい整数値に設定します。
- Properties パラメーターを通して渡されるプロパティは、コマンドライン プロパティよりも高い優先順位を持ちます。
- 同じカタログ スキーマの組み合わせとの接続はプールにまとめ、MX/JDBC ドライバーによって管理されます。指定されたカタログ スキーマの組み合わせについて、最初の接続を取得する場合、アプリケーション プロセスを使用して接続プーリング プロパティの値は、アプリケーション プロセスのどの段階を組み合わせた効果的です。

接続プール認証条件

認証が有効な場合、ドライバーは、接続を認証し、接続は、接続要求で指定されているユーザーによって所有されます。認証が有効でない場合は、接続は、プロセス オーナー (アプリケーションを起動したユーザー) によって所有されます。接続プールは、認証モードを有効にした状態および認証なしで作成された一連の接続を含むことができます。

接続が要求され、プールからの接続が要求されたとき認証モードが有効である場合、ドライバーは、以下を返します。

- ドライバーは、接続要求時に提供されたユーザー認証情報を認証します。認証に失敗する場合、ドライバーは、ユーザー認証例外を返します。
- 認証に成功すると、ドライバーは、プールからユーザーに属している接続を返します。
- プールにユーザーに属している接続がなく、最大接続数に達していない場合、ドライバーは、ユーザー用の新しい接続を作成します。
- プールにユーザーに属している接続がなく、最大接続数に達した場合、ドライバーは、フリー プールからの接続を閉じ、ユーザー用の新しい接続を作成します。

認証が有効でない場合、ドライバーは、接続を認証しません。接続は認証せずに作成され、プロセス オーナー (アプリケーションを起動するユーザー) によって所有されます。接続が要求され、プールからの接続が要求されたとき認証モードが無効である場合、ドライバーは、以下を返します。

- アプリケーションを起動したユーザーに属する既存の接続がフリー プール内にある場合、ドライバーは、この接続を返します。
- プールにユーザー用の接続がなく、最大プール サイズ制限に達していない場合、ドライバーは、新しい接続を作成します。
- ユーザー用の接続がなく、最大プール サイズ制限に達した場合、ドライバーは、フリー プールからの接続を閉じ、ユーザー用の新しい接続を作成します。回収済み接続は、プロセス オーナーまたは別のユーザーのいずれかに属することができます。

ステートメント プーリング

ステートメント プーリング機能により、アプリケーションは、接続プーリング環境において接続を再利用するとの同じ方法で、`PreparedStatement` オブジェクトを再利用できます。ステートメント プーリングは、アプリケーションに対して、完全に透過的に行われます。ステートメント プーリングの使用について、次のトピックで説明します。

- **ステートメント プーリングのためのガイドライン**(45 ページ)
- **ResultSet 処理の性能を制御する**(46 ページ)
- **ステートメント プーリングのトラブルシューティング**(46 ページ)

ステートメント プーリングのためのガイドライン

- `DataSource` オブジェクトの `maxStatements` プロパティを 0 より大きい整数値に設定し、合わせて接続プーリングを有効化することにより、ステートメント プーリングを有効化します。詳細情報については、**connection pooling (接続プーリング)**(41 ページ) を参照してください。
- JDBC アプリケーションに対してステートメント プーリングを有効化すると、性能が劇的に向上する可能性があります。
- 範囲内にない `PreparedStatement` オブジェクトは、アプリケーションが明示的に閉じない限り再利用しないので、`Statement.close` メソッドを使用することにより、PreparedStatement を明示的に閉じてください。
- ご使用のアプリケーションが、`PreparedStatement` を再利用することを確認するため、次のいずれかをコールします。

- Statement.close メソッド — アプリケーションによりコールされます。
- Connection.close メソッド — アプリケーションによりコールされます。使用されていた、すべての PreparedStatement オブジェクトは、接続が再利用されるときに、再利用される準備が整っています。

ResultSet 処理の性能を制御する

3 つ以上の行を返すことが期待される文に対する結果フェッチ JDBC アプリケーションの性能を改善するには、アプリケーションは、文を実行する前にフェッチ サイズを設定することを検討すべきです。ResultSet getter メソッドが、JDBC/MX ドライバー内のデータベースの相互作用を最適化するために、この操作は動作します。JDBC/MX ドライバーは、フェッチ サイズ設定を使用して、データを読み取りまたはバッファーするために使用されるメモリのサイズを決定します。

アプリケーションは、Statement クラス、PreparedStatement クラス、ResultSet クラスの setFetchSize() メソッドを使用することにより、ResultSet のフェッチ サイズを制御できます。

留意事項:

- SQL/MP テーブルではなく、SQL/MX テーブルを使用するアプリケーションは、3 つ以上の行が返される結果フェッチの場合のみ取得の性能が向上します。デフォルトの JDBC/MX フェッチ サイズは 1 に設定されます。
- アプリケーションが、文に対して 3 つ以上の値をフェッチ サイズを設定した場合、アプリケーションが、2 つ以下の値にリセットしないようにしてください。アプリケーションがこれを行うと、デフォルト値を使用する場合に比べて、性能が若干低下します。
- 3 つ以上のフェッチ サイズを設定し、2 つ以下の行が返されたとき、デフォルト フェッチ サイズに比べて、性能が若干低下します。
- 文によって返される行の数より大きい値をフェッチ サイズに設定すると、JDBC/MX ドライバーがより多くのメモリが使用しますが、API の機能には影響しません。

ステートメント プーリングのトラブルシューティング

原因

ステートメント プーリングをトラブルシューティングする場合は、次の JDBC/MX ドライバー インプリメンテーションの詳細に注意してください。

- JDBC/MX ドライバーは、ステートメント プールで、一致する PreparedStatement オブジェクトを検索し、PreparedStatement を再利用します。マッチング基準には、SQL 文字列、現在のカタログ、現在のスキーマ、現在のトランザクションアイソレーション、 resultSetHoldability が含まれます。JDBC/MX ドライバーが、一致する PreparedStatement オブジェクトを検索した場合、JDBC/MX ドライバーは、同じ preparedStatement オブジェクトを再利用のためにアプリケーションに返し、PreparedStatement オブジェクトを使用中としてマーク付けします。
- 文の数が maxStatements 制限に到達すると、後続の生成されるキャッシュ PreparedStatement オブジェクト用のスペースを作るため、アルゴリズム「古いものから削除する」が使用されます。
- MX/JDBC ドライバーは、実行または再使用時に有効な任意の SQL CONTROL 文が、SQL/MX コンパイル時点で有効であったものと同じであることを仮定します。この条件が真でない場合、PreparedStatement オブジェクトの再利用は、予期しない動作を招くことがあります。

解決方法

- ステートメント プーリングにより性能改善をするため SQL/MX 再コンパイルを避ける必要があります。特定の条件が満たされている場合、SQL/MX エグゼキューターは、クエリを自動的に再コンパイルします。これらの条件の一部は、次のとおりです。

  - テーブルの実行時バージョンには、同じテーブルのコンパイル時バージョンとは異なる再定義タイムスタンプがあります。
  - テーブル上の既存のオープン操作は、DDL または SQL ユーティリティ操作により除去されました。
  - 実行時のトランザクションアイソレーション レベルおよびアクセスモードは、コンパイル時とは異なります。

  SQL/MX 再コンパイルに関する詳細情報については、C および COBOL 用 SQL/MX プログラミング マニュアルまたは Java 用 SQL/MX プログラミング マニュアルを参照してください。
- クエリを再コンパイルすると、SQL/MX エグゼキューターは、再コンパイルされたクエリを保存します。したがって、上記の条件のいずれかが再び満たされるまで、クエリは一度だけ再コンパイルされます。
- JDBC/MX ドライバーは、ステートメント プーリングが、アクティブ化されたとき、`CallableStatement` オブジェクトを `PreparedStatement` オブジェクトと同じ方法でプーリングします。
- JDBC/MX ドライバーは、`Statement` オブジェクトをキャッシュしません。

追加 JDBC/MX プロパティの使用

次のアプリケーション機能用に JDBC/MX プロパティを使用できます。

また、これらのトピックに加えて、**アプリケーションサーバーに対するトレースの有効化**(92 ページ) を参照してください。

BatchUpdate 例外処理の改良

バッチ更新のコマンドが失敗すると、バッチの残りのコマンド実行されず、バッチ全体を再実行の原因となります。しかし、このバッチ更新例外処理サポートを用いて、エラーを起こしやすい文の後のバッチの残りのエレメントを実行し、バッチジョブ全体の再実行を避けることができます。

contBatchOnError プロパティ

ContBatchOnError プロパティは、JDBC ドライバーと通信して、任意の BatchUpdateExceptions 後でも、バッチで残りのジョブを続行するようにします。この java プロパティは、以下のように、コマンドラインから設定できます。

```
-Djdbcmx.contBatchOnError={ON|OFF}
```

ここで

```
ON
```

その他のバッチ例外後でも、バッチの実行を継続します。

OFF

その他のバッチ例外後、バッチの実行を終了します。デフォルトでは、OFF に設定されています。

注記:

このプロパティは、java コマンドライン オプションを通して、または、データ ソースのプロパティ ファイルを通して設定できます。

文レベルの原子性

データベースの整合性を維持するために、トランザクションを制御して、処理を正常に完了させるか、中断させなければなりません。JDBC/MX の以前バージョン (H10 AAB 以前) を使用すると、SQL 文実行中にエラーが発生すると、トランザクションは自動的に中断されます。

JDBC/MX ドライバーのこのバージョンは、SQL/MX 3.0 または以降のバージョンをフォローします。文の原子性により、トランザクション内の個々の SQL 文が正常に完了するか、データベースには影響を及ぼさないことが保証されます。この文レベルの原子性が、自動コミットモードを偽に設定することによりフォローされると、挿入、更新、削除操作中に発生する障害は、現在のトランザクションを中断しません。これにより、この現在のトランザクション下のすべての文が、トランザクションのコミットまたはロールバックが発行されるまで実行されます。この機能は、オプションであり、「stmtatomicity」 システムプロパティを設定することにより、有効化できます。

stmtatomicity プロパティ

Stmtatomicity プロパティを有効化することにより、JDBC ドライバーは、文レベルでトランザクションの原子性を設定できます。

この java プロパティは、以下のようにコマンドラインから設定できます。

```
-Djdbcmx.stmtatomicity={ON|OFF}
```

ここで

ON

文レベルの原子性

OFF

トランザクション レベルの原子性です。デフォルトでは、OFF に設定されています。

注記:

この機能は、JDBC/MX T1275H10^AAB 以降のバージョンで利用可能です。

非ブロック JDBC/MX の管理

ブロックモードで JDBC/MX ドライバーを用いることにより、SQL 操作が発生するとき、JVM プロセス全体をブロックします。非ブロック モードにより、JDBC/MX ドライバーは、JVM プロセス全体ではなく、SQL 操作を実行するスレッドのみをブロックします。マルチスレッド Java アプリケーションでは、非ブロック JDBC/MX 機能により、JVM は、他のスレッドを同時にスケジューリングできます。一方、各 SQL 操作は、スレッドにより実行されます。

デフォルトでは、JDBC/MX は、非ブロック モードを使用します。コマンドラインで、-Djdbcmx.sqlmx_nowait オプションを使用することにより、sqlmx_nowait プロパティを OFF に設定し、Java アプリケーションで、非ブロック JDBC/MX を無効化できます。構文は次のとおりです。

```
-Djdbcmx.sqlmx_nowait= {ON |OFF}
```

ここで

ON

は、非ブロック JDBC/MX を指定します。デフォルトは ON です。

OFF

は、JDBC/MX がプロセス全体をブロックすることを指定します。

また、プログラム内で、sqlmx_nowait プロパティを設定することにより、非ブロック JDBC/MX をプログラム的に無効化または有効化できます。アプリケーションに応じて、このプロパティを次のよう設定します。

- DriverManager クラスを使用して、JDBC 接続を取得する JDBC/MX アプリケーションでは、JDBC/MX ドライバーをロードする前に、このプロパティを設定します。
- JNDI API を DataSource インターフェイスと共に使用して、JDBC 接続を取得する JDBC/MX アプリケーションでは、DataSource オブジェクトを作成する前に、このプロパティを設定します。

JDBC 接続が、複数のスレッドから、同時に使用できるようになりました。SQL 文で動作している複数のスレッドは、同じ接続を共有できます。したがって、1 つの接続コンテキストが、複数のスレッド間で使用され、接続オブジェクトに関連する操作は、スレッドセーフになります。

あなたが、非ブロック JDBC を使用するマルチスレッド Java アプリケーションを作成するアプリケーション開発者である場合は、これらの推奨に従ってください。

- 1 つのスレッドあたり 1 つだけの JDBC 接続を作成します。シングル スレッドで複数の JDBC 接続を取得するアプリケーションは、SQL/MX 操作を同時に実行しません。そして、各接続は、独自の SQL/MX コンパイラ プロセスを要求するので、システムリソースを浪費することがあります。
- Cancel() メソッドを用いて、SQL 操作をキャンセルする以外の目的で、Statement または ResultSet オブジェクトなどの JDBC Java オブジェクトをスレッド間で共有しないでください。
- NonStop Server for Java 4 でのスレッド インプリメンテーションのノンプリエンプティブ ネイチャーに注意してください。CPU バウンド スレッドは、スレッド スケジューラに異なるスレッドをスケジュールする機会を提供することなく、その完了まで実行します。
- 複数のスレッド間で接続を共有するアプリケーションを作成する場合は、接続プロパティを変更しないようにしてください。

Prepared Statement のバッチ処理設定

batchBinding プロパティを設定することにより、PreparedStatement.executeBatch() メソッドを使用している場合、バッチ処理の性能を改善できます。batchBinding プロパティが設定されているとき、executeBatch() 操作で、文がバッチ処理されます。

JDBC アプリケーションが、batchBinding プロパティを設定すると、JDBC/MX ドライバーは、指定されたバインディング サイズに応じて、リソースを割り当てます。

batchBinding サイズを設定するには、batchBinding プロパティをコマンドラインで指定します。構文は次のとおりです。

-Djdbcmx.batchBinding=**binding_size**

ここで、**binding_size** は、JDBC/MX ドライバーが実行用に一緒にバインドできる PreparedStatement.executeBatch() メソッドの最大数を指定する正の符号付きロング整数です。整数値は、0 から 2 ギガバイトの範囲で指定できます。

留意事項

- **binding_size** に対して許容される値により、アプリケーションのメモリ不足が発生する可能性があります。**binding_size** をメモリ制限に適切なサイズに設定していることを確認します。
- 文の数がバインディング サイズより大きい場合、JDBC/MX ドライバーは、バインディング サイズに基づくブロック サイズに文の実行を分割します。
- JDBC アプリケーションが、バッチ実行をコールしない場合、jdbcmx.batchBinding プロパティを設定することにより、指定されたバインディング サイズに応じて、データベース リソースが割り当てられます。
- jdbcmx.batchBinding プロパティが設定されていないとき、PreparedStatement.executeBatch() メソッドは、配列内の各項目の対応する文により影響を受ける行数を含む行数配列を返します。デフォルトでは、JDBC/MX ドライバーは、行数配列を返すことによってバッチ処理を実行します。
- jdbcmx.batchBinding プロパティが設定されているとき、前項で説明した詳細情報は利用できません。文の実行が成功すると、行数項目は、JDBC 3.0 仕様に準拠する Statement.SUCCESS_NO_INFO に設定されます。PreparedStatement.getUpdateCount() メソッドは、PreparedStatement.executeBatch() メソッドにより実行されるすべての文により影響を受ける総行数を返します。

reserveDataLocators プロパティの設定

reserveDataLocators プロパティは、LOB テーブルにデータを格納するためのプロセスに予約されているデータ ロケーターの数を設定します。予約される データ ロケーターのデフォルト値は 100 です。プロパティの形式は、次のとおりです。

```
jdbcmx.reserveDataLocators=n
```

ここで、**n** は、予約されるデータ ロケーターの数の整数値です。実際に必要とされるデータ ロケーターの数より大きな値を設定しないでください。データ ロケーターに関する詳細情報については、**データ ロケーターの予約**(54 ページ) を参照してください。

JDBC アプリケーションに対するこの値を変更するには、コマンドラインから、このプロパティを指定します。例えば、次のコマンドは、プログラム myProgramClass 用に 150 のデータ ロケーターを予約します。

```
java -Djbcmx.reserveDataLocators=150 myProgramClass
```

サポートされるキャラクターセット エンコーディング

SQL リテラルがプログラムに存在しない場合、JDBC/MX ドライバーを使用する Java アプリケーションは、Java file.encoding プロパティを指定して、デフォルト エンコーディングを Java によりサポートされる任意のキャラクターセットに設定できます。プログラムが SQL リテラルを含む場合、プログラムは、サポートされている SQL/MX セットに対応する Java エンコーディングのみを使用するようにしてください。

JDBC/MX ドライバーは、次の表に列挙される SQL/MX がサポートするキャラクターセットを使用する場合のみ、SQL CHAR、VARCHAR、VARCHAR_LONG、VARCHAR_WITH_LENGTH データタイプの読み取りおよび書き込みをサポートします。

JDBC/MX ドライバーは、特定の SQL データベース列に関連付けられているキャラクターセット名とは独立の、デフォルト エンコーディングの関数として文字列データタイプをエンコードおよびデコードします。

Java file.encoding プロパティの形式は、次のとおりです。

```
-Dfile.encoding=encoding
```

注記: SQL/MX は、NonStop Server for Java 4 によりサポートされるエンコーディング セットのサブセットをサポートしています。

表 7: 対応する SQL/MX キャラクターセットおよび Java エンコーディング セット

SQL/MX キャラクターセット	対応する java.io API 用 Java エンコーディング セットの正式名称	対応する java.io および java.lang API 用 Java エンコーディング セットの正式名称	説明
ISO88591	ISO-8859-1	ISO8859_1	シングルキャラクター、8 ビット、キャラクターデータタイプ用キャラクターセット。これは、英語およびその他の西ヨーロッパ言語をサポートします。
UCS2	UTF-16BE	UnicodeBigUnmarked	2 バイトのエンコードされたユニバーサル キャラクターセット。UTF16 ビッグエンディアンエンコーディングでのダブルキャラクター Unicode キャラクターセット。 **注記:** UCS2 は、SQL/MX テーブルでのみサポートされます。
KANJI	Shift_JIS	SJIS	マルチバイト キャラクターセット。日本で広く使用されています。シングル バイト キャラクターセットおよびダブル バイト キャラクターセットで構成されます。これは、Shift JIS のサブセット (ダブル キャラクター部分) です。そのエンコーディングは、ビッグエンディアンです。
KSC5601	EUC-KR (コードセット 1)	EUC_KR	韓国内の政府や金融機関によって使用されるシステム上で必要なダブルバイト キャラクターセット。そのエンコーディングは、ビッグエンディアンです。 **注記:** KSC5601 は、SQL/MP テーブルのみでサポートされます。

SQL/MX によりサポートされるキャラクターセットに関する完全な情報および SQL/MP テーブルのサポートに関する追加情報については、最新の **HPE NonStop SQL/MX リリース リファレンスマニュアル**を参照してください。

NonStop Server for Java 5 のエンコーディングに関する完全な情報については、**サポートされるエンコーディング** (**http://download.oracle.com/javase/1.5.0/docs/guide/intl/encoding.doc.html**) を参照してください。

BLOB および CLOB データを用いる作業

本セクションでは、JDBC アプリケーションにおける BLOB および CLOB データを用いる作業について説明します。JDBC 3.0 API 仕様に記述される標準インターフェイスを使用して、NonStop SQL/MX テーブル内の BLOB および CLOB データに JDBC/MX ドライバーによって提供されるサポートを用いてアクセスできます。

BLOB および CLOB は、SQL/MX データベースにおけるネイティブデータ タイプではありません。しかし、データベース管理者は、次のセクション **BLOB および CLOB データ用 SQL/MX テーブルの管理**(64 ページ) で説明するように、JDBC/MX ドライバーまたは MXCI の特殊 SQL 構文を使用することにより、BLOB および CLOB 列を含む SQL/MX テーブルを作成できます。

管理のため、CLOB および BLOB データは、ラージ オブジェクト (LOB) と呼ばれます。これは、いずれかのデータタイプを表します。

注記:

BLOB および CLOB データのサポートには、SQL/MX テーブルが必要です。

セクションは、カテゴリ別トピックにおいて、次のように構成されています。

カテゴリ	トピック
物理ファイル	· **LOB サポート用アーキテクチャー**(53 ページ) · **LOB テーブルのプロパティの設定**(53 ページ) · **LOB テーブルの指定**(54 ページ)
CLOB データにアクセスする	· **CLOB データの格納**(54 ページ) · **CLOB データの読み取り**(57 ページ) · **CLOB データの更新**(57 ページ) · **CLOB データの削除**(58 ページ)
BLOB データにアクセスする	· **BLOB データの格納**(58 ページ) · **BLOB 列からのバイナリデータの読み取り**(60 ページ) · **BLOB データの更新**(61 ページ) · **BLOB データの削除**(61 ページ)
Miscellaneous (その他)	· **NULL と空の BLOB または CLOB 値**(61 ページ) · **Blob および Clob へのアクセスを含むトランザクション**(62 ページ) · **Clob および Blob オブジェクト アクセスに関する留意事項**(62 ページ)

BLOB および CLOB データにアクセスする方法を提示する完全な作業例については、付録 A を参照してください。

BLOB および CLOB データ用テーブルの作成および管理に関する情報については、**BLOB および CLOB データ用 SQL/MX テーブルの管理**(64 ページ) を参照してください。

LOB サポート用アーキテクチャー

LOB データをサポートするテーブルは、次のとおりです。

Base table (実テーブル)

BLOB および CLOB データを挿入、格納、読取、更新するために JDBC アプリケーションにより参照されます。実テーブルでは、JDBC/MX ドライバーは、BLOB および CLOB 列データをデータ-ロケーター列にマップします。データ-ロケーター列は、LOB テーブルと呼ばれる個別のユーザー テーブルに格納される実際の LOB データをポイントします。

LOB テーブル

実際に BLOB および CLOB データをチャンクに含みます。Clob または Blob オブジェクトは、データ ロケーターにより識別されます。LOB テーブルには、次の 2 つの形式があります。BLOB データ用の LOB テーブルおよび CLOB データ用の LOB テーブル。

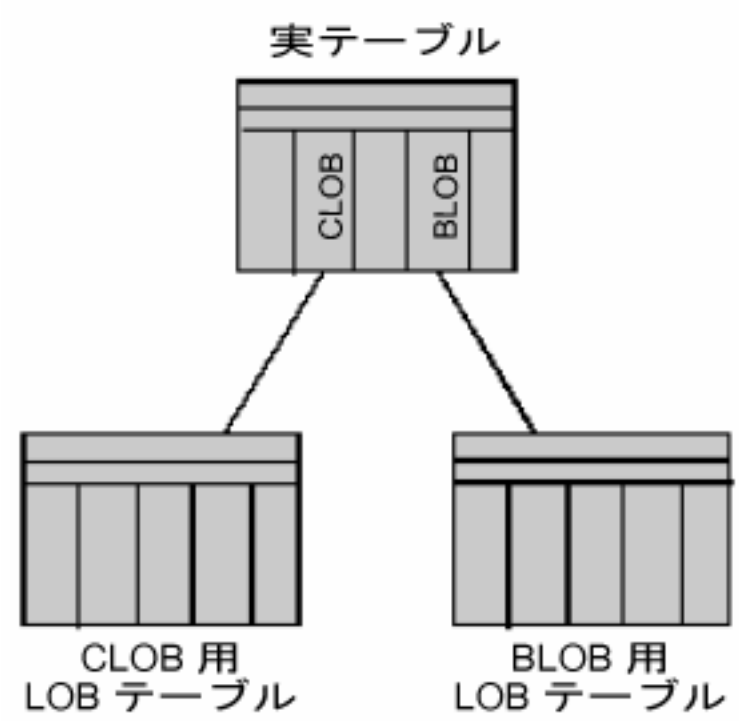

図 2: LOB アーキテクチャー LOB データ サポート用テーブル

LOB テーブルのプロパティの設定

JDBC API を通して BLOB および CLOB データを使用する JDBC アプリケーションを実行する前に、データベース管理者は、LOB テーブルを作成しなければなりません。LOB テーブルの作成に関する情報については、**JDBC/MX Lob Admin ユーティリティを使用して LOB データを管理する**(66 ページ) を参照してください。

BLOB または CLOB データにアクセスする JDBC アプリケーションは、関連する LOB テーブル名を指定して、必要に応じて、reserveDataLocator プロパティを構成しなければなりません。これらの作業については、以下のトピックで説明しています。

LOB テーブルの指定

実行時、JDBC アプリケーションは、JDBC/MX ドライバーにアプリケーションによりアクセスされる実テーブルの CLOB または BLOB 列に関連する LOB テーブルの名前を通知します。1 つの LOB テーブル、または個別のテーブルを BLOB および CLOB データに使用できます。

JDBC アプリケーションは、LOB 列タイプに応じて、次のプロパティの 1 つを使用することにより、システムパラメーターまたは Java Property オブジェクトを通して、LOB テーブル名を指定します。

LOB 列タイプ	プロパティ名
BLOB	blobTableName
CLOB	clobTableName

このプロパティの使用に関する詳細情報については、**LOB テーブル名のプロパティ**(30 ページ) を参照してください。

データ ロケーターの予約

データ ロケーターとは、実テーブル定義において、BLOB または CLOB 列の代わりに用いられるリファレンスポインター値 (SQL LARGEINT データタイプ) です。LOB テーブルに格納される各オブジェクトには、一意のデータ ロケーター値が割り当てられます。LOB テーブルは、特定の LOB テーブルを使用するすべてのアクセサーの間の共有リソースであるため、データ ロケーターを予約して、次の値を取得するための競合を軽減します。予約済みデータ ロケーターのデフォルト設定である 100 を使用して、JVM の各インスタンスは、新しい割り当てを必要とする前に 100 のラージオブジェクト (非チャンク) を挿入できます。

コマンドラインで、JDBC/MX システムプロパティ jdbcmx.reserveDataLocators を使用することにより、使用するアプリケーション用に予約するデータ ロケーターの数 (n) を指定できます。

このプロパティの指定に関する詳細については、**reserveDataLocators プロパティの設定**(50 ページ) を参照してください。

CLOB データの格納

Clob インターフェイスを使用する CLOB 列の挿入

CLOB データタイプを含む行を挿入するとき、列が実際の CLOB データを用いて更新できる前に、「空の」CLOB 値を含む行を挿入できます。INSER 文の CLOB 列 に対して EMPTY_CLOB() 関数を指定することにより、空の CLOB 値を NonStop SQL/MX データベースに挿入できます。

JDBC/MX ドライバーは、EMPTY_CLOB() 関数の SQL 文字列をスキャンし、 次に利用可能なデータ ロケーターで置換します。

次に、更新用に CLOB 列を選択することにより、空の CLOB 列に対するハンドルを取得しなければなりません。

注記:

選択クエリで CLOB 列の名前を変更しないでください。

次のコードは、空の CLOB 列に対するハンドルを取得する方法を示しています。

```
Clob myClob = null;
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("Select myClobColumn
        from myTable where ….for update");
if (rs.next())
        myClob = rs.getClob(1);
```

データを CLOB 列に書き込めるようになりました。**ASCII または Unicode のデータを CLOB 列に 書き込む** (55 ページ) を参照してください。

ASCII または Unicode のデータを CLOB 列に 書き込む

次のように、ASCII または Unicode データを CLOB 列に書き込みます。

- **ASCII データ**(55 ページ)
- **Unicode データ**(55 ページ)

ASCII データ

Clob インターフェイスを使用して、ASCII または Unicode データを CLOB 列に書き込むことができます。次のコードは、Clob インターフェイスの setAsciiStream メソッドを使用して、CLOB データを書き込む例を示します。

```
Clob myClob = null;
// 位置 1 においてストリームを開始
long pos = 1;
// データを含む文字列の例
String s = "TEST_CLOB";for (int i=0; i<5000; i++) s = s + "DATA";
// Clob データを書き込む出力ストリームを取得する
OutputStream os = myClob.setAsciiStream(pos);
// OutputStream を使用して Clob データを書き込む
byte[] myClobData = s.getBytes();
os.write(myClobData);
```

JDBC/MX ドライバーは、出力ストリームをチャンクに分割し、チャンクを LOB テーブルに格納します。

Unicode データ

次のコードは、空の CLOB 列に対するハンドルを取得した後、Unicode データを CLOB 列に書き込む方法を示します。

```
Clob myClob = null;
// 位置 1 においてストリームを開始
long pos = 1;
// Unicode のデータを含む文字列の例
String s ="TEST_UNICODE_DATA";
// Clob データを書き込む出力ストリームを取得する
Writer cw = myClob.setCharacterStream(pos);
```

```
// ライターを使用して Clob データを書き込む
char[] myClobData = s.toCharArray();
cw.write(myClobData);
```

PreparedStatement インターフェイスを使用して CLOB データを挿入する

次のように、PreparedStatement インターフェイスを使用して、データを含む CLOB 列を挿入できます。

- **ASCII データ**(56 ページ)
- **Unicode データ**(56 ページ)

ASCII データ

ASCII または Unicode のデータを含む CLOB 列を FileInputSteam から挿入できます。PreparedStatement インターフェイスを使用して、CLOB 列を挿入しなければなりません。

```
FileInputStream inputAsciiStream = new FileInputStream(myClobTestFile);
int clobLen = inputAsciiStream.available();
PreparedStatement ps = conn.prepareStatement("insert
        into myTable (myClobColumn) values (?)");
ps.setAsciiStream(1, inputAsciiStream, clobLen);
ps.executeUpdate();
```

JDBC/MX ドライバーは、FileInputSteam からデータを読み取り、データを LOB テーブルに書き込みます。JDBC/MX ドライバーは、次に利用可能なデータ ロケーターをテーブルの CLOB 列のパラメーターの代わりに用います。

Unicode データ

Unicode データを含む CLOB 列を FileReader から挿入できます。PreparedStatement インターフェイスを使用して、CLOB 列を挿入しなければなりません。

```
FileReader inputReader = new FileReader(myClobTestFile);
PreparedStatement ps = conn.prepareStatement("insert
        into myTable (myClobColumn) values (?)");
ps.setCharacterStream (1、inputReader、(int)myClobTestFile.length());
ps.executeUpdate();
```

JDBC/MX ドライバーは、データを FileReader から読み取り、データを LOB テーブルに書き込みます。JDBC/MX ドライバーは、次に利用可能なデータ ロケーターをテーブルの CLOB 列のパラメーターの代わりに用います。

SetClob メソッドを使用して Clob オブジェクトを挿入する

JDBC アプリケーションは、直接、Clob オブジェクトをインスタンス化できません。同等の操作を実行するには:

1. ResultSet インターフェイスの getClob メソッドを使用することにより、Clob オブジェクトを取得します。
2. PreparedStatement インターフェイスの setClob メソッドを使用することにより、Clob オブジェクトを別の行に挿入します。

このようなシチュエーションでは、JDBC/MX ドライバーは新しいデータ ロケーターを生成し、PreparedStatement が実行されて、Clob ソースの内容を新しい Clob オブジェクトにコピーします。

CLOB データの読み取り

1 個の CLOB 列から ASCII データを読み取る

Clob インターフェイスまたは InputStream を使用することにより、ASCII または Unicode のデータを CLOB 列から読み取ることができます。

以下のコードは、Clob インターフェイスを使用して、CLOB 列から ASCII データを読み取る方法を説明しています。

```
// ResultSet から Clob を取得する
Clob myClob = rs.getClob("myClobColumn");
// Clob データを読み取るために入力ストリームを取得する
InputStream is = myClob.getAsciiStream();
// Reader を使用して Clob データを読み取る
byte[] myClobData = new byte[length];
int readLen = is.read(myClobData, offset, length);
```

InputStream を使用することにより、ASCII または Unicode のデータを CLOB 列から読み取るには:

```
// InputStream を ResultSet から取得する
 InputStream is = rs.getAsciiStream("myClobColumn");
// Reader を使用して Clob データを読み取る
byte[] myClobData = new byte[length];
int readLen = is.read(myClobData, offset, length);
```

1 個の CLOB 列から Unicode データを読み取る

Clob インターフェイスまたは Reader を使用することにより、Unicode データを CLOB 列から読み取ることができます。以下のコードは、Clob インターフェイスを使用して、CLOB 列から Unicode データを読み取る方法を説明しています。

```
// ResultSet から Clob を取得する
Clob myClob = rs.getClob("myClobColumn");
// Clob データを読み取るために入力ストリームを取得する
Reader cs = myClob.getCharacterStream();
// Reader を使用して、Clob データを読み取る
char[] myClobData = new char[length];
int readLen = cs.read(myClobData, offset, length);
```

読み取りプロセスを使用することにより、Unicode データを CLOB 列から読み取るには:

```
// ResultSet から Reader を取得する
Reader cs = rs.getCharacterStream("myClobColumn");
// Reader を使用して Clob データを読み取る
char[] myClobData = new char[length];
int readLen = cs.read(myClobData, offset, length);
```

CLOB データの更新

Clob インターフェイスのメソッド、または、ResultSet インターフェイスの updateClob メソッドを使用することにより、CLOB データを更新できます。JDBC/MX ドライバーは、CLOB データを直接変更します。し

たがって、JDBC/MX ドライバーは、DatabaseMetaData インターフェイスの locatorsUpdateCopy メソッドに対して偽を返します。アプリケーションは、個別の UPDATE 文を実行して、CLOB データを更新する必要はありません。

次の方法で、CLOB データを更新します。

- **updateClob メソッドを用いる Clob オブジェクトの更新**(58 ページ)
- **Clob オブジェクトの置換**(58 ページ)

updateClob メソッドを用いる Clob オブジェクトの更新

一部の LOB サポート実装とは異なり、JDBC/MX ドライバーは、データベースの CLOB データを直接更新します。したがって、updateClob メソッドの Clob オブジェクトが、getClob を使用して取得した Clob オブジェクトと同じである場合、ResultSet インターフェイスの updateRow メソッドは、Clob オブジェクトと共に何も行いません。

Clob オブジェクトが異なる場合、 updateClob メソッドの Clob オブジェクトは、setClob メソッドが発行されたかのように動作します。**SetClob メソッドを使用して Clob オブジェクトを挿入する**(56 ページ) を参照してください。

Clob オブジェクトの置換

次の方法で Clob オブジェクトを置き換えることができます。

- EMPTY_CLOB() 関数を使用して、Clob オブジェクトを空の Clob オブジェクトに置換して、次に、**Clob インターフェイスを使用する CLOB 列の挿入**(54 ページ) で説明するように、新しいデータを挿入します。
- PreparedStatement.setAsciiStream() または setCharacterStream() メソッドを使用して、既存の Clob オブジェクトを新しい CLOB データに置換します。
- **SetClob メソッドを使用して Clob オブジェクトを挿入する**(56 ページ) および **updateClob メソッドを用いる Clob オブジェクトの更新**(58 ページ) で、以前に説明したように、setClob または updateClob メソッドを使用して、既存の CLOB オブジェクトを置換します。

CLOB データの削除

CLOB データを削除するために、JDBC アプリケーションは、SQL DELETE 文を使用して 、実テーブルの行を削除します。

JDBC アプリケーションによって削除される行が CLOB 列を含むとき、対応する CLOB データは、実テーブルに関連付けられている削除トリガーによって自動的に削除されます。トリガーに関する情報については、**SQL/MX トリガーを使用して LOB データを削除する**(68 ページ) を参照してください。

また、**NULL と空の BLOB または CLOB 値**(61 ページ) を参照してください。

BLOB データの格納

Blob インターフェイスを使用することにより、BLOB 列上で、CLOB 列上と類似する操作を実行できます。以下が可能です。

- EMPTY_BLOB() 関数を INSERT 文で使用して、空の BLOB 列をデータベースに作成します。
- Blob インターフェイスの setBinaryStream メソッドを使用して、InputStream を取得して、BLOB データを読み取ります。

- Blob インターフェイスの getBinaryStream メソッドを使用して、OutputStream を取得して、BLOB データを書き込みます。
- PreparedStatement インターフェイスの setBinaryStream を使用して、データを BLOB 列に書き込みます。

これらの操作の詳細は、次のトピックで説明します。

- **Blob インターフェイスを使用して Blob 列を挿入する**(59 ページ)
- **バイナリ データの Blob 列への書き込み**(59 ページ)
- **PreparedStatement インターフェイスを使用して Blob 列を挿入する**(60 ページ)
- **setBlob メソッドを使用して Blob オブジェクトを挿入する**(60 ページ)

Blob インターフェイスを使用して Blob 列を挿入する

BLOB データタイプを含む行を挿入するとき、列が実際の BLOB データを用いて更新できる前に、「空の」BLOB 値を含む行を挿入できます。INSER 文の BLOB 列 に対して EMPTY_BLOB() 関数を指定することにより、空の BLOB 値を SQL/MX データベースに挿入できます。

JDBC/MX ドライバーは、EMPTY_BLOB() 関数の SQL 文字列をスキャンし、 次に利用可能なデータ ロケーターで置換します。

次に、更新用に BLOB 列を選択することにより、空の BLOB 列に対するハンドルを取得しなければなりません。

次のコードは、空の BLOB 列に対するハンドルを取得する方法を示しています。

```
Blob myBlob = null;
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("Select myBlobColumn
        from myTable where …For update");
if (rs.next())
        myBlob = rs.getBlob(1);
```

データを BLOB 列に書き込めるようになりました。**バイナリ データの Blob 列への書き込み**(59 ページ) を参照してください。

バイナリ データの Blob 列への書き込み

Blob インターフェイスを使用することにより、データを BLOB 列に書き込むことができます。次のコードは、Blob インターフェイスの setBinaryStream メソッドを使用して、BLOB データを書き込む例を示します。

```
Blob myBlob = null
// 位置 1 においてストリームを開始
long pos = 1;
// バイナリデータを含む文字列の例
String s ="BINARY_DATA";
for (int i=0; i<5000; i++) s = s + "DATA";
// Blob データを書き込む出力ストリームを取得する
OutputStream os = myBlob.setBinaryStream(pos);
// OutputStream を使用して Blob データを書き込む
byte[] myBlobData = s.getBytes();
os.write(myBlobData);
```

JDBC/MX ドライバーは、出力ストリームをチャンクに分割し、チャンクを LOB テーブルに格納します。

PreparedStatement インターフェイスを使用して Blob 列を挿入する

また、バイナリ データを含む BLOB 列を FileInputSteam を挿入できます。PreparedStatement インターフェイスを使用して、BLOB 列を挿入しなければなりません。

```
FileInputStream inputBinary = new FileInputStream(myBlobTestFile);
int blobLen = inputBinary.available();
PreparedStatement ps = conn.prepareStatement("insert
        into myTable (myBlobColumn) values (?)");
ps.setBinaryStream(1, inputBinary, blobLen);
ps.executeUpdate();
```

JDBC/MX ドライバーは、FileInputSteam からデータを読み取り、データを LOB テーブルに書き込みます。JDBC/MX ドライバーは、次に利用可能なデータ ロケーターをテーブルの BLOB 列のパラメーターの代わりに用います。

setBlob メソッドを使用して Blob オブジェクトを挿入する

JDBC アプリケーションは、直接、Blob オブジェクトをインスタンス化できません。同等の操作を実行するには:

1. ResultSet インターフェイスの getClob メソッドを使用することにより、Blob オブジェクトを取得します。
2. PreparedStatement インターフェイスの setBlob メソッドを使用することにより、Blob オブジェクトを別の行に挿入します。

このようなシチュエーションでは、アプリケーションが、PreparedStatement インターフェイスの setBlob メソッドを発行するときに、JDBC/MX ドライバーは、新しいデータ ロケーターを生成し、ソース Blob の内容を新しい Blob オブジェクトにコピーします。

BLOB 列からのバイナリデータの読み取り

Blob インターフェイスまたは InputStream を使用することにより、バイナリデータを BLOB 列をから読み取ることができます。以下のコードは、Blob インターフェイスを使用して、BLOB 列からバイナリデータを読み取る方法を説明しています。

```
// ResultSet セットから Blob を取得する
Blob myBlob = rs.getBlob("myBlobColumn");
// Blob データを読み取るために入力ストリームを取得する
InputStream is = myBlob.getBinaryStream();
// InputStream を使用して Blob データを読み取る
byte[] myBlobData = new byte[length];
is.read (myBlobData, offset, length);
```

InputStream を使用することにより、データを BLOB 列から読み取るには

```
// InputStream を ResultSet から取得する
InputStream is = rs.getBinaryStream("myBlobColumn");
// InputStream を使用して Blob データを読み取る
byte[] myBlobData = new byte[length];
is.read(myBlobData, offset, length);
```

BLOB データの更新

Blob インターフェイスのメソッド、または、ResultSet インターフェイスの updateBlob メソッドを使用することにより、BLOB データを更新できます。JDBC/MX ドライバーには、BLOB データを直接変更します。したがって、JDBC/MX ドライバーは、DatabaseMetaData インターフェイスの locatorsUpdateCopy メソッドに対して、偽を返します。アプリケーションは、個別の UPDATE 文を実行して、BLOB データを更新する必要はありません。

次の方法で、BLOB データを更新します。

- **updateBlob メソッド使用して Blob オブジェクトを更新する**(61 ページ)
- **Blob オブジェクトの置換**(61 ページ)

updateBlob メソッド使用して Blob オブジェクトを更新する

一部の LOB サポート実装とは異なり、JDBC/MX ドライバーは、データベースの BLOB データを直接更新します。したがって、updateBlob メソッドの Blob オブジェクトが、getBlob を使用して取得したオブジェクトと同じである場合、ResultSet インターフェイスの updateRow メソッドは、Blob オブジェクトと共に何も行いません。

Blob オブジェクトが異なる場合、 updateBlob メソッドの Blob オブジェクトは、setBlob メソッドが発行されたかのように動作します。「setBlob メソッドを用いて Blob オブジェクトを挿入する」を参照してください。

Blob オブジェクトの置換

次の方法で Blob オブジェクトを置き換えることができます。

- EMPTY_BLOB() 関数を使用して、Blob オブジェクトを空の Blob オブジェクトに置換します。
- **Blob インターフェイスを使用して Blob 列を挿入する**(59 ページ) で説明するように、新しいデータを含む Blob オブジェクトを挿入することにより、行の既存 Blob オブジェクトを置換します。
- PreparedStatement インターフェイスの setBinaryStream() メソッドを使用して、既存の Blob オブジェクトを新しい BLOB データに置換します。
- **setBlob メソッドを使用して Blob オブジェクトを挿入する**(60 ページ) および **updateBlob メソッド使用して Blob オブジェクトを更新する**(61 ページ) で説明するように、setBlob または updateBlob メソッドを使用して、既存の BLOB オブジェクトを置換します。

BLOB データの削除

BLOB データを削除するために、JDBC アプリケーションは、SQL DELETE 文を使用して、実テーブルの行を削除します。

JDBC アプリケーションによって削除される行が BLOB 列を含むとき、対応する BLOB データは、実テーブルに関連付けられている削除トリガーによって自動的に削除されます。トリガーに関する情報については、**SQL/MX トリガーを使用して LOB データを削除する**(68 ページ) を参照してください。

また、**NULL と空の BLOB または CLOB 値**(61 ページ) を参照してください。

NULL と空の BLOB または CLOB 値

BLOB または CLOB 列が、INSERT 文で省略されている場合、データ ロケーターは ヌル値を持つことができます。JDBC/MX ドライバーは、アプリケーションがこのような列の値を取得するときに NULL を返します。

アプリケーションは、 EMPTY_BLOB() メソッドまたは EMPTY_CLOB() メソッドを使用して、空の BLOB または CLOB データを BLOB または CLOB 列に挿入し、JDBC/MX ドライバーは、データを含まない Blob または Clob オブジェクトを返します。

Blob および Clob へのアクセスを含むトランザクション

ヒューレット・パッカード エンタープライズでは、BLOB 列または CLOB 列が、外部トランザクションを使用するか、接続を手動コミットモードに設定することによってアクセスされる場合、JDBC アプリケーションが、トランザクションを制御することを推奨します。

自動コミット モードを有効化した状態で、BLOB または CLOB データを含む prepared statement を実行し、外部トランザクションが存在しない場合、JDBC/MX ドライバーは、自動コミット モードを模倣します。この操作は、LOB 業務データの挿入または更新が、実テーブルおよび LOB テーブルの両方が変更された後でのみ、コミットされることを保証します。

いくつかのインスタンスでは、オブジェクトを不明な期間保持できる場合、outputStream または Writer オブジェクトが、アプリケーションに返されます。したがって、次のインターフェイスは、LOB データが含まれ、自動コミットが有効であり、外部トランザクションが存在しないとき、例外「自動コミットがオンかつ LOB オブジェクトが含まれます」をスローします。

- Clob.setAsciiStream
- Clob.setCharacterStream
- Blob.setBinaryStream

実テーブルおよび LOB テーブルの更新中、SQL/MX または FS 例外が発生した場合、この操作に使用される内部トランザクションはロールバックされ、例外がスローされます。

JDBC/MX が、実テーブルに対する自動コミットモードおよび LOB テーブル上の挿入または更新操作を模倣している間に、SQL/MX またはファイルシステム例外が発生した場合、この操作に使用される内部トランザクションはロールバックされ、次の例外がスローされます。

トランザクション エラー {0} - {1} LOB テーブルの更新中

説明については、**29070 HY000 トランザクション エラー {0}-{1} LOB テーブルの更新中**の下のメッセージ情報を参照してください。

JDBC/MX ドライバーは、その独自のトランザクションにおいて、データ ロケーターを予約し、データ ロケーターの予約を試みる様々なプロセスの間で同時性を改善します。

詳細情報については、**トランザクション**(38 ページ) を参照してください。

Clob および Blob オブジェクト アクセスに関する留意事項

JDBC/MX ドライバーにより、現在の Clob オブジェクトまたは Blob オブジェクト (LOB オブジェクトと呼ばれます) 上でのすべての有効な操作が可能になります。これらの LOB オブジェクトを含む行が現在の行である限り、LOB オブジェクトは現在です。JDBC/MX ドライバーは、アプリケーションが、最新でない LOB オブジェクトで操作を実行しようとすると、次のメッセージを発行しながら、SQLException をスローします。

```
Lob object {object-id} is not current
```

1 つの InputStream または Reader のみ、および、1 つの OutputStream または Writer のみを、現在の LOB オブジェクトに関連付けることができます。

- アプリケーションが、InputStream または Reader を LOB オブジェクトから取得すると、JDBC/MX ドライバーは、既に LOB オブジェクトに関連付けられている InputStream または Reader を閉じます。
- 同様に、アプリケーションが、OutputStream または Writer を LOB オブジェクトから取得すると、JDBC/MX ドライバーは、既に LOB オブジェクトに関連付けられている OutputStream または Writer を閉じます。

BLOB および CLOB データ用 SQL/MX テーブルの管理

BLOB および CLOB は、SQL/MX データベースにおけるネイティブデータ タイプではありません。しかし、データベース管理者は、本セクションで説明するように、JDBC/MX ドライバーまたは MXCI の特殊 SQL 構文を使用することにより、BLOB および CLOB 列を含む SQL/MX テーブルを作成できます。管理のため、CLOB および BLOB データは、ラージ オブジェクト (LOB) と呼ばれます。これは、いずれかのデータタイプを表します。

注記:

BLOB および CLOB データのサポートには、SQL/MX テーブルが必要です。

本セクションを使用する前に、必ず、LOB のデータが含まれるテーブルのファイルの説明を参照してください。この情報は、前セクションのトピック **LOB サポート用アーキテクチャー**(53 ページ) にあります。

上記の例外を除き、本セクションでは、データベース管理者が、LOB データをサポートするために必要なテーブルを作成および管理するために必要な情報を提供します。トピックは、次のとおりです。

- **LOB 列 を含む実テーブルの作成**(64 ページ)
- **JDBC/MX Lob Admin ユーティリティを使用して LOB データを管理する**(66 ページ)
- **SQL/MX トリガーを使用して LOB データを削除する**(68 ページ)
- **CLOB および BLOB データタイプの制限事項**(68 ページ)

LOB 列 を含む実テーブルの作成

次のトピックにおいて説明するように、JDBC プログラムを書いて、LOB 列を含む実テーブルを作成すること、 SQL/MX 対話インターフェイス MXCI 使用することが可能です。

- **LOB 列のデータタイプ**(64 ページ)
- **MXCI を使用して LOB 列を含む実テーブルを作成する**(65 ページ)
- **JDBC プログラムを使用して、LOB 列を含む実テーブルを作成する**(65 ページ)

LOB 列のデータタイプ

LOB 列のデータタイプは、次のとおりです。

CLOB

キャラクター ラージ オブジェクト データ

BLOB

バイナリラージオブジェクト データ

注記:

CLOB および BLOB データタイプの仕様は、このマニュアルに説明されるように、JDBC/MX ドライバーでアクセスする実テーブルで使用することができる特殊な構文です。

MXCI を使用して LOB 列を含む実テーブルを作成する

テーブルを作成する手順を使用する前に、MXCI を使用して、実テーブルを使用するときは、次の特殊コマンドを MXCI セッションで入力して、 LOB (BLOB または CLOB) 列を含む実テーブル作成を有効化しなければならないことに注意してください。

```
CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE'
```

LOB 列を含む実テーブルを作成する以下の手順に従ってください。

1. OSS プロンプトで、SQL/MX ユーティリティ MXCI を呼び出します。タイプ:

   MXCI

2. 次のコマンドを入力し、LOB 列を含むテーブルの作成を有効化します。

   ```
   CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE'
   ```

3. CREATE TABLE 文を入力します。例えば、次の文のような単純な形式を使用する場合があります。

   ```
   CREATE TABLE table1 (c1 INTEGER NOT NULL, c2 CLOB, c3 BLOB, PRIMARY KEY(c1))
   ```

   ここで、

   table1

   実テーブルの名前。

   c1

   列 1、NOT NULL 制約を持つ INTEGER データタイプとして定義されます。

   c2

   列 2、CLOB データタイプとして定義されます。

   c3

   列 3、BLOB データタイプとして定義されます。

   PRIMARY KEY

   プライマリキーとして、**c1** を定義します。

この例は、実テーブルを作成する原型として使用します。表 (table1) および (c1, c2, and c3) の有効な名前に関する情報、CREATE TABLE 文に関する情報については、**SQL/MX リファレンスマニュアル**を参照してください。

JDBC プログラムを使用して、LOB 列を含む実テーブルを作成する

JDBC プログラムを使用して、LOB 列を含む基本テーブルを作成する場合は、他の SQL 文と同様に、CREATE TABLE 文をプログラムに挿入します。CREATE TABLE 文の例については、上記の説明 **MXCI を使用して LOB 列を含む実テーブルを作成する**(65 ページ) を参照してください。

JDBC/MX Lob Admin ユーティリティを使用して LOB データを管理する

JDBC/MX ドライバーは、これらのタスクに使用できるよう JDBC/MX Lob Admin ユーティリティを提供しています。

- LOB テーブル (LOB のデータを保持するテーブル) を作成します。
- 実テーブル LOB 列用の SQL/MX トリガーを作成して、親なし LOB データが LOB テーブルで発生しないことを確認します。

JDBC/MX Lob Admin Utility の使用に関する情報は、次のトピックで提供されます。

- **JDBC/MX LOB Admin ユーティリティの実行**(66 ページ)
- **JDBC/MX Lob Admin ユーティリティからのヘルプリスト**(67 ページ)
- **SQL/MX トリガーを使用して LOB データを削除する**(68 ページ)

JDBC/MX LOB Admin ユーティリティの実行

OSS 環境で、次の形式を使用して、JDBC/MX LOB Admin ユーティリティを実行します。

```
java [<java_options>] JdbcMxLobAdmin [<program_options>] [<table_name>]
```

java_options

java_options は、`java` コマンドラインの `-D` オプションで使用できるプロパティです。

プロパティの仕様	説明
`jdbcmx.blobTableName`	`BLOB` 列に使用する、LOB テーブルを指定します。`BLOB` 列を使用する場合指定必須です。**LOB テーブル名のプロパティ**(30 ページ) を参照してください。
`jdbcmx.clobTableName`	`CLOB` 列に使用する、LOB テーブルを指定します。`CLOB` 列を使用する場合指定必須です。**LOB テーブル名のプロパティ**(30 ページ) を参照してください。.
`jdbcmx.catalog`	デフォルト カタログを設定します。**デフォルトのカタログおよびスキーマ**(30 ページ) を参照してください。
`jdbcmx.schema`	デフォルト スキーマを設定します。**デフォルトのカタログおよびスキーマ**(30 ページ) を参照してください。

program_options

prog_option	説明
`-help`	ヘルプ情報を表示します。
`-exec`	生成された SQL 文を実行します。

表は続く

prog_option	説明
-create	LOB テーブルを作成する SQL 文を生成します。これらの文は、テーブルのアーキテクチャーを記述します。したがって、LOB テーブルの説明を提供します。
-trigger （トリガー）	指定したテーブルのトリガーを作成する SQL 文を生成します。テーブルが存在しなければなりません。
-unicode	LOB テーブルを作成する SQL 文を生成します。CLOB データ用のみに使用します。
-drop	指定されたテーブルのトリガーをドロップする SQL 文を生成します。テーブルが存在しなければなりません。
-out	指定された SQL 文を OSS ファイルに書き込みます。
-bigblock	24K バイト サイズの LOB 列および 32K バイト サイズの属性ブロックを作成するための SQL 文を生成します。

table_name

Table_name は、LOB 列を含む実テーブルを表します。**table_name** の形式は、次のとおりです。

[**catalogName.**][**schemaName.**]**baseTableName**

カタログ、スキーマ、テーブル名に関する情報については、**SQL/MX リファレンスマニュアル**を参照してください。

JDBC/MX Lob Admin ユーティリティからのヘルプリスト

JDBC/MX Lob Admin ユーティリティのヘルプを表示するコマンドと、それに続くヘルプリストを以下に示します。

```
java JdbcMxLobAdmin -help
```

Hewlett-Packard JDBC/MX Lob Admin Utility 2.0 (c) Copyright 2004, 2005 Hewlett-Packard Development Company, LP.

```
java [<java_options>] JdbcMxLobAdmin [<prog_options>] [<table_name>]

<java_options> is:
    [-Djdbcmx.clobTableName=<clobTableName>]
    [-Djdbcmx.blobTableName=<blobTableName>]
    [-Djdbcmx.catalog=<catalog>]
    [-Djdbcmx.schema=<schema>]

<prog_options> is:
    [-exec] [-create] [-trigger] [-unicode] [-help] [-drop] [-out <filename>][-bigblock]
where -help - Display this information.
        -exec - Execute the SQL statements that are generated.
        -create - Generate SQL statements to create LOB tables.
        -trigger - Generate SQL statements to create triggers for <table_name>.
        -unicode - Generate SQL statements to create unicode LOB tables (CLOB only).
        -drop - Generate SQL statements to drop triggers for <table_name>.
        -out - Write the SQL statements to <filename>.
        -bigblock - Generates SQL statement to create LOB column size of 24K bytes and attribute block size
of 32K.

<clobTableName> | <blobTableName> is:
    <catalogName>.<schemaName>.<lobTableName>

<table_name> is:
    [<catalogName>.][<schemaName>.]<baseTableName>
```

```
<baseTableName> is the table that contains LOB column(s).
<lobTableName> is the table that contains the LOB data.
```

SQL/MX トリガーを使用して LOB データを削除する

JDBC/MX Lob Admin ユーティリティを使用して、基本の行を削除するとき、トリガーを生成して、LOB データを LOB テーブルから削除します。これらのトリガーは、親なし LOB データが、LOB テーブルで発生しないことを確認します。トリガーを管理するには、これらの JDBC/MX Lob Admin ユーティリティのオプションを使用します。

`-trigger (トリガー)`

SQL 文を生成し、トリガーを作成します。

`-drop`

SQL 文を生成し、トリガーをドロップします。

例えば、次のコマンド (1 行で入力) は、SQL 文を生成し、実テーブル `sales.paris.pictures` に対するトリガーを作成します。これには、`BLOB` 列が含まれ、次の文を実行します。

```
java -Djdbcmx.blobTableName=sales.paris.lobTable4pictures JdbcMxLobAdmin -trigger
        -exec sales.paris.pictures
```

CLOB および BLOB データタイプの制限事項

CLOB および BLOB データタイプ、総称して、LOB データの制限事項は次のとおりです。

- LOB 列は、これらの SQL 文のターゲット列リストにのみ存在できます。
  - INSERT 文
  - SELECT 文選択リスト
  - UPDATE 文の SET 句の列名
- LOB 列は、SQL/MX 関数および式で参照できません。
- 見出しは、システムによって自動的に生成されますので、LOB 列の明示的な見出しは許可されません。
- トリガーを確立していない限り、基本の行を削除すると、LOB テーブルから LOB データは削除されません。トリガーに関する情報については、**SQL/MX トリガーを使用して LOB データを削除する**(68 ページ)を参照してください。
- 実テーブル名が変更される場合、LOB データにアクセスできません。
- LOB 列は、SQL/MX ユーティリティ コマンドを使用して、別のテーブルにコピーできません。
- `CLOB` または `BLOB`

  列を含む実テーブルの名前は、これらの実テーブルの複数が 1 つの LOB テーブルを共有する場合、すべてのカタログおよびスキーマ全体で、一意でなければなりません。

モジュール ファイル キャッシング

モジュール ファイル キャッシング (MFC) 機能は、JDBC/MX T2 データベース接続間や JVM プロセス間で、SQL/MX prepared statement プランを共有します。これは、JDBC/MX T2 アプリケーションの定常状態の中に SQL/MX コンパイル時間を短縮し、リソース消費量を減らします。

注記: モジュール ファイル キャッシングは、J06.07 以降の J シリーズ RVU を実行するシステムでのみサポートされます。

本チャプターでは、次のトピックについて説明します。

新機能

SQL/MX リリース 3.2 以降、MFC は、パラメーター化されたクエリにおいて、`BigNum` データタイプをサポートします。

MFC の設計

MFC 設計の詳細については、**HPE NonStop SQL/MX 接続サービス マニュアル**を参照してください。

MFC の有効化

以下は、JDBC/MX T2 ドライバーを使用するアプリケーションにおいて、MFC を使用するために必要な 2 つの新しいプロパティです。

- `enableMFC` プロパティ:

  MFC を有効化するには、このプロパティの値を ON に設定する必要があります。

- `compiledModuleLocation` プロパティ:

  このプロパティの値は、有効なディレクトリ名でなければなりません。例: `/usr/temp`。これは、MFC 用の *.mdf などの中間ファイルを生成する場所です。

これらのプロパティ両方を MFC を有効にするため に設定しなければなりません。

MFC の制限事項

- MFC キャッシュは、本番環境システム上でのみ使用する必要があります。SQL/MX が変更される開発またはユーザー受け入れテスト (UAT) システム上では使用しないでください。
- セッション固有 SQL/MX CONTROL QUERY DEFAULTS (CQD) および SQL/MX CONTROL QUERY SHAPE (CQS) は処理しません。

- ヒューレット・パッカード エンタープライズは、MFC を使用するとき、SQL/MX の自動再コンパイル機能を無効にするため、SQL/MX で CQD を設定することを推奨します。自動再コンパイルを無効化する CQD は、「CONTROL QUERY DEFAULT RECOMPILATION_WARNINGS 'ON'」です。'". プランがモジュール ファイルで生成されるため、これにより、SQL/MX オブジェクトにおける変更により、自動再コンパイルが回避されます。アプリケーションは、クエリに必要な自動再コンパイルがある場合、SQL/MX 例外を受け取ります。アプリケーションの作業を続行する前に、古いモジュール ファイルを消去しなければなりません。CQD に関する詳細情報については、最新の HPE NonStop SQL/MX リリース リファレンスマニュアル を参照してください。
- 軽量クエリの場合、MFC は、SQL/MX コンパイルより若干優れています。
- 外部ステートメントキャッシュを組み合わせることで、メモリの利点を生成しません。WebLogic サーバー (WLS) のステートメントキャッシュは、外部ステートメントキャッシュの例です。JDBC/MX T2 ステートメントキャッシュを使用することを推奨します。
- MFC は、BLOB および CLOB データタイプをサポートしません。

MFC のトラブルシューティング

MFC のトラブルシューティングには、以下が含まれます。

MFC の利点

java.sql.PreparedStatement オブジェクトを使用する JDBC アプリケーションは、プロセッサー利用率とメモリ消費量を低下させ、応答時間を短縮します。

MFC の環境の設定

MFC の有効化(69 ページ)を参照してください。

.lock ファイル

*.lock ファイルは、MFC モジュール ファイルの作成プロセスを通過するすべてのクエリに対して生成されます。これらのファイルは、別の接続が同じモジュール ファイルを再作成しないように、同期に対して使用されます。これらの *.lock ファイルは、バイナリ モジュールが、/usr/tandem/sqlmx/USERMODULES ディレクトリに正常に作成されると削除されます。

*.lock ファイルは、モジュール ファイルを作成できないクエリの場合は削除されません。

.mdf ファイル

これらの一時ファイルは、前処理中に生成されます。これらの .mdf ファイルは、容易なサポートとトラブルシューティングのために保持されます。

ディスク アクティビティ

ディスク OSH 位置 (/usr/tandem/sqlmx/USERMODULE) に格納される MFC アクセスプランは、ディスクのプロセスまたは使用率を増加させます。この問題を克服するには、そのディレクトリのファイルセットを使用します。データボリューム上に OSS キャッシュを持つことが有益です。ファイルセットおよび OSS キャッシュを有効化するには、**ファイルセットおよび OSS のキャッシュを有効化する**(71 ページ) を参照してください。

注記:

- DDL を変更した場合は、管理スクリプト (mgscript) を実行して、テーブルまたはカタログに関連付けられているモジュール ファイルを削除することを推奨します。管理スクリプトに関する情報については、**HPE NonStop SQL/MX 接続サービス における MFC 管理マニュアル**を参照してください。
- テーブルまたは一連のテーブルで DDL 変更が行われた場合、予期しない動作を防止するため、そのテーブルまたは一連のテーブルが関連付けられているモジュール ファイルを削除しなければなりません。

ファイルセットおよび OSS のキャッシュを有効化する

USERMODULES ディレクトリをポイントするファイルセットを追加するには、以下の手順を実行します。

1. TACL プロンプトで、

   SCF

   を入力し、次に、

   assume $zpmon を入力します

2. SCF プロンプトで、SCF コマンド:

   add server #zpns1,cpu 1,backupcpu 2 を入力します

3. ファイルセットを追加します。

   ```
   add fileset mxc1,nameserver #zpns1,catalog $oss, pool
   mxcpool, mntpoint "/usr/tandem/sqlmx/USERMODULES"
   ```

4. ファイルセットのステータスを確認します。

   ```
   info fileset mxc1,detail
   ```

5. ファイルセットを起動します。

   ```
   start fileset mxc1
   ```

OSS キャッシュを有効化するために、次の手順を実行します。

1. TACL プロンプトで、

   SCF

   を入力し、次に、

assume $zpmon を入力します

2. SCF プロンプトで、ご使用のシステム上のすべてのファイルセットを停止するために次の SCF コマンドを入力します。

   STOP FILESET $ZPMON.*

   このコマンドは、マウントされた最後のファイルセットで始まり、最後に起動された逆の順番でファイルセットを停止します。

3. OSS 監視プロセスを停止します。

   OSS 監視が、標準プロセスとして実行中の場合は、TACL プロンプトで、STOP コマンドを入力します。

   STOP $ZPMON

   OSS 監視が、永続プロセスとして実行中の場合は、SCF プロンプトで、ABORT コマンドを入力します。

   ABORT PROCESS $ZZKRN.#ZPMON

4. SCF プロンプトで、ファイルセットの各ディスクボリュームに対して、次の一連のコマンドを入力します。

   STOP DISK diskname

   ALTER DISK diskname, OSSCACHING ON

   START DISK diskname

   diskname とは、OSS ファイルを含むディスクボリュームの名前です。

5. 適切なコマンドを用いて、正常なまたは永続性のあるプロセスとして OSS モニターを再起動します。
6. SCF コマンドを入力して、OSS ファイルシステムを再起動します。

   START FILESET $ZPMON.filesetname

   ここで、filesetname は、root で始まる、マウントポイントの発生順序で指定された OSS ファイルを含む、それぞれのファイルセットの名前です。

既知の問題

シナリオ 1

実テーブルを変更または削除すると、MFC プランが廃止されます。次の操作手順は、この問題を示しています。

操作	予想される結果	実際の結果	備考
Create table testing(info int);	Success	Success	testing テーブルが作成されます。
Stmt1 = Prepare("select * from testing")	Success	Success	Stmt1 は、MXCMP を用いて準備されます。
Stmt1.execute()	Success	Success	Stmt1 が実行されます。
Stmt1.fetch()	Success	Success	testing テーブル中のデータが取得されます。

表は続く

操作	予想される結果	実際の結果	備考
`Stmt1 = Prepare("select * from testing")`	Success	Success	コンパイル済みプランが MFC から取得されます。
`Stmt1.execute()`	Success	Success	MFC 文は、期待どおりに動作します。
`Stmt1.executeUpdate("drop table testing")`	Success	Success	testing テーブルが削除されます。
`Stmt1.executeUpdate("create table testing(mycol varchar(10))")`	Success	Success	testing テーブルに varchar 列が作成されます。
`Stmt1 = Prepare("select * from testing")`	Success	Success	MFC プランを作成された時から、テーブルのデータタイプが変更されているため、正しくないコンパイル済みプランが、MFC から取得されます。
`Stmt1.execute()`	Success	Failure	JDBC T2 ドライバーは、SQL/MX CQD recompilation_warnings を ON に切り替えます。SQL/MX は、類似性チェックの失敗時に SQL 例外をスローし、JDBC T2 ドライバーは、`compiledmodulelocation` プロパティを使用して、指定された位置から無効なモジュール ファイルをドロップします。
`Stmt1 = Prepare("select * from testing")`	Success	Success	新しいプランが、MFC 位置に作成されます。
`Stmt1.execute()`	Success	Success	MFC 文は、期待どおりに動作します。

上記の操作を行う場合は、MFC で無効なモジュール ファイルが見つかったとき、execute() コールが失敗します。しかし、後続の prepare() コールは、新しいモジュール ファイルを作成します。この問題は、ドライバー側キャッシュが、JDBC/MX T2 ドライバー内に存在する問題に類似しています。

JDBC/MX の準拠

NonStop SQL/MX 用 JDBC/MX ドライバーは、該当する場合、Oracle JDBC 3.0 API 仕様に準拠しています。しかし、JDBC/MX ドライバーは、NonStop SQL/MX および JDBC/MX ドライバーの制限があるためにいくつかの方法で、JDBC 標準と異なります。本サブセクションでは、未サポートの JDBC メソッド、仕様から逸脱するメソッドおよび機能、JDBC 標準に対する HPE 拡張ソフトウェアを説明します。仕様に準拠する JDBC 機能は、本サブセクションでは説明しません。

トピックは、次のとおりです。

サポートされない機能

`java.sql` パッケージの次のインターフェースは、含まれるデータタイプが、NonStop SQL/MX によってサポートされないため、JDBC/MX ドライバーに実装されません。

- `java.sql.Array`
- `java.sql.Ref`
- `java.sql.Savepoint`
- `java.sql.SQLData`
- `java.sql.SQLInput`
- `java.sql.SQLOutput`
- `java.sql.Struct`

注記: `java.sql.Blob` および `java.sql.Clob` パッケージのサポートには、**BLOB および CLOB データを用いる作業**(52 ページ) で説明されるように、SQL/MX のユーザー テーブルを使用する必要があります。これらのパッケージは、SQL/MP ユーザー テーブルのアクセスをサポートしていません。

次のメソッドは、`java.sql` パッケージで、`SQLException` をメッセージ「未サポートの機能 -メソッド-名」と共にスローします。

メソッド	コメント
CallableStatement.getArray(int parameterIndex) CallableStatement.getArray(String parameterName) CallableStatement.getBlob(int parameterIndex) CallableStatement.getBlob(String parameterName) CallableStatement.getClob(int parameterIndex) CallableStatement.getClob(String parameterName) CallableStatement.getObject(int parameterIndex, Map map) CallableStatement.getObject(String parameterName, Map map) CallableStatement.getRef(int parameterIndex) CallableStatement.getRef(String parameterName) CallableStatement.getURL(int parameterIndex) CallableStatement.getURL(String parameterName) CallableStatement.executeBatch()	特定の CallableStatement メソッドはサポートされていません。
Connection.releaseSavepoint(Savepoint savepoint) Connection.rollback(Savepoint savepoint) Connection.setSavepoint() Connection.setSavepoint(String name)	特定の Connection メソッドはサポートされていません。
PreparedStatement.setArray(int parameterIndex, Array x) PreparedStatement.setRef(int parameterIndex, Ref x) PreparedStatement.setURL(int parameterIndex, URL x)	特定の PreparedStatement メソッドはサポートされていません。
ResultSet.getArray(int columnIndex) ResultSet.getArray(String columnName) ResultSet.getObject(int columnIndex, Map map) ResultSet.getObject(String columnName, Map map) ResultSet.getRef(int columnIndex) ResultSet.getRef(String columnName) ResultSet.getURL(int columnIndex) ResultSet.getURL(String columnName) ResultSet.updateArray(int columnIndex) ResultSet.updateArray(String columnName) ResultSet.updateRef(int columnIndex) ResultSet.updateRef(String columnName)	特定の ResultSet メソッドはサポートされていません。

SQL/MP ユーザー テーブルへのアクセスに使用する場合は、次のメソッドはサポートされません。

メソッド	コメント
`PreparedStatement.setBlob(int parameterIndex, Blob x)` `PreparedStatement.setClob(int parameterIndex, Clob x)`	特定の `PreparedStatement` メソッドは、SQL/MP ユーザー テーブルのみへのアクセスのためにサポートされていません。

`ResultSet.getBlob(int columnIndex)`
`ResultSet.getBlob(String columnName)`
`ResultSet.getClob(int columnIndex)`
`ResultSet.getClob(String columnName)`
`ResultSet.updateBlob(int columnIndex)`
`ResultSet.updateBlob(String columnName)`
`ResultSet.updateClob(int columnIndex)`
`ResultSet.updateClob(String columnName)`
	特定の `ResultSet` メソッドは、SQL/MP ユーザー テーブルのみへのアクセスのためにサポートされていません。

次のメソッドは、`java.sql` パッケージで、`SQLException` をメッセージ「自動生成されたキーはサポートされません」と共にスローします。

メソッド	コメント

`Connection.prepareStatement(String sql, intautoGeneratedKeys)`
`Connection.prepareStatement(String sql, int[]columnIndexes)`
`Connection.prepareStatement(String sql, String[]columnNames)`
	自動的に生成されたキーはサポートされません。

`Statement.execute(String sql, int autoGeneratedKeys)`
`Statement.execute(String sql, int[] columnIndexes)`
`Statement.execute(String sql, String[] columnNames)`
`Statement.executeUpdate(String sql, intautoGeneratedKeys)`
`Statement.executeUpdate(String sql, int[]columnIndexes)`
`Statement.executeUpdate(String sql, String[]columnNames)`
`Statement.getGeneratedKeys()`
	自動的に生成されたキーはサポートされません。

次のメソッドは、`java.sql` パッケージで、`SQLException` をメッセージ「データタイプがサポートされません」と共にスローします。

メソッド	コメント
`CallableStatement.getBytes(int parameterIndex)` `CallableStatement.getBytes(String parameterName)`	特定のデータタイプはサポートされません。
`CallableStatement.setBytes(String parameterIndex,bytes[] x)`	BLOB、VARCHAR、BINARY、LONGVARCHAR、VARBINARY、LONGVARBINARY のみをサポートします。その他の特定のデータタイプはサポートされません、
`PreparedStatement.setBytes(int ColumnIndex, bytes[]x)`	BLOB、CHAR、DATE、TIME, TIMESTAMP,VARCHAR、BINARY、LONGVARCHAR、VARBINARY、LONGVARBINARY のみをサポートします。その他の特定のデータタイプはサポートされません、
`PreparedStatement.setObject(int parameterIndex,Object x int targetSqlType)` `PreparedStatement.setString(int parameterIndex,String x)`	ARRAY、BINARY、BIT、DATALINK、JAVA_OBJECT、および REF タイプはサポートされません。
`ResultSet.getBytes(int ColumnIndex)` `ResultSet.getBytes(String ColumnName)`	BLOB、CHAR、VARCHAR、BINARY、LONGVARCHAR、VARBINARY、LONGVARBINARY のみをサポートします。その他の特定のデータタイプはサポートされません、

`javax.sql` パッケージ次のオプション インターフェイスは、JDBC/MX ドライバーに実装されていません。

メソッド	コメント
javax.sql.XAConnection javax.sql.XADataSource	分散トランザクション は、JDBC 3.0 API 仕様に従い、実装されていません。
javax.sql.RowSet javax.sql.RowSetInternal javax.sql.RowSetListener javax.sql.RowSetMetaData javax.sql.RowSetReader javax.sql.RowSetWriter	RowSet は、JDBC/MX ドライバーで実装されていません。しかし、RowSet のリファレンス インプリメンテーションを Oracle (**http://java.sun.com/developer/earlyAccess/jdbc/jdbc-rowset.html**) からダウンロードできます。
javax.sql.JdbcRowSet.getArray(intcolumnIndex) javax.sql.JdbcRowSet.getArray(StringcolumnName) javax.sql.JdbcRowSet.getObject(intcolumnIndex, Map map) javax.sql.JdbcRowSet.getObject(StringcolumnName, Map map) javax.sql.JdbcRowSet.getRef(intcolumnIndex) javax.sql.JdbcRowSet.getRef(StringcolumnName) javax.sql.JdbcRowSet.getURL(intcolumnIndex) javax.sql.JdbcRowSet.getURL(StringcolumnName) javax.sql.JdbcRowSet.rollback(Savepoint savepoint) javax.sql.JdbcRowSet.setArray(intparameterIndex, Array x) javax.sql.JdbcRowSet.setRef(intparameterIndex, Ref x) javax.sql.JdbcRowSet.updateArray(intcolumnIndex) javax.sql.JdbcRowSet.updateArray(StringcolumnName) javax.sql.JdbcRowSet.updateRef(intcolumnIndex) javax.sql.JdbcRowSet.updateRef(intcolumnIndex)	JdbcRowSet API メソッドは、ここに挙がっているもの以外サポートされます。ここに挙がっているものは Unsupported feature -method-name SQLMXException をスローします。

一部のメソッドの逸脱に関する追加情報については、**逸脱**(78 ページ) を参照してください。

JDBC/MX タイプ 2 ドライバーの制限事項

SQL/MX リリース 3.2 用 JDBC/MX タイプ 2 ドライバーは、SQL/MX リリース 2.x と相互運用できません。SQL/MX Release 3.2 用 JDBC/MX タイプ 2 ドライバーで SQL/MX Release 2.x への接続を試みる場合、その動作は予測不能です。

逸脱

次の表は、JDBC 仕様とは挙動が異なるメソッドを列挙します。メソッドで引数が無視される場合、JDBC/MX ドライバーは、SQLException をスローしません。したがって、アプリケーションの処理を続行が許可され

ます。しかし、アプリケーションが、期待どおりの結果を取得できない可能性があります。列挙される他のメソッドは、仕様と異なる場合でも、特に明記されない限り、SQLException をスローしません。

注記: java.sql.DatabaseMetaData.getVersionColumns () メソッドは、java.sql.DatabaseMetaData.getBestRowIdentifier() メソッドを模倣します。これは、SQL/MX が SQL_ROWVER (指定されたテーブルの列の内で、トランザクションによって行のいずれかのデータが更新される時に、データ ソースによって自動的に更新される列があれば返す関数) をサポートしないためです。

メソッド	コメント
java.sql.DatabaseMetaData.getColumns(Stringcatalog, String schemaPattern, StringtableNamePattern, String columnNamePattern)	SQL/MX が、次のタイプの列タイプをサポートしていないため、列は列データに追加されますが、値は NULL に設定されます。SCOPE_CATALOG、SCOPE_SCHEMA、SCOPE_TABLE、および SOURCE_DATA_TYPE。
java.sql.DatabaseMetaData.getSchemas()	TABLE_CATALOG は、列データに追加され、カタログ名が返されます。
java.sql.DatabaseMetaData.getTables(String catalog, String schemaPattern, String[] types)	SQL/MX が、次のタイプの列タイプをサポートしていないため、列は列データに追加されますが、値は NULL に設定されます。TYPE_CAT、TYPE_SCHEMA、TYPE_NAME、SELF_REFERENCING_COL_NAME、および REF_GENERATION。
java.sql.DatabaseMetaData.getUDTs(String catalog,String schemaPattern, String tableNamePattern,int[] types)	SQL/MX が、基本タイプをサポートしていないため、BASE_TYPE は列データに追加されますが、値は NULL に設定されます。
java.sql.DatabaseMetaData.getVersionColumns()	DatabaseMetaData.getBestRowIdentifier() メソッドを模倣します。これは、SQL/MX が SQL_ROWVER (指定されたテーブルの列の内で、トランザクションによって行のいずれかのデータが更新される時に、データ ソースによって自動的に更新される列があれば返す関数) をサポートしないためです。
java.sql.DriverManager.getConnection(String url,String usr, String password) java.sql.DriverManager.getConnection(String url,Properties info) javax.sql.DataSource.getConnection(Stringusername, String password)	ユーザー名およびパスワードの引数は無視されます。すべての接続は、Java VM を呼び出したユーザーとして同じセキュリティ特権を持ちます。
java.sql.DriverManager.setLoginTImeout(...) javax.sql.DataSource.setLoginTimeout(...)	ログイン タイムアウトは無視されます。
javax.sql.DataSource.setLogWriter	このメソッドは、JDBC トレース機能が有効である限り、影響を与えません。JDBC トレース機能の情報については、 **NonStop Server for Java プログラマ リファレンス**を参照してください。

表は続く

メソッド	コメント
java.sql.Connection.createStatement(...) java.sql.Connection.prepareStatement(...)	JDBC/MX ドライバーは、スクロール可能な結果セット タイプをサポートしません。したがって、アプリケーションがこのタイプを要求すると、SQLWarning が発行されます。結果セットは、スクロール可能タイプに変更されます。
java.sql.Connection.setReadOnly(...)	読み取り専用の属性は無視されます。
java.sql.ResultSetMetaData.getPrecision(intcolumn) java.sql.ResultSetMetaData.getColumnDisplaySize(intcolumn)	CLOB および BLOB 列の場合、これらのメソッドは、無制限値を示すために 0 を返します。標準 API に従い、getPrecision () メソッドおよび getColumnDisplaySize () メソッドは整数値を返しますが、最大整数値より大きい LOB データをデータベースに格納できます。
java.sql.ResultSet.setFetchDirection(...)	フェッチ方向属性は無視されます。
java.sql.Statement.setEscapeProcessing(...)	SQL/MX が、エスケープ構文を解析するため、エスケープ処理を無効にしても効果はありません。
java.sql.Statement.setFetchDirection(...)	フェッチ方向属性は無視されます。
java.sql.Statement.setQueryTimeout(...)	クエリのタイムアウト値は無視されます。クエリ タイムアウト時間が経過しても、JDBC/MX ドライバーは実行を中断しません。
javax.sql.JdbcRowSet.setUsername(String username) javax.sql.JdbcRowSet.setPassword(String password) javax.sql.JdbcRowSet(String url, String username,String password)	ユーザーを認証するように設定しない場合、JDBC T2 ドライバーは、ユーザー名およびパスワードを無視します。
javax.sql.JdbcRowSet.setReadOnly(...)	読み取り専用の属性は無視されます。
javax.sql.JdbcRowSet.setEscapeProcessing(...)	SQL/MX が、エスケープ構文を解析するため、エスケープ処理を無効にしても効果はありません。
javax.sql.JdbcRowSet.setFetchDirection(...)	フェッチ方向属性は無視されます。
javax.sql.JdbcRowSet.setQueryTimeout(...)	クエリのタイムアウト値は無視されます。クエリ タイムアウト時間が経過しても、JDBC/MX ドライバーは実行を中断しません。

以下の機能は/MX JDBC ドライバーで実装他のドライバーからの実装とは異なる場合があります。

更新可能結果セット

JDBC/MX ドライバーは、読み取り専用および更新可能な並列モードの両方をサポートしています。JDBC/MX ドライバーは、更新可能な結果セットに対して、次の条件を想定しています。

- 結果セットの最初の列のテーブル名が、更新されるテーブルと見なされます。この仮定により、複数のテーブルからのクエリで、かつ更新可能とすることができます。
- クエリは、更新されるテーブルのプライマリキー列を選択します。

JDBC/MX ドライバーは、次の条件のいずれかが発生したとき、`SQLException` をスローします。

- プライマリキー列が更新されたとき。
- 最新の読み取り時刻以降、選択した列が更新されたとき。(しかし、JDBC/MX ドライバーは、最後に列が読み取られて以降、選択クエリの一部ではないテーブルの列が未変更であることを確認しません。)
- クエリは、ヌルにできない列またはデフォルト値を含まない列を選択しません。

結果セットは、次の方法でも影響されます。

- 削除された行が結果セットから削除されます。メソッド `databaseMetaData.deletesAreDetected()` が偽を返します。
- 挿入された行は、結果セットの現在のカーソル位置に追加されます。メソッド `databaseMetaData.insertsAreDetected()` が真を返します。

バッチ アップデート

バッチ アップデート機能により、`Statement` オブジェクトは、異なる種類の更新、挿入、削除コマンドをまとめて、データベースに対するシングル ユニットとしてサブミットできます。また、この機能により、複数のパラメーター セットを `PreparedStatement` オブジェクトに関連付けられます。

自動コミット モードを有効にすると、JDBC/MX ドライバーは、バッチ内のすべてのコマンドが成功する場合にのみ、更新をコミットします。バッチでの任意のコマンドが失敗すると、自動コミットおよび非自動コミットモードの両方で、更新がロールバックされます。

BatchUpdate 例外処理の改善サポートにより JDBC ドライバーは、BatchUpdateExceptions 後も、バッチにおいて、残りのジョブの処理を継続するようになりました。実行中に発生したすべてのバッチ例外がある場合、例外をキューに入れ、残りのバッチ コマンドを実行します。バッチ内のすべての要素の実行の完了時にキューに入れられた例外がスローされます。ユーザーアプリケーションが、例外時のバッチ トランザクションをハンドル、コミット、ロールバックしなければなりません。これにより、ジョブ全体を再実行を回避できます。ただし、TMF エラーによるトランザクション失敗の結果の場合、この機能強化で対処することはできません。

DatabaseMetaData コール

DatabaseMetaData コールの場合、カタログおよびスキーマの値を指定します。これらの値を指定しないと、JDBC/MX ドライバーは、次のデフォルト値を使用します。

カタログ = `NONSTOP_SQLMX_NSK;`
スキーマ = `PUBLIC_ACCESS_SCHEMA;`

ヒューレット・パッカード エンタープライズ拡張ソフトウェア

JDBC 標準に対する次の拡張機能が、JDBC/MX ドライバーで実装されます。

間隔データタイプ

間隔データタイプは、Java 2 JDBC 3.0 仕様で定義されている汎用 SQL タイプではありませんが、SQL/MX は、間隔データタイプをサポートします。SQL/MX 用 JDBC アプリケーションに間隔データタイプへのアクセスを許可するために、JDBC/MX ドライバーは、それを `Types.OTHER` データタイプにマップします。

JDBC/MX ドライバーは、ResultSet インターフェイスの getObject() および getString() メソッド、PreparedStatement インターフェイスの setObject() および setString() メソッドを有効化して、このデータタイプにアクセスします。間隔データタイプは、文字列 オブジェクトとして、常にアクセスされます。また、JDBC/MX ドライバーは、間隔リテラルに対するエスケープ構文を許可します。

国際化機能

JDBC/MX ドライバーは、Java メッセージが、様々な言語を採用できるように設計されています。JDBC/MX コンポーネントのエラーメッセージは、ソース コードの外部に個別のプロパティ ファイルで保存されており、ロケール設定に基づいて、動的に取得されます。異なる言語でのエラーメッセージは、言語および国に基づく個別のプロパティ ファイルに格納されます。この拡張は、JDBC アプリケーションが実行している場合、発生するすべてのメッセージには適用されません。

SQL への準拠

JDBC/MX は、SQL:1999 の SQL 言語エントリーレベルに準拠しています。本サブセクションでは、以下の JDBC/MX サポートについて説明します。

SQL スカラー関数

JDBC/MX は、次の表に示すように、JDBC スカラー関数を同等の SQL/MX 関数に割り当てます。

表 8: 数値関数

JDBC 関数	同等の SQL/MX 関数
ABS	ABS
ACOS	ACOS
ASIN	ASIN
ATAN	ATAN
ATAN2	ATAN2
CEILING	CEILING
COS	COS
DEGREES	DEGREES
EXP	EXP
FLOOR	FLOOR
LOG	LOG
LOG10	LOG10

表は続く

JDBC 関数	同等の SQL/MX 関数
MOD	MOD
PI	PI
POWER	POWER
RADIANS	RADIANS
SIGN	SIGN
SIN	SIN
SINH	SINH
SQRT	SQRT
TAN	TAN

表 9: 文字列関数

JDBC 関数	同等の SQL/MX 関数
ASCII	ASCII
CHAR	CHAR
CHAR_LENGTH	CHAR_LENGTH
CONCAT	CONCAT
INSERT	INSERT
LCASE	LOWER
LEFT	SUBSTRING
LENGTH	LENGTH
LOCATE	LOCATE (JDBC LOCATE **start** パラメーターはサポートされません)
LOWER	LOWER
LPAD	LPAD
LTRIM	LTRIM
OCTET_LENGTH	OCTET_LENGTH
POSITION	POSITION
REPEAT	REPEAT
REPLACE	REPLACE
RIGHT	RIGHT
RTRIM	TRIM...TRAILING
SPACE	SPACE
SUBSTRING	SUBSTRING
UCASE	UPPER | UPSHIFT

注記: クエリ中の JDBC 文字列関数は、固定長 (CHAR) 列名に対して、予期せぬ結果を返すことがあります。これは、SQL/MX が、固定長文字列に定義長さまでブランクをパディングし、JDBC 文字列関数からの結果には、文字列の終端にトレーリング ブランクが含まれる可能性があるためです。クエリ内で RTRIM 関数を使用して、SQL/MX に列名から余分なブランクをトリムさせてください。

表 10: 時刻および日付関数

JDBC 関数	同等の SQL/MX 関数
CONVERTTIMESTAMP	CONVERTTMESTAMP
CURRENT	CURRENT
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP
CURDATE , CURRENT_DATE	CURRENT_DATE
CURTIME , CURRENT_TIME	CURRENT_TIME
DATEFORMAT	DATEFORMAT
DAY	DAY
DAYNAME	DAYNAME
DAYOFMONTH	DAYOFMONTH
DAYOFWEEK	DAYOFWEEK
DAYOFYEAR	DAYOFYEAR
EXTRACT	EXTRACT
HOUR	HOUR
JULIANTIMESTAMP	JULIANTIMESTAMP
MINUTE	MINUTE
MONTH	MONTH
MONTHNAME	MONTHNAME
QUARTER	QUARTER
SECOND	SECOND
WEEK	WEEK
YEAR	YEAR

表 11: システム関数

JDBC 関数	同等の SQL/MX 関数
CURRENT_USER	CURRENT_USER
SYSTEM_USER	SYSTEM_USER
USER	USER

CONVERT (変換) 関数

JDBC/MX は、SQL/MX CAST 関数を使用して、JDBC CONVERT 関数をサポートします。JDBC CONVERT 関数は、次の形式を有しています。

```
{ fn CONVERT( value_exp, data_type ) }
```

SQL/MX CAST は、次の形式を有しています。

```
CAST( { value_exp | NULL } AS data_type )
```

SQL/MX は、CONVERT 構文を CAST 構文に変換し、データタイプ引数を同等の SQL/MX 値に変換します。例えば、文字データに対する JDBC データタイプが、整数値 (SQL_CHAR または 1) である場合、同等の SQL/MX データタイプは、CHARACTER 値を持つ文字列リテラルになります。

JDBC データタイプ

次の表は、JDBC/MX によりサポートされている JDBC データタイプと対応する MX/SQL データタイプを示しています。

JDBC データタイプ	JDBC/MX によりサポートされているか	SQL/MX データタイプ
Types.Array	非対応	
Types.BIGINT	対応	LARGEINT
Types.BINARY	非対応	
Types.BIT	非対応	
Types.BLOB	対応	
Types.CHAR	対応	Char (n)
Types.CLOB	対応	
Types.DATE	対応	DATE
Types.DECIMAL	対応	DECIMAL(p,s)
Types.DISTINCT	非対応	
Types.DOUBLE (*)	対応	DOUBLE PRECISION
Types.FLOAT (*)	対応	FLOAT(p)
Types.INTEGER	対応	INTEGER
Types.JAVA_OBJECT	非対応	
Types.LONGVARBINARY	非対応	

表は続く

JDBC データタイプ	JDBC/MX によりサポートされているか	SQL/MX データタイプ
Types.LONGVARCHAR	対応**	VARCHAR[(n)]
Types.NULL	非対応	
Types.NUMERIC	対応	NUMERIC(p,s)
Types.NUMERIC	対応	NUMBER
Types.REAL	対応	FLOAT(p)
Types.REF	非対応	
Types.SMALLINT	対応	SMALLINT
Types.STRUCT	非対応	
Types.TIME	対応	TIME
Types.TIMESTAMP	対応	TIMESTAMP
Types.TIMESTAMP	対応	DATE type2 (DC_MODE)
Types.TINYINT	非対応	
Types.VARBINARY	非対応	
Types.VARCHAR	対応	VARCHAR(n)
Types.VARCHAR	対応	VARCHAR2(n)

* **浮動小数点のサポート**(88 ページ) を参照してください。

** 最大長に関する詳細については、**SQL/MX リファレンスマニュアル**を参照してください。

JDBC ドライバーは、次の SQL/MX データタイプを JDBC データタイプ Types.OTHER にマッピングします。

```
DATETIME YEAR
DATETIME YEAR TO MONTH
DATETIME YEAR TO DAY
DATETIME YEAR TO HOUR
DATETIME YEAR TO MINUTE
DATETIME MONTH
DATETIME MONTH TO DAY
DATETIME MONTH TO HOUR
DATETIME MONTH TO SECOND
DATETIME DAY
DATETIME DAY TO HOUR
DATETIME DAY TO MINUTE
DATETIME DAY TO SECOND
DATETIME HOUR
DATETIME HOUR TO MINUTE
DATETIME MINUTE
```

```
DATETIME MINUTE TO SECOND
DATETIME SECOND
DATETIME FRACTION

INTERVAL YEAR(p)
INTERVAL YEAR(p) TO MONTH
INTERVAL MONTH(p)
INTERVAL DAY(p)
INTERVAL DAY(p) TO HOUR
INTERVAL DAY(p) TO MINUTE
INTERVAL DAY(p) TO SECOND
INTERVAL HOUR(p)
INTERVAL HOUR(p) TO MINUTE
INTERVAL HOUR(p) TO SECOND
INTERVAL MINUTE(p)
INTERVAL MINUTE(p) TO SECOND
INTERVAL SECOND(p)
```

注記: Java オブジェクトの種類および JDBC のターゲット タイプの間で実行される変換の詳細情報は、JDBC 3.0 仕様を参照してください: **http://java.cnam.fr/iagl/biblio/spec/jdbc-3_0-fr-spec.pdf**

NUMBER、VARCHAR2、DATE タイプ 2 のサポート

NUMBER、VARCHAR2、DATE タイプ 2 データは、JDBC タイプ 2 ドライバーによりサポートされます。

JDBC 変換

JDBC アプリケーションでは、setObject()、setDate()、setString()、setTime()、setTimeStamp() メソッドを使用して値をバインドでき、getObject()、getDate()、getTime()、getTimeStamp()、getString() メソッドを使用して値を取得できます。DATE タイプ 2 は、Java の java.sql.Timestamp オブジェクトにマッピングされます。DATE タイプ 2 列をバインドするために setTime() メソッドがコールされる場合、デフォルトの日付値は 1970-01-01 です。DATE タイプ 2 列をバインドするために setDate() メソッドが呼び出される場合、デフォルトの時間値は 00:00:00 AM です。

DC_MODE が 1 に設定されている場合、ResultSetMetadata メソッドで新しい DATE タイプ 2 列タイプが表示されます。

表 12: 設定メソッドの変換の例

オブジェクトタイプの例	SQL/MX に格納される値
java.sql.Date - 2016-09-23	2016-09-23 00:00:00
java.sql.Time - 00:21:56	1970-01-01 00:21:56
java.sql.Timestamp - 2016-09-23 17:03:50	2016-09-23 17:03:50

表 13: 取得メソッドの変換の例

SQL/MX からの例	取得メソッド - 表示される値
2016-09-23 17:03:50	getObject() - 2016-09-23 17:03:50.0

表は続く

2016-10-25 00:00:00	getDate() - 2016-10-25
2016-11-23 00:21:56	getTime() - 00:21:56
2016-09-23 17:03:50	getTimeStamp() - 2016-09-23 17:03:50.0
2016-10-25 17:10:56	getString() - 2016-10-25 17:10:56

留意事項

DATE_FORMAT セッションのデフォルト属性は、mxci インターフェイスおよび JDBC T2 ドライバーでサポートされています。ODBC または JDBC T2 アプリケーションから DATE_FORMAT を使用するには、フォーマット済み出力を保持する十分な長さのある CHARACTER 文字列として DATE type2 をキャストします。

浮動小数点のサポート

JDBC/MX ドライバーおよび NonStop Server for Java では、IEEE 754 浮動小数点形式で、FLOAT (32 ビット) 数または DOUBLE (64 ビット) 数を渡します。

浮動小数点値は、IEEE 754 値として、SQL/MX テーブルに格納されます。

浮動小数点値は、Tandem 形式 (TNS 形式とも呼ばれます)として SQL/MP テーブルに格納されます。Tandem 形式で SQL/MP テーブルに保存されている浮動小数点値について、SQL/MX は、値を格納する時には IEEE 754 形式から Tandem 形式に変換し、値を取得して渡すときは、Tandem 形式から IEEE 754 形式に変換を行います。

SQL/MX テーブルは、IEEE 754 浮動小数点値を格納しますので、浮動小数点データにアクセスする JDBC アプリケーションには、浮動小数点例外は受け取りません。JDBC アプリケーションは、プラス (+) またはマイナス (-) 無限大状態を確認して、オーバーフローまたはアンダーフローが発生したかを判断します。また、アプリケーションは、例えば、0 で除算された数字については、非数値を渡すことができます。この処理は、IEEE 754 標準に従って行われます。

SQL/MP テーブルは、浮動小数点例外を生成できます。

IEEE 754 形式および TNS 形式の浮動小数点値および倍精度値の範囲については、 **NonStop Server for Java プログラマ リファレンス**を参照してください。SQL/MX における浮動小数点数の形式に関する情報については、最新の **HPE NonStop SQL/MX リリース リファレンス マニュアル**の「データ型」を参照してください。

SQL エスケープ句

JDBC/MX は、SQL エスケープ句を受け入れ、それらを次の表に示すように同等の SQL/MX 句に変換します。

SQL エスケープ句	同等の SQL/MX 句
`{ d 'date-literal' }`	`DATE 'date-literal'`
`{ t 'time-literal' }`	`TIME 'time-literal'`
`{ ts 'timestamp-literal' }`	`TIMESTAMP 'timestamp-literal'`
`{ oj join-expression }`	`join-expression *`
`{ INTERVAL sign interval-string interval-qualifier }`	`INTERVAL sign interval-string interval-qualifier`
`{ fn scalar-function }`	`scalar-function`

表は続く

SQL エスケープ句	同等の SQL/MX 句
`{ escape 'escape-character' }`	`escape 'escape-character'`
`{ call procedure-name... }`	`CALL procedure-name...`
`{ ?=call procedure-name... }`	現在のリリースでサポートされていません

* JDBC 構文には、ネストした結合が含まれません。一方、SQL/MX には、ネストした結合が含まれます。

JDBC/MX は、外部結合用に SQL エスケープ構文を拡張します。

JDBC/MX タイプ 2 ドライバーの機能

JDBC/MX タイプ 2 ドライバーは、次の機能をサポートします。

- MX テーブルの場合、行サイズ制限が、最大ブロックサイズ 32768 に合わせて増やされます。
- 範囲およびハッシュ パーティション済み MX テーブルの場合、クラスタリングキー長さは、2048 バイトまで許容されます。
- 有符号および無符号値の場合、数値データタイプの精度は 128 桁まで拡張されます。

スレッド ライブラリ コンプライアンス

JDBC T2 ドライバーは、NSJ6 をサポートしています。NSJ6 と共に使用される場合、JDBC T2 ドライバーは、SPT スレッド ライブラリをロードします。この機能を構成するための処置は必要ありません。

JDBC トレース機能

JDBC トレース機能は、Java アプリケーションからコールされるすべての JDBC メソッドのエントリーポイントをトレースします。この機能を汎用的にするため、JDBC ドライバーのラッパーとして実装されます。

JDBC トレース機能は、データベースに対する JDBC 接続を取得できる次の方法のいずれかで有効化できます。

DriverManager クラスを使用するトレース

Java アプリケーションは、`DriverManager` クラスを使用して、JDBC 接続を取得し、JDBC トレース ドライバーをロードすることにより、JDBC トレース機能を有効化できます。`com.tandem.jdbc.TDriver` は、Driver インターフェイスを実装するトレース ドライバー クラスです。アプリケーションは、次の方法のいずれかで、JDBC トレース ドライバーをロードできます。

- JDBC トレース ドライバー クラスをコマンドラインの `-Djdbc.drivers` オプションで指定できます。
- アプリケーション内で `Class.ForName` メソッドを使用します。
- JDBC トレース クラスをアプリケーション内の `jdbc.drivers` プロパティに追加します。

ドライバー クラスの `getConnection` メソッドに渡される JDBC URL が、どの JDBC ドライバーが接続を取得するかを決定します。次の URL および JDBC ドライバーを使用して、JDBC 接続を取得します。

```
jdbc:sqlmx:
```

Java アプリケーションは、`DriverManager.setLogWriter` メソッドを使用して、トレースをオンに切り替える必要があります。例えば、アプリケーションで次の JDBC API コールを使用します。

```
DriverManager.setLogWriter(new PrintWriter(new FileWriter("FileName")));
```

データソース インプリメンテーションを使用するトレース

これは JDBC 接続を確立し、JDBC トレース機能を有効化するための好ましい方法です。この方法では、論理名は、Java ネーミングおよびディレクトリ インターフェイス (JNDI) を使用するネーミング サービスの手段により、トレース データソース オブジェクトにマッピングされます。

次の表では、トレース データソース オブジェクトに必要なプロパティのセットについて説明します。

プロパティ名	タイプ	description
dataSourceName	文字列	データソース名。
description	文字列	このデータソースの説明。
traceDataSource	文字列	トレースされる DataSource オブジェクトの名前。

traceDataSource オブジェクトを使用して、データベースへの JDBC 接続を取得します。DataSource インターフェイスの setLogWriter メソッドを使用して、Java アプリケーションが、トレースをオンに切り替える必要があります。

Java コマンドを使用するトレース

Java プログラムを起動するとき、次の引数を使用するシステムプロパティのトレースを指定することによって、トレースを有効化にします。

```
java -Djdbcmx.traceFile=logFile -Djdbcmx.traceFlag=n
```

logFile は、トレース情報が含まれるファイル名です。traceFlag に対する **n** 値を次の値にすることができます。

n に対する値	説明
0	トレースなし。
1	接続およびステートメント プーリング コールのみをトレースします。
2	LOB コード パスのみをトレースします。
3	すべての JDBC メソッドのエントリーポイントをトレースします。

注記: 同時に 1 つのみの traceFlag 値を有効にできます。

System.setProperty メソッドを使用するトレース

System.setProperty(key, value) を使用することにより、トレースを有効化して、上述と同じ値を設定します。例:

```
System.setProperty("traceFile", "myLogFile.log");
System.setProperty("traceFlag", "2");
```

プログラムが任意の JDBC API コールを行う前に、システムプロパティを設定します。

プログラム内でのトレース ドライバーをロードすることによるトレース

Class.forName("com.tandem.jdbc.TDriver") メソッドを使用して、プログラム内部で JDBC トレース ドライバーをロードすることにより、トレースを有効にします。また、この方法では、DriverManager.setLogWriter メソッドをコールする必要があります。

Wrapper データソースを使用するトレース

トレースするデータソースの周りにラッパー データソースを作成することにより、トレースを有効化します。ラッパー データソースには、トレースするデータソースに対して設定可能な TraceDataSource プロパティが含まれます。この方法を表示するデモ プログラムに関する情報については、**JDBC トレーシング機能デモプログラム**(96 ページ) を参照してください。

アプリケーションサーバーに対するトレースの有効化

通常、トレース出力は、DataSource.setLogWriter() メソッドまたは DriverManager.setLogWriter() メソッドを使用することにより、アプリケーションが設定する PrintWriter オブジェクトに書き込まれます。ユーザー作成の Java アプリケーションは、これらのメソッドを JDBC トレース機能と共に使用できます。

しかし、アプリケーションサーバーは、setLogWriter() メソッドを用いて、JDBC トレースを有効化する可能性はありません。代わりに、アプリケーションサーバーは、トレースを有効化して、次の JDBC/MX プロパティを使用することにより、トレース レベルを設定できます。

- **jdbcmx.traceFile プロパティ**(92 ページ)
- **jdbcmx.traceFlag プロパティ**(92 ページ)

jdbcmx.traceFile プロパティ

アプリケーションサーバーのトレースを有効にするには、コマンドラインで指定される jdbcmx.traceFile プロパティを使用します。

-Djdbcmx.traceFile=**trace_file_name**

ここで、trace_**file_name** は、OSS ファイル名です。ファイルが既に存在している場合、トレース出力は、既存ファイルに付加されます。

setLogWriter() メソッド使用して設定される PrintWriter オブジェクトは、jdbcmx.traceFile システムプロパティ設定よりも高い優先順位を有しています。このプロパティは、最初の接続前に、コマンドラインまたはプログラム的に指定できます。

jdbcmx.traceFlag プロパティ

jdbcmx.traceFile プロパティ使用するアプリケーションサーバーのトレース レベルを設定するには、コマンドラインで、指定される traceFlag プロパティを使用します。

-Djdbcmx.traceFlag=**n**

ここで、**n** は、トレース レベルを指定する整数です。値は、0、1、2、 3 にすることができます。デフォルト レベルは 0 です。3 より大きい任意の値は、3 のように扱われます。トレース レベルは、次のとおりです。

レベル	意味
0	トレースなし。
1	接続およびステートメント プーリング コールのみをトレースします。
2	LOB コード パスのみをトレースします。
3	すべての JDBC メソッドのエントリーポイントをトレースします。

注記:

同時に 1 つのみの `traceFlag` 値を有効にできます。

トレースファイル出力フォーマット

トレースファイルの開始時に、トレースされる JDBC/MX ドライバーの vproc を表示するトレース エントリーが出力されます。このエントリーは、`traceFlag` 値が 1、2、または 3 であるときのみ表示されます。例えば、

```
jdbcTrace:[05/02/12 01:05:24]:TRACING JDBC/MX VERSION: T1275R32_30AUG2012_JDBCMX_1209
```

トレース出力の形式には、2 つのタイプがあります。JDBC/MX ドライバがマッピングするオブジェクトを持つ場合のみ、2 番目のタイプが使用されます。形式は次のとおりです。

Format 1

```
jdbcTrace:[timestamp] [thread-id]:[object-id] :className.method(param...)
```

Format 2

```
jdbcTrace:[timestamp] [thread-id]:[object-id] :className.method(param...)
returns [return-object] [return-object-id]
```

ここで

timestamp

は、次の形式で、日付と時刻を表したものです: `mm/dd/yy hr:min:sec`

mm は月、`dd` は日、`yy` は年、`hr` は時、`min` は分、`sec` は秒です。

thread-id

は、現在のスレッドの文字列表現です。

object-id

は、JDBC オブジェクトのハッシュコードです。

classname

は、JDBC インプリメンテーション クラス名です。

return-object

は、トレースされる メソッドにより返されるオブジェクトです。return-object は、次のインターフェイスタイプの 1 つをとることができます。CallableStatement、Connection、PooledConnection、ResultSet、Statement、DatabaseMetaData、ParameterMetaData、または ResultSetMetaData。

return-object-id

は、トレースされるメソッドによって返されるオブジェクトのハッシュコードです。

トレース出力は、setLogWriter メソッドで指定される PrintWriter に送信されます。

```
jdbcTrace:[10/12/05 10:04:39]
[Thread[main,5,main]]:[5256233]:com.tandem.sqlmx.SQLMXPreparedStatement.executeQuery()
```

一部のトレース メソッドには、2 つのトレース文、メソッド エントリーポイントの 1 つおよびリターン オブジェクト マッピング用のエントリーポイントがあります。一部のコード パスでは、メソッド エントリーとリターンの間に追加のトレース文を記録する場合があります。例えば、SQLMXConnection.prepareStatement () トレース エントリーの間に、以下が表示されます。

```
jdbcTrace:[10/12/05 10:04:39]
[Thread[main,5,main]]:[10776760]:SQLMXConnection.prepareStatement("select c1, c2 from tconpool
where c1 = ?")
```

<additional trace entries>

```
<jdbcTrace:[10/12/05 10:04:39]
[Thread[main,5,main]]:[10776760]:SQLMXConnection.prepareStatement("select c1, c2 from tconpool
where c1 = ?") returns PreparedStatement [23276589]
```

SQL 文 ID と対応する JDBC SQL 文のログ

JDBC/MX ドライバーは、対応する JDBC SQL 文を用いてマッピングされる実行 SQL 文の SQL 文 ID (STMID) を表示する補足ログファイルを書き込むことができます。

idMapFile には、アプリケーションによって発行されるすべての SQL 文のリストが含まれ、それらを内部ドライバー STMTID (ハッシュ コード) に相関させます。トレースファイル出力 (トレースファイル出力を参照) は、STMID (トレース出力における **オブジェクト ID**) を列挙し、idMapFile トレースファイルにおける SQL 文を参照するために使用できます。

文 ID は、トレースファイルの **オブジェクト ID** が、すべてのエントリーについて、詳細および潜在的に大きな SQL 文に置換されることを避けるため、idMapFile に記録されます。

文 ID の SQL 文へのマッピングは、文を準備または実行する任意のインターフェイス、例えば、PreparedStatement、Connection、ResultSet、 JdbcRowSet、および Statement に適用されます。

- **文 ID ログの指定**(94 ページ)
- **文 ID ログのプロパティ**(95 ページ)
- **文 ID ログ出力**(96 ページ)

文 ID ログの指定

補足ログを指定するには:

- enableLog プロパティを ON に設定し、ログを有効化します。
- idMapFile プロパティを設定し、ログファイルを指定します。デフォルトでは、ログは、画面に書き込まれます。

これらのプロパティに関する追加情報については、**enableLog プロパティ**(95 ページ) および **idMapFile プロパティ**(95 ページ) を参照してください。

これらのプロパティをコマンドラインまたは 「Java コマンドを使用するトレース」および「system.setProperty メソッドを使用するトレース」において、以前に説明した設定トレースに類似するプログラムで指定できます。

> コマンドラインのログを指定します

```
java -Djdbcmx.idMapFile=logFile -Djdbcmx.enableLog=on
```

ログをプログラムで指定します

```
System.setProperty("enableLog", "on");
System.setProperty("idMapFile", "myMapFile.log");
```

文 ID ログのプロパティ

enableLog プロパティ

SQL 文の ID および対応する JDBC SQL 文のロギングを有効化します。enableLog プロパティの形式は、次のとおりです。

```
-Djdbcmx.enableLog=boolean
```

データタイプ: boolean

デフォルト: off

有効な値は、on または off です。このプロパティは、java コマンドラインのみで指定できます。

java コマンドラインの次の仕様が、ロギングを有効化します。

```
-Djdbcmx.enableLog=on
```

詳細情報については、**SQL 文 ID と対応する JDBC SQL 文のログ**(94 ページ)を参照してください。

idMapFile プロパティ

JDBC トレース機能が、SQL 文 ID および対応する JDBC SQL 文を記録するファイルを指定します。idMapFile プロパティの形式は、次のとおりです。

```
-Djdbcmx.idMapFile=filename
```

データタイプ: 文字列

デフォルト: 画面に記録します

有効な OSS ファイル名を指定します。このプロパティは、java コマンドラインのみで指定できます。

java コマンドラインの次のエントリーは、ファイル /sales/app5/STMID-Log に対するログの収集を指定します。

```
-Djdbcmx.idMapFile=/sales/app5/STMID-log
```

ログを有効にするには、enableLog プロパティを使用します。詳細情報については、**SQL 文 ID と対応する JDBC SQL 文のログ**(94 ページ) を参照してください。

文 ID ログ出力

文 ID ログ出力エントリーの形式は、次のとおりです。

```
[timestamp] STMTobject-id (sql-statement)
```

ここで

timestamp

は、次の形式で日付と時刻を表したものです: **mm/dd/yy hr:min:sec**

mm は月、**dd** は日、**yy** は年、**hr** は時、**min** は分、**sec** は秒です。

object-id

は、JDBC オブジェクトのハッシュ コードです。

sql-statement

は、文 ID にマップされた実際の SQL 文です。

例

```
[08/05/05 10:32:38] STMT16399041 ("insert into TST_TBL (c1) values = ?")
```

JDBC トレーシング機能デモ プログラム

JDBC/MX ドライバーは、インストールディレクトリにデモ プログラム jdbcTrace を提供します。プログラムについては、README_JDBCTrace ファイルで説明されています。場所については、**JDBC/MX ドライバー ファイルの位置**(17 ページ) を参照してください。これらのプログラムは、トレースするドライバー固有のデータソースの周りにラッパーを作成することによってトレースを説明します。詳細については、**見本プログラムのサマリー**(13 ページ) を参照してください。

トレースの制限事項

問題

JDBC/MX T2 ドライバーは、sqlmx.jar (SQL/MP 用 JDBC タイプ 2 ドライバー) および jdbcMx.jar (SQL/MX 用 JDBC タイプ 2 ドライバー) が CLASSPATH に含まれ、sqlmp.jar が jdbcMx.jar に CLASSPATH で先行している場合、例外 java.lang.IllegalAccessError を発行します。

例えば、

```
CLASSPATH=.:/usr/tandem/jdbcMp/T1277H10/lib/sqlmp.jar:/usr/tandem/jdbcMx/T1275H50/lib/jdbcMx.jar
```

原因

この問題は、同じパッケージとクラス com.tandem.jdbc.TPreparedStatement が、sqlmx.jar (SQL/MP 用 JDBC タイプ 2 ドライバー) と jdbcMx.jar (SQL/MX 用 JDBC タイプ 2 ドライバー) の両方で利用可能あるため発生します。

解決方法

このエラーを解決するには、jdbcMx.jar が sqlmp.jar に CLASSPATH で先行していることを確認します。例えば、

```
CLASSPATH=.:/usr/tandem/jdbcMx/T1275H50/lib/jdbcMx.jar :/usr/tandem/jdbcMp/T1277H10/lib/sqlmp.jar
```

メッセージ

JDBC/MX は、SQLException の `getErrorCode()` メソッドのエラーコードとして、sqlcode とファイル システム エラーコードを返します。

JDBC ドライバー (29000 ～ 29079 の範囲) の Java 部分からのメッセージ	JDBC ドライバー (29250 ～ 29499 の範囲) のネイティブ インターフェイス部分からのメッセージ
29001-29009 29050-29059 29010-29019 29060-29069 29020-29029 29070-29079 29030-29039 29080-29089 29040-29049	29251-29259 29260-29267 29269 - 29270

メッセージは、SQLCODE 順にリストされます。説明には、以下が含まれます。

`SQLCODE SQLSTATE` **メッセージ テキスト**
原因 [メッセージをトリガーするために何が発生したか。]
影響 [これが発生するときに、何が起きるか。]
リカバリ [問題を診断・修正する方法。]

これらの範囲外のエラーコードに関する情報について、最新の **HPE NonStop SQL/MX リリース メッセージ マニュアル**を参照してください。

JDBC/MX ドライバーの Java サイドからのメッセージ

29001 HYC00 サポートされていない機能 - {0}

原因: 列挙される機能は、JDBC ドライバーではサポートされません。

影響: 未サポート例外がスローされ、NULL resultSet が返されます。

リカバリ: プログラムから、機能を削除します。

29002 08003 接続が存在しません

原因: データベースへの接続がクローズされたとき、操作が試行されました。

影響: データベースにアクセスできません。

リカバリ: データベースへの接続が確立された後、操作を再試行します。

29003 HY000 文が存在しません

原因: 妥当性確認の試みがクローズされた文の getter または exec の呼び出しで行われました。

影響: getter または exec 呼び出し妥当性確認は失敗します。

リカバリ: 文が開いているとき、`validateGetInvocation()` または `validateExecDirectInvocation` を実行します。

29004 HY024 無効なトランザクションアイソレーション値

原因: トランザクションアイソレーション レベルを無効な値に設定しようとしました。

影響: `SQLMXConnection.setTransactionIsolation` が、トランザクションアイソレーション値を設定しません。

リカバリ: 有効な値は、次のとおりです。`SQL_TXN_READ_COMMITTED`、`SQL_TXN_READ_UNCOMMITTED`、`SQL_TXN_REPEATABLE_READ`、および `SQL_TXN_SERIALIZABLE`。アイソレーション値が指定されていない場合、デフォルトは `SQL_TXN_READ_COMMITTED` です。

29005 HY024 無効な ResultSet タイプ

原因: 無効な ResultSet タイプ値を設定しようとしました。

影響: resultSetType パラメーターを用いる SQLMX ステートメント コールは失敗します。

リカバリ: 有効な ResultSet タイプは、次のとおりです。TYPE_FORWARD_ONLY、TYPE_SCROLL_INSENSITIVE、および TYPE_SCROLL_SENSITIVE

29006 HY000 無効な結果セット同時実行

原因: 無効な結果セット同時実行値を設定しようとしました。

影響: resultSetConcurrency を用いる SQLMXStatement は失敗します。

リカバリ: 有効な resultSetConcurrency 値は、次のとおりです。CONCUR_READ_ONLY および CONCUR_UPDATABLE。

29007 07009 無効な記述子インデックス

原因: ResultSetMetadata 列パラメーターまたは ParameterMetaData param パラメーターが、記述子範囲外です。

影響: ResultSetMetadata または ParameterMetaData メソッド データが、期待通りに返されません。

リカバリ: メソッドに供給される列またはパラメーターを確認します。

29008 24000 無効なカーソル状態

原因: 接続が閉じられたとき、ResultSet メソッドがコールされました。

影響: メソッド コールが成功しません。

リカバリ: ResultSet メソッドをコールする前に、接続が開いていることを確認します。

29009 HY109 無効なカーソルの位置

原因: ResultSet 行カーソルが挿入行上にあったとき、deleteRow() メソッド、updateRow() メソッドまたは cancelRowUpdates メソッドを実行しようとしました。または、ResultSet 行カーソルが挿入行上になかったとき、insertRow() メソッドを実行しようとしました。

影響: 行の変更とカーソルの操作は成功しません。

リカバリ: 行を挿入するには、カーソルを挿入行に移動します。行を削除、キャンセル、または、行を更新するには、カーソルを挿入行から移動します。

29010 07009 無効な列名

原因: 列検索には、columnName 文字列が含まれません。

影響: 列の比較または検索は成功しません。

リカバリ: 有効な columnName 文字列を findColum()、validateGetInvocation()、および validateUpdInvocation() メソッドに提供します。

29011 07009 無効な列インデックスまたは記述子インデックス

原因: 有効範囲外の列パラメーターを含む ResultSet メソッドが実行されました。

影響: ResultSet メソッド データが、期待通りに返されません。

リカバリ: メソッドに提供される列を確認します。

29012 07006 制限付きデータタイプ属性違反

原因: 無効なデータタイプが設定されているとき、またはデータタイプが SQL 列タイプに一致しないとき、メソッドを実行しようとしました。

影響: インターフェイス メソッドは実行されません。

リカバリ: 列タイプに対して、適切なメソッドと Java データタイプが使用されていることを確認します。

29013 HY024 フェッチ サイズが 0 未満です

原因: フェッチする ResultSet.setFetchSize 行に対するサイズ設定が 0 未満です。

影響: ResultSet オブジェクトに対して更なる行が必要なとき、データベースからフェッチする必要がある行番号が設定されません。

リカバリ: setFetchSize() メソッドの行パラメーターをゼロより大きい値に設定します。

29014 HY000 SQL データタイプが認識されません

原因: JDBC により、認識されない SQL データタイプが検出されました。

影響: 例外がスローされます。データは更新されません。

リカバリ: SQL データタイプが JDBC によりサポートされていることを確認します。エラーは、JDBC/MX ドライバーに対する内部エラーです。

29015 HY024 無効なフェッチ方向

原因: SetFetchDirection() メソッドの方向パラメーターを無効な値に設定します。

影響: この ResultSet オブジェクトが処理される行の方向が設定されません。

リカバリ: 有効なフェッチ方向は、次のとおりです。ResultSet.FETCH_FORWARD、ResultSet.FETCH_REVERSE、および ResultSet.FETCH_UNKNOWN。

29016 22018 SQL 列 {0, number, integer}] データタイプを指定された Java データタイプに変換できません

原因: ResultSet.getLong() メソッドを使用して、非数字文字列を BigDecimal に変換しようとしました。

影響: 例外が報告され、データを取得できません。

リカバリ: 列が変換される有効なタイプであることを確認します。

29017 HY004 SQL データタイプが未サポートです

原因: サポートされていない SQL データタイプが、setter メソッドで検出されました。

影響: ARRAY、BINARY、BIT、DATALINK、JAVA_OBJECT、および REF データタイプはサポートされません。

リカバリ: JDBC setter メソッドでサポートされているデータタイプを使用します。

29018 22018 キャスト仕様における無効なキャラクター値

原因: 文字列を数値タイプに変換しようとしましたが、文字列が適切な形式ではありません。

影響: getter メソッドを通して取得した文字列は、メソッド タイプにキャストできません。

リカバリ: 互換性のあるタイプであることを確認するために、データベース内の文字列を確認します。

29019 07002 一連のパラメーター {1, number, integer} に対するパラメーター {0, number, integer} が設定されていません

原因: 入力記述子には、値が設定されていないパラメーターが含まれます。

影響: checkIfAllParamsSet() メソッドが、設置されていないパラメーターをレポートします。

リカバリ: 列挙されるパラメーターの値を設定します。

29020 07009 無効なパラメーターインデックス

原因: getter または setter メソッドのパラメーター数インデックスが有効な入力記述子の範囲外であるか、入力記述子範囲が NULL です。

影響: getter および setter メソッド呼び出し妥当性確認は失敗します。

リカバリ: getter および setter パラメーター インデックスを有効なパラメーター値に変更します。

29021 HY004 オブジェクトタイプが未サポートです

原因: プリペアード ステートメント `setObject()` メソッドコールにサポートされていないオブジェクト タイプが含まれています。

影響: `setObject()` メソッドは、指定されたパラメーターの値を設定しません。

リカバリ: 情報メッセージです。修正アクションは必要ありません。有効なオブジェクト タイプは、`null`、`BigDecimal`、`Date`、`Time`、`Timestamp`、`Double`、`Float`、`Long`、`Short`、`Byte`、`Boolean`、`String`、`byte[]`、`Clob`、および `Blob` です。

29022 HY010 関数シーケンス エラー

原因: `PreparedStatement.execute()` メソッドは、`PreparedStatement.addBatch()` メソッドの使用をサポートしません。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: `PreparedStatement.executeBatch()` メソッドを使用します。

29023 HY109 カーソルが 1 番目の行の前にあるため、データを取得できません。

原因: カーソルが 1 番目の行の前にあるとき、`getCurrentRow()` がコールされました。

影響: 例外がレポートされます。データは取得されません。

リカバリ: `getCurrentRow()` メソッドに対するアプリケーション コールを確認します。

29024 HY109 カーソルが最後の行の後ろにあります。これは、結果セットに行が含まれていないか、すべての行が取得されたためです。

原因: カーソルが最後の行の後ろにあるとき、`getCurrentRow()` がコールされました。

影響: 例外がレポートされます。データは取得されません。

リカバリ: `getCurrentRow()` メソッドに対するアプリケーション コールを確認します。

29025 22003 データ値 ({0}) は、列/パラメーター番号 {1, number, integer} の範囲外です

原因: 値が列データタイプに対して有効な範囲外にあるとき、データベースに値を設定する、または、データベースから値を取得しようとしました。

影響: 例外がスローされます。データは取得または更新されません。

リカバリ: 値が列タイプに対する有効範囲内にあることを確認します。

29026 の HY000 AutoCommit モードがオンであるとき、トランザクションをコミットまたはロールバックできません

原因: `AutoCommit` モードが有効であるとき、トランザクションをコミットしようとしました。

影響: トランザクションがコミットされません。

リカバリ: `AutoCommit` を無効化します。このメソッドは、`AutoCommit` モードが無効である場合にのみ使用します。

29027 HY011 トランザクションがアクティブであるため、SetAutoCommit は不可能です

原因: トランザクションがアクティブであるとき、`setAutoCommit()` モードをコールしようとしました。

影響: 現在、`AutoCommit` モードは変更されません。

リカバリ: トランザクションを完了し、`AutoCommit` モードの設定を試行します。

29028 22003 データ値 ({0}) は負ですが、列/パラメーター番号 {1, number, integer} は符号なしです

原因: 負の値を符号なし列に設定しようとしました。

影響: 例外がスローされます。データは更新されません。

リカバリ: 値が列タイプに対する有効範囲内にあることを確認します。

29030 22003 列/パラメーター値 {1,number,integer} 対するデータ値 ({0}) は切り上げられなければなりませんでした

原因: `SetBigDecimal()` メソッドが、列に挿入するため値を切り上げました。

影響: 値が切り上げられたことを示す、SQLWarning を発行します。データは、データベース列に入力されます。

リカバリ: なし。警告状態です。

29031 HY000 バッチ内の SQL SELECT 文が不正です

原因: `executeBatch()` メソッドで、SELECT SQL 文が使用されました。

影響: 例外がレポートされます。バッチ クエリで、`SELECT` SQL クエリを使用できません。

リカバリ: `executeQuery()` メソッドを使用して、`SELECT` SQL 文を実行します。

29032 23000 最後の読み取り以降、行が変更されました

原因: カーソルが挿入行の上にあるとき、`ResultSet` オブジェクト行を更新または削除しようとしました。

影響: `ResultSet` 行の変更は成功しません。

リカバリ: 行を更新または削除する前に、`ResultSet` オブジェクト カーソルを行から移動します。

29033 23000 プライマリキー列の値を更新できません

原因: テーブル内のプライマリキー列を更新しようとしました。

影響: 列は更新されません。

リカバリ: プライマリキー定義内の列を更新できません。列定義において、`NOT NULL` 句を省略した場合でも、NULL 値を含むことができません。

29035 HY000 IO 例外が発生しました {0}

原因: ASCII、バイナリまたは文字ストリームの setter または updater メソッドは、`java.io.IOException` を引き起こします。

影響: 指定した setter または updater メソッドは、ASCII、バイナリまたは文字ストリームを変更しません。

リカバリ: 情報メッセージです。修正アクションは必要ありません。

29036 HY000 サポートされていないエンコーディング {0}

原因: 文字エンコーディングは未サポートです。

影響: 要求された文字エンコードはサポートされていない場合は、例外がスローされます。

リカバリ: ASCII (ISO88591)、KANJI、KSC5601、および UCS2 のみが、サポートされる文字エンコーディングです。SQL/MP テーブルは、UCS2 文字エンコーディングをサポートしていません。

29037 HY106 ResultSet タイプは TYPE_FORWARD_ONLY です

原因: オブジェクト タイプが `TYPE_FORWARD_ONLY` として設定されているとき、`ResultSet` カーソルが前の行をポイントしようとしました。

影響: `ResultSet` オブジェクト カーソル操作は発生しません。

リカバリ: `TYPE_FORWARD_ONLY ResultSet` オブジェクト タイプ カーソルは、前方にのみ移動できます。`TYPE_SCROLL_SENSITIVE` および `TYPE_SCROLL_INSENSITIVE` タイプはスクロール可能です。

29038 HY107 行番号が無効です

原因: 行番号が 0 に設定されていたとき、`ResultSet absolute()` メソッドがコールされました。

影響: カーソルは、指定された行番号に移動しません。

リカバリ: 正の行番号を (結果セットの先頭から数えた行番号を指定) を提供するか、負の行番号 (結果セットの終了から数えた行番号を指定) を提供します。

29039 HY092 ResultSet の同時処理モードは CONCUR_READ_ONLY です

原因: 同時処理が `CONCUR_READ_ONLY` に設定されているため、更新できない `ResultSet` オブジェクト上で操作を行おうとしました。

影響: `ResultSet` オブジェクトは変更されません。

リカバリ: 更新の場合、`ResultSet` オブジェクトの同時処理を `CONCUR_UPDATABLE` に設定しなければなりません。

29040 HY000 操作が無効です。現在の行は、挿入行です

原因: 現在の挿入行の情報を取得、更新、削除、または挿入しようとしました。

影響: `ResultSet` 行の情報の取得に成功しません。

リカバリ: 行の情報を取得するには、`ResultSet` オブジェクト カーソルを挿入行から移動します。

29041 HY000 操作が無効です。テーブルのプライマリキーがありません

原因: プライマリキーキーを定義せずに作されたテーブル上で `getKeyColumns()` メソッドは失敗します。

影響: テーブルのプライマリキー データは返されません。

リカバリ: プライマリキー列を含めるようにテーブルを変更します。

29042 HY000 フェッチ サイズ値が無効です

原因: 0 よりも小さい値をフェッチ行サイズを設定しようとしました。

影響: 複数の行を必要とするデータベースからフェッチする行番号が設定されていません。

リカバリ: `setFetchSize()` メソッドの場合、0 以上の有効な行の値を提供します。

29043 HY000 最大行の値が無効です

原因: `ResultSet` オブジェクトが含む最大行番号に 0 未満の制限を設定しようとしました。

影響: 行の最大番号の制限が設定されません。

リカバリ: `setMaxRows()` メソッドの場合、0 以上の有効な値を提供します。

29044 HY000 クエリ タイムアウト値が無効です

原因: ドライバーが `Statement` オブジェクトの実行を待機する秒数に 0 未満の値を設定しようとしました。

影響: クエリ タイムアウトの制限が設定されません。

リカバリ: `setQueryTimeout()` メソッドの場合、0 以上の有効な行の値を提供します。

29045 01S07 端数切り捨て

原因: `ResultSet` getter メソッドにより取得されたデータが切り捨てられました。

影響: 取得されたデータが切り捨てられます。

リカバリ: 取得されるデータが有効なデータタイプ範囲内にあることを確認します。

29046 22003 数値が範囲外です

原因: `ResultSet` getter メソッドが取得した値が、データタイプの範囲外です。

影響: `ResultSet` getter メソッドは、データを取得しません。

リカバリ: 取得されるデータが有効なデータタイプ範囲内にあることを確認します。

29047 HY000 バッチ更新に失敗しました。詳細については、次の例外を参照してください

原因: バッチ更新のコマンドの 1 つが適切な実行に失敗しました。

影響: バッチ更新コマンドの一部が失敗します。詳細情報については、後続の例外を参照してください。

リカバリ: 可能なリカバリ操作については、後続の例外を参照してください。

29048 HY009 無効な NULL の使用

原因: 期待されるテーブル名を含むパラメーターが NULL に設定されています。

影響: `DatabaseMetadata` メソッドは、結果を報告しません。

リカバリ: `DatabaseMetaData` メソッドの場合、NULL ではない、有効なテーブル名を提供します。

29049 25000 無効なトランザクション状態

原因: トランザクションが進行中だったとき、`begintransaction()` メソッドがコールされました。

影響: 新しいトランザクションは開始されません。

リカバリ: `begintransaction()` メソッドをコールする前に、他のトランザクションが現在起動されているかどうかを確認します。

29050 HY107 行の値が範囲外です

原因: 取得された `getCurrentRow` に対するコールは、最初と最後の行の範囲外です。

影響: 現在の行は取得されません。

リカバリ: これは情報メッセージのみです。リカバリを行う必要はありません。サービスプロバイダに全体のメッセージがレポートされます。

29051 01S02 ResultSet タイプが TYPE_SCROLL_INSENSITIVE に変更されました

原因: 結果セット タイプが変更されました。

影響: なし。

リカバリ: このメッセージは、SQL 警告として報告されます。これは情報メッセージのみです。リカバリを行う必要はありません。

29052 22003 カラム/パラメーター番号 {1,number,integer} のタイムスタンプ ({0}) が形式 yyyy-mm-dd hh:mm:ss.fffffffff ではありません

原因: `TIMESTAMP` 列タイプに無効なタイムスタンプ形式を入力しようとしました。

影響: 例外がスローされます。データは更新されません。

リカバリ: タイムスタンプの形式として `yyyy-mm-dd hh:mm:ss.fffffffff` が使用されていることを確認します。

29053 HY000 SQL SELECT 文が executeUpdate() メソッドで無効です

原因: `executeUpdate()` メソッドで、SELECT SQL 文が使用されました。

影響: `SQL` クエリが実行されません例外が報告されます。

リカバリ: `executeQuery()` メソッドを使用して、SELECT SQL 文を実行します。

29054 HY000 SQL SELECT 文のみが executeQuery() 方式で有効です

原因: `executeQuery()` メソッドで、非 SELECT SQL 文が使用されました。

影響: 「`SQL` クエリが実行されません」例外が報告されます。

リカバリ: `executeUpdate()` メソッドを使用して、非 SELECT SQL 文を実行します。

29055 22003 カラム/パラメーター番号 {1,number,integer} の日付 ({0}) が形式 yyyy-mm-dd ではありません

原因: `DATE` 列タイプに無効な日付形式を入力しようとしました。

影響: 例外がスローされます。更新されません。

リカバリ: 日付の形式として yyyy mm-dd が使用されていることを確認します。

29056 HY000 文が既に閉じられています

原因: 文が閉じられるとき、validateSetInvocation() または validateExecuteInvocation メソッドが使用されました。

影響: 文の妥当性確認は失敗し、例外を返します。

リカバリ: 文を閉じる前に、validateSetInvocation() または validateExecuteInvocation メソッドを使用します。

29057 HY000 自動的に生成されたキーはサポートされません

原因: 自動生成されたキーの機能を使用しようとしました。

影響: 試みが成功しません。

リカバリ: 自動生成されたキーの機能はサポートされません。

29058 HY000 接続は PooledConnection オブジェクトに関連付けられていません

原因: PooledConnection オブジェクトが確立される前に、getPooledConnection() メソッドが呼び出されました。

影響: プールからの接続を取得できません。

リカバリ: getPooledConnection() メソッドを使用する前に、PooledConnection オブジェクトが確立されていることを確認してください。

29059 HY000 「blobTableName」プロパティが設定されていない、NULL 値または無効な値に設定されています

原因: blobTableName プロパティを設定せず、BLOB 列にアクセスしようとした、もしくはプロパティが無効な値に設定されています。

影響: アプリケーションが、BLOB 列にアクセスできません。

リカバリ: blobTableName プロパティを有効な LOB テーブル名に設定します。LOB テーブル名は、catalog.schema.lobTableName の形式です。

29060 HY000 「clobTableName」プロパティが設定されていない、NULL 値または無効な値に設定されています

原因: clobTableName プロパティが設定されていない、NULL 値または無効な値に設定されています。

影響: アプリケーションが、CLOB 列にアクセスできません。

リカバリ: clobTableName プロパティを有効な LOB テーブル名に設定します。LOB テーブル名は、catalog.schema.lobTableName の形式です。

29061 HY000 Lob オブジェクト {0} が最新ではありません

原因: jdbcmx.clobTableName プロパティを設定せず、CLOB 列にアクセスしようとすると、プロパティは無効に設定されます。

影響: アプリケーションが、CLOB 列にアクセスできません。

リカバリ: jdbcmx.clobTableName プロパティを有効な LOB テーブル名に設定します。LOB テーブル名は、catalog.schema.lobTableName の形式です。

29062 HY000 プライマリキーが選択リストにないため、操作は許可されません

原因: プライマリキー列なしで作成されたテーブル上での getKeyColumns() は失敗します。

影響: テーブルのプライマリキー データは返されません。

リカバリ: テーブルをプライマリキー列を含むように変更するか、getKeyColumns() メソッド コールをプログラムから削除します。

29063 HY000 トランザクションのエラー {0} - {1} 開始データ ロケーターの取得中

原因: LOB 列の挿入または更新中に、JDBC/MX ドライバーが、与えられたプロセスに対するデータ ロケーターを予約しようとすると、トランザクション エラーが発生します。

影響: アプリケーションは、LOB 列を挿入または更新できません。

リカバリ: メッセージのファイルシステムエラーを確認し、適切なリカバリ操作を実施します。

29064 22018 Java データタイプが列の SQL データタイプと一致しません

原因: 無効な列データタイプを含む PreparedStatement setter メソッドをコールしようとしました。

影響: 例外がスローされます。データは更新されません。

リカバリ: 列データタイプが、PreparedStatement setter メソッドに対して有効であることを確認します。

29065 22018 Java データタイプを指定された SQL データタイプに変換できません

原因: 与えられた SQL データタイプに対する PreparedStatement setter メソッドの Java オブジェクトの変換が無効です。

影響: 例外がスローされます。データは更新されません。

リカバリ: 列データタイプが、PreparedStatement setter メソッドに対して有効であることを確認します。

29066 22018 文字列データ {0} を数値に変換できません

原因: PreparedStatement setter メソッドが、文字列を整数に変換できませんでした。

影響: 例外がスローされ、文字列データは変換されません。

リカバリ: 整数に変換するデータが有効であることを確認します。

29067 07009 メソッド {0} で無効な値が入力されたました

原因: 与えられたの方法で 1 つまたは複数の入力値は無効です。

影響: 与えられた入力メソッドが失敗しました。

リカバリ: 与えられたメソッドに対する入力値を確認します。

29068 07009 位置の値は 1 以上の LOB データの長さの間で任意の値を指定できます

原因: Blob.setBinaryStream、Clob.setCharacterStream、または Clob.setAsciiStream の位置の入力値は、1 以上の LOB データの長さの間にすることができます。

影響: アプリケーションは、LOB データを指定された位置に書き込むことができません。

リカバリ: 位置の入力値を修正します。

29069 HY000 自動コミットがオンで、LOB オブジェクトが関連しています

原因: LOB データが有効になっている自動コミットと関連して、外部のトランザクションが存在しません。

影響: 例外がレポートされます。LOB 列が設定されません。

リカバリ: Clob.setASciiStream()、Clob.setCharacterStream()、または Blob.setBinaryStream() メソッドを使用するときは、外部のトランザクションを開始するか、自動コミット モードを無効します。

29070 HY000 トランザクション エラー {0} - {1} LOB テーブルの更新中

原因: 内部トランザクション内の実テーブルまたは LOB テーブル上での挿入または更新操作中に、SQL またはファイル システム (FS) 例外が発生しました。

影響: 例外がレポートされます。内部トランザクションはロールバックされます。

リカバリ: SQL または FS のエラーメッセージを参照してください。

29071 HY000 内部プログラミング エラー - {0}

原因: JNI レイヤー (get オブジェクト メソッド) は、常にバイト配列を返します。したがって、他のインスタンスは、プログラミング エラーと見なされます。

影響: 例外が報告されます。

リカバリ: なし。エラーは、JDBC/MX ドライバーに対する内部エラーです。

29072 HY000 最大接続プール サイズ ({0,number,integer}) を超えようとしました

原因: 設定された接続プール サイズ制限の外の接続を取得しようとしました。

影響: 例外がスローされます。

リカバリ: `maxPoolSize` コマンドライン プロパティを使用することにより、接続プール サイズを大きくします。

29073 22003 カラム/パラメーター番号 {1,number,integer} の時刻 ({0}) が形式 hh:mm:ss ではありません

原因: `TIME` 列タイプに無効な時刻形式を入力しようとしました。

影響: 例外がスローされます。データは更新されません。

リカバリ: 時刻の形式として `hh:mm:ss` が使用されていることを確認します。

29074 42821 getter メソッド {0} は 列/パラメーター番号 {1, number, integer} のデータ取得に使用できません

原因: サポートされていない列タイプを `ResultSet.getString` メソッドで使用しようとしました。

影響: 例外が報告され、データを取得できません。

リカバリ: `BLOB`、`ARRAY`、`REF`、`STRUCT`、`DATALINK`、または `JAVA_OBJECT` 以外のサポートされる列データタイプを `ResultSet.getString` メソッドと共に使用します。

29075 HY000 「transactionMode」プロパティに NULL または無効な値を設定しました

原因: 無効なトランザクション モードを使用して、`SQLMXDataSource.setTransactionMode()` または `SQLMXConnectionPoolDataSource.setTransactionMode()` をコールしました。

影響: アプリケーションは、トランザクションモードを設定できません。

リカバリ: 有効なトランザクションモード (外部、内部、または混在) を使用します。

29076 HY000 「maxStatements」 ({0, number, integer}) を超過しました - 性能が低下します。

原因: キャッシュされた文の数が、`maxStatements` プロパティにより設定される制限に到達しました。すべての文は使用中です。

影響: SQL の警告状態。文を引き続き内部キャッシュに追加します。

リカバリ: SQL の警告状態。`maxStatements` プロパティ (または `-Djdbcmx.maxStatements` コマンドライン プロパティ) を使用して、許可される文の数を増やします。

29077 HY000 最大行の値をフェッチ サイズより小さくできません

原因: `JdbcRowSet.setMaxRows` メソッドに渡される行の値が、現在のフェッチ サイズ設定より小さい値です。

影響: `JdbcRowSet` オブジェクトが含むことができる行の最大番号が設定されていません。

リカバリ: `JdbcRowSet.setFetchSize` を使用することにより、フェッチ サイズ値を増やすか、`JdbcRowSet.setMaxRows` メソッドに渡す最大行の値を増やします。

29078 HY000 無効な JdbcRowSet 状態 - {0} {1}

原因: JdbcRowSet 操作に関連する Connection、ResultSet、 または PreparedStatement の値が NULL です。

影響: メソッド コールが失敗します。

リカバリ: JdbcRowSet.execute() メソッドをコールしていることを確認します。

29079 HY000 列が設定された列に一致しません

原因: unsetMatchColumn() メソッドに渡される指定された列が、以前に設定された列に一致しません。

影響: 指定された列が、この JdbcRowSet オブジェクトに対して未設定ではありません。

リカバリ: setMatchColumn() メソッドを使用して、指定された列を一致する列として設定します。

29080 HY000 取得する前に一致する列を設定します

原因: getMatchColumnNames()、 getMatchColumnIndexex()、または unsetMatchColumn() メソッドに対するコールが、NULL を返すか、列に一致します。

影響: 一致する列の値が、この JdbcRowSet オブジェクトに取得されません。

リカバリ: setMatchColumn() を使用して、指定された列を一致する列として設定します。

29081 HY000 一致する列は 0 より大きくする必要があります

原因: プログラムが、ゼロより小さい列インデックス値を setMatchColumn() メソッドに渡しました。

影響: この JdbcRowSet オブジェクトに対して指定された一致する列が設定されていません。

リカバリ: ゼロより大きい有効な列インデックス値を用いて、setMatchColumn() メソッドをコールします。

29082 HY000 一致する列を NULL または空の文字列にすることはできません

原因: NULL または空の列名の文字列が、setMatchColumn() メソッドに渡されました。

影響: この JdbcRowSet オブジェクトに対して指定された一致する列が設定されていません。

リカバリ: 有効な NULL ではない列名文字列を用いて setMatchColumn()メソッドをコールします。

29083 HY000 未設定ではない列がセットされた列と同じではありません

原因: unsetMatchColumn() メソッドに渡される指定された列が、以前に設定された列に一致しません。

影響: 指定された列が、この JdbcRowSet オブジェクトに対して未設定ではありません。

リカバリ: setMatchColumn() メソッドを使用して、指定された列を一致する列として設定します。

29084 HY000 列名を unsetMatchColumn に対する引数として使用します

原因: 列名の文字列の値は、unsetMatchColumn(integer i) メソッドに渡されます。

影響: 例外がスローされます。この JdbcRowSet オブジェクトに対して指定された一致する列が未設定ではありません。

リカバリ: 整数の列 ID 値を用いて unsetMatchColumn(integer i) をコールするか、unsetMatchColumn (String s) メソッドを使用します。

29085 HY000 列 ID を unsetMatchColumn に対する引数として使用します

原因: 列名の整数値は、unsetMatchColumn(String s) メソッドに渡されます。

影響: 例外がスローされます。この JdbcRowSet オブジェクトに対して指定された一致する列が未設定ではありません。

リカバリ: 文字列の列名の値を用いて unsetMatchColumn(String s) をコールするか、unsetMatchColumn(integer i) メソッドを使用します。

29086 HY000 JdbcRowSet パラメーター ({0, number, integer}) が欠落しています

原因: 内部ドライバーの状態により、JdbcRowSet パラメーターに設定値がないことが検出されました。

影響: JdbcRowSet.execute() メソッドは失敗します。

リカバリ: なし。エラーは、JDBC/MX ドライバーに対する内部エラーです。

29087 HY000 JdbcRowSet setProperties エラー {0} - {1}

原因: 内部ドライバーの状態により、メッセージでレポートされる JdbcRowSet プロパティが、JdbcRowSet プリペアードステートメントに対して設定できなかったことが検出されました。

影響: JdbcRowSet.execute() メソッドは失敗します。

リカバリ: エラーは、JDBC/MX ドライバーに対する内部エラーです。

29088 HY000 JdbcRowSet 準備エラー - {0}

原因: JdbcRowSet プリペアードステートメント準備中、ドライバーに内部エラーが発生しました。

影響: 例外が報告されます。

リカバリ: なし。エラーは、JDBC/MX ドライバーに対する内部エラーです。

29089 HY000 JdbcRowSet 接続エラー - {0} {1}

原因: ドライバーの接続を確立する際に内部エラーが発生しました。

影響: 例外が報告されます。

リカバリ: なし。エラーは、JDBC/MX ドライバーに対する内部エラーです。

JDBC/MX ドライバーの JNI サイドからのメッセージ

29251 HY000 プログラミング エラー

原因: SQL が 文の SQL パラメーターの 1 つにエラーを検出しました。または、SQL が 試行した操作にエラーを返しましたが、JDBC により処理されません。

影響: 例外が報告されます。

リカバリ: SQL パラメーター エラーの場合、例外メッセージのテキストは通常、修正すべき問題を識別します。未処理の SQL エラーの場合は、例外のエラーコードは、捕捉された SQL エラーを識別します。エラーコードの詳細については、SQL エラーメッセージ ドキュメントを参照してください。

29252 HY008 操作はキャンセルされました

原因: SQL 操作は、ブレークによってキャンセルされました。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: このメッセージは、アプリケーション固有です。文を再度実行します。

29253 22003 数値が範囲外です

原因: 数値が、ターゲット列の範囲内にありません。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: SQL の列タイプの有効な範囲に数値を調整します。

29254 22001 文字列データの右切り捨てが発生しました

原因: データベースに文字列を設定しようとしましたが、文字列は、データベースの制限を超えています。

影響: 一部のデータはデータベースに格納されません。

リカバリ: 文字列の長さを短くします。

29255 HY000 TMF エラーが発生しました: [tmf-error]

原因: 内部トランザクション要求が失敗しました。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: TMF エラーメッセージ **tmf-error** を参照してください。

29256 HY000 システム カタログ名取得中にエラーが発生しました: [error]

原因: JDBC ドライバーの初期化中、システム カタログ名を決定する際にエラーが発生しました。

影響: JDBC ドライバーが、ドライバー マネージャーに登録されません。

リカバリ: SQL がインストールされていること、システム カタログが存在することを確認します。

29257 07002 すべてのパラメーターが設定されていません

原因: 読み取られたパラメーターが NULL でした。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: Null 以外のパラメーター値を入力します。

29258 25000 無効なトランザクション状態

原因: TMF を介してトランザクションを開始または再開する際に、トランザクション状態の問題が検出されました。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: 情報メッセージです。修正アクションは必要ありません。サービスプロバイダに全体のメッセージがレポートされます。

29259 HY000 モジュール エラー

原因: モジュールからカタログ情報を取得または文を準備しようとするとき、無効なパラメーターが検出されました。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: リカバリの詳細については、例外メッセージを参照してください。

29260 HY000 無効な文/接続ハンドル

原因: 無効な SQL 文ハンドルが検出されました。

影響: 例外がレポートされます。操作が完了していません。

リカバリ: 情報メッセージです。修正アクションは必要ありません。サービスプロバイダに全体のメッセージがレポートされます。

29261 HY000 SQL/MX 診断領域にエラーメッセージはありませんが、SQLCODE が ゼロではありません

原因: SQL エラーが検出されましたが、SQL/MX によって、エラーメッセージが報告されませんでした。

影響: 診断メッセージを表示せずに、SQL 例外または警告がスローされます。

リカバリ: 不明。

29262 HY090 無効または NULL の SQL 文字列

原因: ストアド プロシージャーおよびプリペアード ステートメントコールに無効な SQL 文字列が含まれています。

影響: ストアド プロシージャーまたはプリペアード ステートメントは実行されません。

リカバリ: ストアド プロシージャーまたはプリペアード ステートメントに有効な SQL コマンドが含まれていることを確認します。

29263 HY000 無効または NULL の文のラベルまたは名前

原因: データベースをコールするストアド プロシージャーまたはプリペアード ステートメントに、無効な文ラベルの入力パラメーターがあります。

影響: ストアド プロシージャーまたはプリペアード ステートメントは実行されません。

リカバリ: 文ラベル パラメーターが有効であることを確認します。

29264 HY000 無効または NULL のモジュール名

原因: データベースをコールするストアド プロシージャーまたはプリペアード ステートメントに、無効なモジュール名の入力パラメーターがあります。

影響: ストアド プロシージャーまたはプリペアード ステートメントは実行されません。

リカバリ: モジュール名パラメーターが有効であることを確認します。

29265 HY000 サポートされていないキャラクターセット エンコーディング

原因: 文字セットの種類、 CHAR、VARCHAR、VARCHAR_LONG、または VARCHAR_WITH_LENGTH 列のキャラクターセット タイプは、JDBC/MX ドライバーの setter または getter メソッドによってサポートされていません。

影響: SQL 例外がスローされます。

リカバリ: 列のキャラクターセット タイプを JDBC/MX ドライバーによってサポートされるタイプに変更します。

29266 HY000 データタイプは未サポートです: [data type]

原因: サポートされていないデータタイプが、SQL から取得されました。

影響: 例外がスローされます。

リカバリ: なし。BIT、BITVAR、BPINT_UNSIGNED、SQLTYPECODE_FLOAT、SQLTYPECODE_REAL、および SQLTYPECODE_DOUBLE データタイプが SQL/MX から返されることは期待されません。JDBC/MX ドライバーは、これらのデータタイプをサポートしません。

29267 HY000 過剰な JVM メモリが割り当てられています

原因: JDBC が、使い果たした後、内部的に JVM メモリを割り当てようとしました。

影響: 条件は、JVM のヒープサイズの関数です。例外がスローされます。

リカバリ: 最大 JVM のヒープサイズを適切に構成します。

29269 X0T7H ユーザー認証に失敗しました [[リターンコード *retcode*] | [ステータス *status_num*] | [追加情報]]

ここで、

retcode - Guardian API から返されるエラー番号。

status_num - *retcode* が 48 の場合に返されるステータス コード。

原因: 接続要求に対して提供されたユーザー認証情報が無効であるか、認証サブシステムが、要求の処理中にエラーを返しました。

影響: データベース接続要求が処理されません。例外は、アプリケーションにエラーメッセージと共に返されます。

リカバリ: *retcode* と *status_num* を確認して、認証失敗の原因を確認します。Guardian プロシージャーコール リファレンスマニュアル に USER_AUTHENTICATE_ API のステータスおよびエラーコードが説明されています。ユーザー認証情報が正しい場合、データベース接続要求およびサブシステム エラーにより失敗した接続を再試行してください。

29270 X0T7I 定義 =_JDBCMX_AUTHENTICATION 処理に失敗しました [[定義が MAP 定義ではないか、指定された FILE 値に誤りがあります] | [エラー *error-num*]]。

ここで、

error-num – 返されるエラー番号。

原因: =_JDBCMX_AUTHENTICATION 定義が MAP 定義ではないか、定義に指定された値が ON または OFF ではありません。

影響: データベース接続要求が処理されません。アプリケーションには、例外が返されます。

リカバリ: 定義タイプまたは値を修正し、アプリケーションを再起動します。

JDBC ドライバー (29000 ～ 29249 の範囲) の Java 部分からのメッセージ	JDBC ドライバー (29250 ～ 29499 の範囲) のネイティブ インターフェイス部分からのメッセージ
29001-29009 29050-29059 29010-29019 29060-29069 29020-29029 29070-29079 29030-29039 29080-29089 29040-29049	29251-29259 29260-29267 29269 - 29270

Web サイト

全般的な Web サイト

Hewlett Packard Enterprise Information Library

http://www.hpe.com/info/EIL

Hewlett Packard Enterprise サポートセンター

http://www.hpe.com/support/hpesc

Contact Hewlett Packard Enterprise Worldwide

http://www.hpe.com/assistance

サブスクリプションサービス/サポートのアラート

http://www.hpe.com/support/e-updates-ja

Software Depot

http://www.hpe.com/support/softwaredepot

カスタマーセルフリペア

http://www.hpe.com/support/selfrepair

L シリーズのマニュアル

http://www.hpe.com/info/nonstop-ldocs

J シリーズのマニュアル

http://www.hpe.com/info/nonstop-jdocs

上記以外の Web サイトについては、**サポートと他のリソース**を参照してください。

サポートと他のリソース

Hewlett Packard Enterprise サポートへのアクセス

- ライブアシスタンスについては、Contact Hewlett Packard Enterprise Worldwide の Web サイトにアクセスします。

  http://www.hpe.com/assistance

- ドキュメントとサポートサービスにアクセスするには、Hewlett Packard Enterprise サポートセンターの Web サイトにアクセスします。

  http://www.hpe.com/support/hpesc

ご用意いただく情報

- テクニカルサポートの登録番号（該当する場合）
- 製品名、モデルまたはバージョン、シリアル番号
- オペレーティングシステム名およびバージョン
- ファームウェアバージョン
- エラーメッセージ
- 製品固有のレポートおよびログ
- アドオン製品またはコンポーネント
- 他社製品またはコンポーネント

アップデートへのアクセス

- 一部のソフトウェア製品では、その製品のインターフェイスを介してソフトウェアアップデートにアクセスするためのメカニズムが提供されます。ご使用の製品のドキュメントで、ソフトウェアの推奨されるソフトウェアアップデート方法を確認してください。
- 製品のアップデートをダウンロードするには、以下のいずれかにアクセスします。

  Hewlett Packard Enterprise サポートセンター

  http://www.hpe.com/support/hpesc

  Hewlett Packard Enterprise サポートセンター：ソフトウェアのダウンロード

  http://www.hpe.com/support/downloads

  Software Depot

  http://www.hpe.com/support/softwaredepot

- eNewsletters およびアラートをサブスクライブするには、以下にアクセスします。

http://www.hpe.com/support/e-updates-ja

- お客様の資格を表示したりアップデートしたり、契約や保証をお客様のプロファイルにリンクしたりするには、Hewlett Packard Enterprise サポートセンターの **More Information on Access to Support Materials** ページにアクセスします。

  http://www.hpe.com/support/AccessToSupportMaterials

重要: 一部のアップデートにアクセスするには、Hewlett Packard Enterprise サポートセンターからアクセスするときに製品資格が必要になる場合があります。関連する資格を使って HPE パスポートをセットアップしておく必要があります。

カスタマーセルフリペア（CSR）

Hewlett Packard Enterprise カスタマーセルフリペア（CSR）プログラムでは、ご使用の製品をお客様ご自身で修理することができます。CSR 部品を交換する必要がある場合、お客様のご都合のよいときに交換できるよう直接配送されます。一部の部品は CSR の対象になりません。Hewlett Packard Enterprise もしくはその正規保守代理店が、CSR によって修理可能かどうかを判断します。

リモートサポート（HPE 通報サービス）

リモートサポートは、保証またはサポート契約の一部としてサポートデバイスでご利用いただけます。リモートサポートは、インテリジェントなイベント診断を提供し、ハードウェアイベントを Hewlett Packard Enterprise に安全な方法で自動通知します。これにより、ご使用の製品のサービスレベルに基づいて、迅速かつ正確な解決が行われます。ご使用のデバイスをリモートサポートに登録することを強くおすすめします。

ご使用の製品にリモートサポートの追加詳細情報が含まれる場合は、検索を使用してその情報を見つけてください。

リモートサポートおよびプロアクティブケア情報

HPE 通報サービス

http://www.hpe.com/jp/hpalert

HPE プロアクティブケアサービス

http://www.hpe.com/services/proactivecare-ja

HPE プロアクティブケアサービス：サポートされている製品のリスト

http://www.hpe.com/services/proactivecaresupportedproducts

HPE プロアクティブケアアドバンストサービス：サポートされている製品のリスト

http://www.hpe.com/services/proactivecareadvancedsupportedproducts

保証情報

ご使用の製品の保証またはサーバー、ストレージ、電源、ネットワーク、およびラック製品の安全と準拠に関する情報に関するドキュメントを確認するには、下記の Web サイトを参照してください。

http://www.hpe.com/support/Safety-Compliance-EnterpriseProducts

追加保証情報

HPE ProLiant と x86 サーバーおよびオプション

http://www.hpe.com/support/ProLiantServers-Warranties

HPE エンタープライズサーバー

http://www.hpe.com/support/EnterpriseServers-Warranties

HPE ストレージ製品

http://www.hpe.com/support/Storage-Warranties

HPE ネットワーク製品

http://www.hpe.com/support/Networking-Warranties

規定に関する情報

安全、環境、および規定に関する情報については、Hewlett Packard Enterprise サポートセンターからサーバー、ストレージ、電源、ネットワーク、およびラック製品の安全と準拠に関する情報を参照してください。

http://www.hpe.com/support/Safety-Compliance-EnterpriseProducts

規定に関する追加情報

Hewlett Packard Enterprise は、REACH（欧州議会と欧州理事会の規則 EC No 1907/2006）のような法的な要求事項に準拠する必要に応じて、弊社製品の含有化学物質に関する情報をお客様に提供することに全力で取り組んでいます。この製品の含有化学物質情報レポートは、次を参照してください。

http://www.hpe.com/info/reach

RoHS、REACH を含む Hewlett Packard Enterprise 製品の環境と安全に関する情報と準拠のデータについては、次を参照してください。

http://www.hpe.com/info/ecodata

社内プログラム、製品のリサイクル、エネルギー効率などの Hewlett Packard Enterprise の環境に関する情報については、次を参照してください。

http://www.hpe.com/info/environment

ドキュメントに関するご意見、ご指摘

Hewlett Packard Enterprise では、お客様により良いドキュメントを提供するように努めています。ドキュメントを改善するために役立てさせていただきますので、何らかの誤り、提案、コメントなどがございましたら、ドキュメントフィードバック担当（**docsfeedback@hpe.com**）へお寄せください。この電子メールには、ドキュメントのタイトル、部品番号、版数、およびドキュメントの表紙に記載されている刊行日をご記載ください。オンラインヘルプの内容に関するフィードバックの場合は、製品名、製品のバージョン、ヘルプの版数、およびご利用規約ページに記載されている刊行日もお知らせください。

CLOB および BLOB データにアクセスする見本プログラム

この付録では、2 つの作業プログラムを示しています。

- CLOB データにアクセスする見本プログラム
- BLOB データにアクセスする見本プログラム

CLOB データにアクセスする見本プログラム

この見本プログラムは、CLOB インターフェイスまたは、PreparedStatement インターフェイスを通して実行できる操作を示します。見本プログラムは、両インターフェイスが変数をとり、変数の値を CLOB 列を含む実テーブルに入れる例を示します。

```
// LOB 操作は、Clob インターフェイス
// または PreparedStatement インターフェイスを通して実行できます。
// このプログラムは、両方のインターフェイスが
// 変数をとり、それを cat.sch.clobbase テーブルに入れる例を示します。
//
// この例の LOB 実テーブルは、次のように作成されます。
//     >> create table clobbase
//             (col1 int not null not droppable,
//             col2 clob, primary key (col1));
//
// この例の LOB テーブルは、
// JdbcMxLobAdmin ユーティリティを通して作成されます。
//     >> create table cat.sch.clobdatatbl
//             (table_name char(128) not null not droppable,
//                data_locator largeint not null not droppable,
//             chunk_no int not null not droppable,
//             lob_data varchar(3880),
//             primary key(table_name, data_locator, chunk_no))
//             attributes extent(1024), maxextents 768 ;
//
// ***** 次の Clob インターフェイスは...
// - LOB 列の値として、EMPTY_CLOB() を持つ
// 基本の行を挿入します
// - '更新用' の LOB 列を選択し
// - データを含む byte[] をロードし
// - Outputstream.write(byte[]) を使用します
//
// ***** 次の PreparedStatement インターフェイスは...
// - 既にデータを含む Inputstream オブジェクトを必要とし
// - 制約する PreparedStatement オブジェクトを必要とし
// 実テーブルの DML を '挿入し...'
// - LOB データ用の ps.setAsciiStream()
// - DML 用の ps.executeupdate();
//
// この例を実行するには、以下を実行します。
// # java TestCLOB 1 TestCLOB.java 1000
//

import java.sql.*;
import java.io.*;

public class TestCLOB
{
    public static void main (String[] args)
                throws java.io.FileNotFoundException,
                java.io.IOException
{
```

```
int         length = 500;
int         recKey;
long         start;
long         end;
Connection         conn1 = null;

// jdbcmx.clobTableName システム プロパティを設定します。このプロパティ
// 以下を通して、コマンドラインで追加することもできます。
// "-Djdbcmx.clobTableName=..."。または、
// java.util.Properties オブジェクトが使用して、以下に引き渡すことができます
// getConnection. System.setProperty( "jdbcmx.clobTableName","cat.sch.clobdatatbl" );

if (args.length < 2) {
                System.out.println("arg[0]=;
                arg[1]=file;
                arg[2]=");
                return;
}

String k = "K";
for (int i=0; i<5000; i++) k = k + "K";
System.out.println("string length = " + k.length());

FileInputStream clobFs = new FileInputStream(args[1]);
int clobFsLen = clobFs.available();

if (args.length == 3)
                length = Integer.parseInt(args[2]);
recKey = Integer.parseInt(args[0]);

System.out.println("Key: " + recKey +"; Using " + length + " of file " + args[1]);

try {
        Class.forName("com.tandem.sqlmx.SQLMXDriver");
        start = System.currentTimeMillis();
        conn1 = DriverManager.getConnection("jdbc:sqlmx:");

        System.out.println("Cleaning up test tables...");
        Statement stmt0 = conn1.createStatement();
        stmt0.execute("delete from clobdatatbl");
        stmt0.execute("delete from clobbase");

        conn1.setAutoCommit(false);

}
catch (Exception e1) {
        e1.printStackTrace();
}

// PreparedStatement インターフェイスの例 - この手法は、
// LOB データが既に NonStop
// システムディスク上にある場合に適しています。
try {
        System.out.println("PreparedStatement interface LOB insert...");
        String stmtSource1 = "insert into clobbase values (?,?)";
        PreparedStatement stmt1 = conn1.prepareStatement(stmtSource1);
stmt1.setInt(1,recKey);
        stmt1.setAsciiStream(2,clobFs,length);
        stmt1.executeUpdate();
        conn1.commit();
}
catch (SQLException e) {
        e.printStackTrace();
        SQLException next = e;
    do {
            System.out.println("Messge : " + e.getMessage());
            System.out.println("Error Code : " + e.getErrorCode());
            System.out.println("SQLState : " + e.getSQLState());
        } while ((next = next.getNextException()) != null);
    }

// Clob インターフェイスの例 - この手法は、
// LOB データが既にアプリ内にあり、
```

```
// msgbuf に転送されている場合に適しています。
try {
// 2 番目の実テーブルの列に空の LOB 列を挿入します。
    System.out.println("CLOB interface EMPTY LOB insert...");
    String stmtSource2 = "insert into clobbase values (?,EMPTY_CLOB())";
    PreparedStatement stmt2 = conn1.prepareStatement(stmtSource2);
stmt2.setInt(1,recKey+1);
stmt2.executeUpdate();

Clob clob = null;

    System.out.println("Obtaining CLOB data to update (EMPTY in this case)...");
    PreparedStatement stmt3 = conn1.prepareStatement("select col2 from clobbase where col1 = ? for
update");
    stmt3.setInt(1,recKey+1);
    ResultSet rs = stmt3.executeQuery();
    if (rs.next()) clob = rs.getClob(1); // has to be there
// 実テーブルへの挿入に失敗した場合

    System.out.println("Writing data to previously empty CLOB...");
    OutputStream os = clob.setAsciiStream(1);
    byte[] bData = k.getBytes();
    os.write(bData);
    os.close();
conn1.commit();
}
catch (SQLException e) {
e.printStackTrace();
SQLException next = e;
    do {
                System.out.println("Messge : " + e.getMessage());
                System.out.println("Vendor Code : " + e.getErrorCode());
                System.out.println("SQLState : " + e.getSQLState());
            } while ((next = next.getNextException()) != null);
        }
} // メイン
} // クラス
```

BLOB データにアクセスする見本プログラム

この見本プログラムは、Blob インターフェイスと PreparedStatement インターフェイスの両方を使用して、バイト変数を取得し、その変数を値を BLOB 列を含む実テーブルに入れる例を示します。

```
// LOB 操作は、または、
// PreparedStatement インターフェイスを通して実行できます。このプログラムでは、
// 両方のインターフェイスを使用して、byte[] 変数を取得し、
// それを cat.sch.blobtiff テーブルに挿入する例を示します。
//
// この例の LOB 実テーブルは、次のように作成されます。
// >> create table blobtiff // (col1 int not null not droppable,
// tiff blob, primary key (col1));
//
// この例の LOB テーブルは、
// JdbcMxLobAdmin ユーティリティを通して作成されます。
// >> create table cat.sch.blobdatatbl
// (table_name char(128) not null not droppable,
// data_locator largeint not null not droppable,
// chunk_no int not null not droppable,
// lob_data varchar(3880),
// primary key(table_name, data_locator, chunk_no))
// attributes extent(1024), maxextents 768 ;
// ***** 次の Blob インターフェイスは...
// - LOB 列の値として、EMPTY_BLOB() を持つ
// - '更新用' の LOB 列を選択し
// - データを含む byte[] をロードし
// - Outputstream.write(byte[]) を使用します
//
// ***** 次の prep stmt インターフェイスは...
// - 既にデータを含む Inputstream オブジェクトを必要とし
// - 制約する PreparedStatement オブジェクトを必要とし
```

```
// 実テーブルの DML を '挿入し...'
// - LOB データ用の ps.setAsciiStream()
// - DML 用の ps.executeupdate();
//
// この例を実行するには、以下を実行します。
// # java TestBLOB 1 TestBLOB.class 1000
//

import java.sql.*;
import java.io.*;

public class TestBLOB
{
    public static void main (String[] args)
throws java.io.FileNotFoundException, java.io.IOException
    {
        int numBytes;
        int recKey;
        long start;
        long end;
        Connection conn1 = null;

        // Set jdbcmx.blobTableName System Property. This property
// 以下を通して、コマンドラインで追加することもできます。
// "-Djdbcmx.blobTableName=...", または
// java.util.Properties オブジェクトが使用して、以下に引き渡すことができます
        // getConnection.
System.setProperty ("jdbcmx.blobTableName","cat.sch.blobdatatbl");

        if (args.length < 2) {
                System.out.println("arg[0]=; arg[1]=file; arg[2]=");
return;
        }

// blob のバイト配列
        byte[] whatever = new byte[5000];
        for (int i=0; i<5000; i++) whatever[i] = 71; // "G"

        String k = "K";
        for (int i=0; i<5000; i++) k = k + "K";
        System.out.println("string length = " + k.length());

        java.io.ByteArrayInputStream iXstream = new java.io.ByteArrayInputStream(whatever);

        numBytes = iXstream.available();
        if (args.length == 3) numBytes = Integer.parseInt(args[2]);
        recKey = Integer.parseInt(args[0]);

        System.out.println("Key: " + recKey +"; Using " + numBytes + " of file " + args[1]);

        try {
                Class.forName("com.tandem.sqlmx.SQLMXDriver");
                start = System.currentTimeMillis();
                conn1 = DriverManager.getConnection("jdbc:sqlmx:");

                System.out.println("Cleaning up test tables...");
                Statement stmt0 = conn1.createStatement();
                stmt0.execute("delete from blobdatatbl");
                stmt0.execute("delete from blobtiff");

                conn1.setAutoCommit(false);
        }
        catch (Exception e1) {
        e1.printStackTrace();
        }

// PreparedStatement インターフェイスの例 - この手法は、
// LOB データが既に NonStop
// システムディスク上にある場合に適しています。
try {
        System.out.println("PreparedStatement interface LOB insert...");
        String stmtSource1 = "insert into blobtiff values (?,?)";
        PreparedStatement stmt1 = conn1.prepareStatement(stmtSource1);
```

```
        stmt1.setInt(1,recKey);
        stmt1.setBinaryStream(2,iXstream,numBytes);
        stmt1.executeUpdate();
        conn1.commit();
    }
    catch (SQLException e) {
        e.printStackTrace();
        SQLException next = e;
        do {
                System.out.println("Messge : " + e.getMessage());
                System.out.println("Error Code : " + e.getErrorCode());
                System.out.println("SQLState : " + e.getSQLState());
        } while ((next = next.getNextException()) != null);
    }

// Blob インターフェイスの例 - この手法は、
// LOB データが既にアプリ内にあり、
// msgbuf に転送されている場合に適しています。
try {
// 2 番目の実テーブルの行に空の LOB 列を挿入します。
        System.out.println("BLOB interface LOB insert...");
        String stmtSource2 = "insert into blobtiff values (?,EMPTY_BLOB())";
        PreparedStatement stmt2 = conn1.prepareStatement(stmtSource2);
        stmt2.setInt(1,recKey+1);
        stmt2.executeUpdate();

Blob tiff = null;

        System.out.println("Obtaining BLOB data to update (EMPTY in this case)...");
        PreparedStatement stmt3 = conn1.prepareStatement("select tiff from blobtiff where col1 = ? for
update");
        stmt3.setInt(1,recKey+1);
        ResultSet rs = stmt3.executeQuery();
        if (rs.next()) tiff = rs.getBlob(1); // has to be there else the base table insert failed

        System.out.println("Writing data to previously empty BLOB...");
        OutputStream os = tiff.setBinaryStream(1);
        byte[] bData = k.getBytes();
        os.write(bData);
os.close();
conn1.commit();
    }
catch (SQLException e) {
e.printStackTrace();
SQLException next = e;
            do {
                System.out.println("Messge : " + e.getMessage());
                System.out.println("Vendor Code : " + e.getErrorCode());
                System.out.println("SQLState : " + e.getSQLState());
            } while ((next = next.getNextException()) != null);
        }
} // メイン
} // クラス
```

用語集

A

abstract class (抽象クラス)

Java では、サブクラスを派生できる親としてのみ指定され、それ自体はインスタンス生成には適していないクラス。抽象クラスは多くの場合、不完全な機能セットを「抽象化」するために使用されます。抽象後、欠けた部分のさまざまなバリエーションを追加する兄弟サブクラスのグループで共有することができます。

American Standard Code for Information Interchange (ASCII)

現代のコンピューターの主流の文字セットエンコーディング。ASCII は、1 文字ごとに 7 ビットを使用します。これには、アクセント文字や英語で使用されないその他の文字形式 (ドイツ語の ß やノルウェー語の æ など) は一切含まれません。Unicode と比べてください。

American National Standards Institute (ANSI)

コンピューターや通信などの多くの領域における米国の規格の承認を担当する米国政府機関。ANSI は、ISO のメンバーです。ANSI では、ANSI および ISO (国際) 規格を販売しています。

ANSI

「American National Standards Institute (ANSI)」を参照してください。

API

「アプリケーションプログラムインターフェイス (API)」を参照してください。

application program (アプリケーション プログラム)

1. 特定の目的のためにユーザーに向けて、またはユーザーによって作成されたソフトウェアプログラム。
2. 制御関数ではなく、データ処理関数を実行するコンピューター プログラム。

application program interface (アプリケーションプログラムインターフェイス (API))

アプリケーションプログラムが、他のソフトウェアコンポーネントと通信するためにコールする関数またはプロシージャーのセット。

ASCII

「American Standard Code for Information Interchange (ASCII)」を参照してください。

AWT

「抽象ウィンドウツールキット (AWT)」を参照してください。

autocommit mode (自動コミット モード)

JDBC ドライバーは、自動的にプログラマ コール `commit()` なくトランザクションをコミットします。

B

base table (実テーブル)

物理的な存在を持つテーブル: つまり、ファイルに格納されたテーブルです。

BLOB

Binary Large OBject (バイナリラージオブジェクト) の短縮形。データベース管理システムの単一エンティティとして格納されるバイナリデータの集合。これらのエンティティは、主に、イメージ、ビデオ、

音のようなマルチメディア オブジェクトを格納する使用されます。また、これらは、プログラムまたはコードのフラグメントを格納するために使用できます。Java `Blob` オブジェクト (Java type, `java.sql.Blob`) は、SQL `BLOB` データタイプに対応します。

branded

Sun Microsystems, Inc. が準拠と認定した Java 仮想マシン。

browser (ブラウザー)

ハイパーテキストを読み込むためのプログラム。ブラウザーを使用して、ノードの内容を表示し、ノードからノードに移動できます。Internet Explorer、Netscape Navigator、NCSA Mosaic、Lynx、および W3 が、WWW 用ブラウザーの代表例です。リモートサーバーへのクライアントとして機能します。

bytecode (バイトコード)

Java コンパイラである javac が生成するコード。

C

catalog (カタログ)

SQL/MP および SQL/MX におけるテーブル、ビュー、列、インデックス、ファイル、パーティションなど SQL オブジェクトの記述を含む一連のテーブル。

class path (クラスパス)

`/usr/tandem/java/bin` クラス ライブラリ (classes.zip など) 用ディレクトリ検索に配置される Java 仮想マシンおよび他の Java プログラムのディレクトリ。明示的または `CLASSPATH` 環境変数を使用してクラス パスを設定できます。

CLOB

Character Large OBject (キャラクターラージオブジェクト) の短縮形。データベース管理システムの単一エンティティとして格納されるテキスト データ。A Java `Clob` オブジェクト (Java type, `java.sql.Clob`) は、SQL `CLOB` データタイプに対応します。

client (クライアント)

サーバーからサービスを要求するソフトウェアプロセス、ハードウェアデバイス、またはそれら 2 つの組み合わせ。多くの場合、クライアントは、プログラム可能なワークステーション上に存在するプロセスであり、ユーザーインターフェイスを提供するプログラムの一部です。ワークステーション クライアントは、プログラム ロジックの他の部分を実行する場合もあります。リクエスターとも呼ばれます。

command (コマンド)

オペレーターまたはプログラムによって要求された操作。サブシステムによるアクションの要求やサブシステムからの情報の要求。コマンドは通常、プログラムからサブシステムにプロセス間メッセージとして伝達されます。

concurrency (同時処理数)

2 つ以上のトランザクションが同時にデータベース内の同じレコードに作用している状況。トランザクションを処理するために、プログラムはデータベースからの入力が整合していることを前提とし、データベースに同時に複数の変更が行われていることを考慮しません。TMF は、同時実行制御を通じて同時トランザクションを管理します。

concurrency control (同時実行制御)

複数のプロセスによる同時アクセスからのデータベース レコードの保護。TMF は、影響を受けるレコードを動的にロックおよびロック解除して、それらのレコードにアクセスするトランザクションを一度に 1 つだけに制限することにより、この制御を適用します。

connection pooling (接続プーリング)

JDBC 接続をプールするためのフレームワークです。

Core Packages (コア パッケージ)

互換性のあるあらゆるインプリメンテーションでサポートしている必要のある、Java プラットフォーム エディションの必須 API セット。

D

Data Control Language (データ制御言語 (DCL))

SQL 言語内のデータ制御文のセット。

Data Manipulation Language (データ操作言語 (DML))

SQL 言語内のデータ操作文のセット。これらの文には、INSERT、DELETE、UPDATE が含まれ、リモート複製データベース機能 (RDF) がレプリケートするデータベースの変更を引き起こします。

DCL

「データ制御言語 (DCL)」を参照してください。

DML

「データ操作言語 (DML)」を参照してください。

ドライバー

NonStop SQL/MX など特定のデータベース管理システムへの接続を実装する JDBC のクラス。NonStop Server for Java 6.0 には、以下のドライバー インプリメンテーションが含まれます。SQL/MP 用 JDBC ドライバー (JDBC/MP) および SQL/MX 用 JDBC ドライバー (JDBC/MX)。

`DriverManager`

ドライバーを管理する JDBC クラス。

E

exception (例外)

プログラムを正常に続行できなくするプログラム実行中のイベント。一般にエラー。Java メソッドでは、throw キーワードを使用して例外を生成し、try、catch、finally ブロックを使用して例外を処理します。

Expand

地理的に分散した NonStop システムのネットワークにフォールトトレランスの概念を拡張した HPE NonStop オペレーティングシステムネットワーク。ネットワークが適切に設計されていれば、単一回線障害またはコンポーネント障害が起きた場合でも、通信パスを常に使用できます。

expandability (拡張性)

「スケーラビリティ」を参照してください。

F

フォールトトレランス

データや機能を失わずに、単一の障害（システムコンポーネントの障害）の間やその後で処理を続行できるコンピューターシステムの能力。

G

`get()` メソッド

データ アイテムを読み取るために使用されるメソッドです。例えば、`SQLMPConnection.getAutoCommit()` メソッドは、JDBC ドライバーの接続のトランザクションモードを SQL/MP または SQL/MX データベースに返します。Set() メソッドに対して比較します。

Guardian

NonStop オペレーティングシステムと共に対話型かつプログラミング的に利用可能な環境。Guardian 環境で実行するプロセスは、Guardian システム プロシージャーコールを API として使用します。Guardian 環境の対話型ユーザーは、TACL または別の HPE 製品のコマンドインタープリターを使用します。OSS に対して比較します。

GUI

「グラフィカル ユーザーインターフェイス (GUI)」を参照してください。

H

Hotspot virtual machine (Hotspot 仮想マシン)

「Java Hotspot virtual machine (Java Hotspot 仮想マシン)」を参照してください。

HPE JDBC Driver for SQL/MP (SQL/MP 用 HPE JDBC ドライバー(JDBC/MP))

SQL/MP へのアクセスを可能にし、JDBC API に準拠する製品。

HPE JDBC Driver for SQL/MX (SQL/MX 用 HPE JDBC ドライバー (JDBC/MX))

SQL/MX へのアクセスを可能にし、JDBC API に準拠する製品。

HPE NonStop ODBC Server (HPE NonStop ODBC サーバー)

NonStop システム用の ODBC HPE インプリメンテーション。

HPE NonStop Server for Java Transaction API

Java Transaction API (JTA) のインプリメンテーション。NonStop Server for Java Transaction API には、JTS を使用するバージョンも、TMF を使用するバージョンもあります。

HPE NonStop SQL/MP (SQL/MP)

HPE NonStop Structured Query Language/MP、NonStop サーバー用の HPE リレーショナルデータベース管理システム。

HPE NonStop SQL/MX（SQL/MX）

HPE NonStop Structured Query Language/MX、NonStop サーバーでのビジネスクリティカルなアプリケーション用の HPE 次世代リレーショナルデータベース管理システム。

HPE NonStop Transaction Management Facility (HPE NonStop トランザクション管理ファシリティ (TMF))

トランザクションの保護、データベースの整合性、およびデータベースのリカバリを提供する HPE 製品。NonStop SQL データベースに対して JDBC ドライバーを通じて発行された SQL 文は、TMF サブシステムのプロシージャーをコールします。

HPE NonStop TS MP（TS/MP）

オンライントランザクション処理 (OLTP) 環境で NonStop SQL/MP データベースや Enscribe データベースにアクセスする Pathway サーバーの作成をサポートする HPE 製品。

HPE Tandem Advanced Command Language (TACL)

オペレーティングシステムのコマンドインタープリター。プログラミング言語としても機能し、ユーザーがエイリアス、マクロ、ファンクションキーを定義できるようにします。

HTML

「Hypertext Markup Language (HTML)」を参照してください。

HTTP

「Hypertext Transfer Protocol (HTTP)」を参照してください。

hyperlink (ハイパーリンク)

あるハイパーテキストドキュメント内のある場所から、別のドキュメント内の場所や同じドキュメント内の別の場所への参照 (リンク)。ブラウザーは通常、別の色、フォント、またはスタイルでハイパーリンクを表示します。ユーザーがリンクをアクティブ化すると (通常はマウスでクリック)、リンクの対象がブラウザーに表示されます。

hypertext (ハイパーテキスト)

対話型ブラウザーを用いて、閲覧者があるマニュアルから別のマニュアルに簡単に移動できるようにする相互参照またはリンクを含んだ一連のドキュメント (ノード)。

Hypertext Mark-up Language (HTML)

World Wide Web で 使用されるハイパーテキスト ドキュメント形式。

Hypertext Transfer Protocol (HTTP)

HTML ドキュメントの交換に World Wide Web で使用されるクライアントサーバーの TCP/IP プロトコル。

I

IEC

「International Electrotechnical Commission (国際電気標準会議 (IEC))」を参照してください。

IEEE

Institute for Electrical and Electronic Engineers (米国電気電子技術者協会(IEEE))。

inlining (インライン化)

メソッド コールをコールされるメソッドのコードに置き換えて、コールを解消すること。

interactive (対話型)

ユーザーとコンピューターシステムとの間の質問と回答の交換。

interface (インターフェイス)

一般に、人、プログラム、またはデバイス同士の通信または相互接続のポイント、またはそのやり取りのための一連のルール。「API」も参照してください。

International Electrotechnical Commission (国際電気標準会議(IEC))。

ISO と同じレベルの標準化団体。

International Organization for Standardization (国際標準化機構(ISO))。

1946 年に創設された自発的な非条約的組織。コンピューターや通信など多くの領域での国際的な標準の作成を担います。そのメンバーは、ANSI を含む 89 カ国の国家規格組織です。

Internet (インターネット)

TCP/IP ネットワーク通信プロトコルを使用する、相互接続された何千ものネットワークによるネットワーク。電子メール、ファイル転送、ニュース、リモートログイン、および数千のデータベースへのアクセスを提供します。インターネットには、次の 3 種類のネットワークが含まれています。

- NSFNET や MILNET などの高速バックボーン ネットワーク
- 企業ネットワークや大学ネットワークなどの中間レベルのネットワーク
- 個々の LAN などのスタブ ネットワーク

Internet Protocol version 6 (インターネットプロトコルバージョン 6(IPv6))。

IP は、パケットの形式やアドレッシング方式を指定します。IP の現在のバージョンは IPv4 です。IPv6 は、接続するホスト数と、転送されるデータトラフィックの合計量の両方の点から、インターネットが着実に成長できるように設計された IP の新しいバージョンです。(IP はアイピーと発音されます)

interoperability (相互運用性)

1. 複数のベンダーのシステムや、同じベンダーからのオペレーティングシステムの複数バージョンなどの異種の環境間で通信、プログラムの実行、またはデータ転送を行う機能。HPE のドキュメントでは、このコンテキストで接続性という用語を使用することが多いですが、他のベンダーでは、ハードウェアの互換性を意味するために**接続性**を使用します。
2. NonStop システム ノード内で、ある環境から別の環境への機能またはファシリティを使用する能力。例えば、OSS 環境での `gtacl` コマンドは、対話型ユーザーが Guardian 環境で Guardian ツールを起動して使用できるようにします。

interpreter (インタープリター)

ネイティブ マシンコードのバイトコードを解釈する Java 仮想マシンのコンポーネント。

Invocation API (呼び出し API)

Java 仮想マシンを起動し、そのマシン上でメソッドを呼び出す C 言語 API。呼び出し API は JNI のサブセットです。

IPv6

「Internet Protocol version 6 (インターネットプロトコルバージョン 6 (IPv6))」を参照してください。

ISO

「International Organization for Standardization (国際標準化機構(ISO))」を参照してください。

iTP Secure WebServer

サーブレットを使用して、NonStop Server for Java が組み込まれる HPE Web サーバー。

J

`jar`

Java アーカイブツール。複数のファイルを単一の Java アーカイブ (JAR)ファイルにまとめます。また、Java アーカイブツールを実行するコマンドも指します。

JAR file (JAR ファイル)

Java アーカイブファイル。Java アーカイブツールによって生成されます。

Java

Java アプリケーションランチャー。Java ランタイム環境を始動し、指定されたクラスをロードし、そのクラスの `main` メソッドを呼び出すことによりアプリケーションを起動します。

Java 2 Platform, Enterprise Edition (J2EE プラットフォーム)

エンタープライズアプリケーションを開発して展開するための環境。J2EE プラットフォームは、複数階層の Web ベースアプリケーションを開発する機能を提供する一連のサービス、アプリケーションプログラミングインターフェイス (API)、およびプロトコルから構成されます。

JavaBeans

Java がサポートする、プラットフォームに依存しないコンポーネント アーキテクチャー。複数のハードウェアおよびオペレーティングシステムの環境用のソフトウェアの開発またはアセンブルを目的としています。詳細情報については、Sun Microsystems JavaBeans ドキュメント (http://java.sun.com/javase/6/docs/technotes/guides/beans/index.html) を参照してください。

JavaBeans Development Kit (JavaBeans 開発キット (BDK))

NonStop Server for Java 6.0 付属の JavaBeans を作成するための一連のツール。

Java Conformance Kit (JCK)

Sun Microsystems の仕様に準拠するためにベンダーの JDK が合格する必要のある適合性テストをまとめたもの。

Java Database Connectivity (JDBC)

Java プラットフォームと NonStop SQL/MP や NonStop SQL/MX などのリレーショナルデータベース間の、データベースに依存しない接続用の業界標準。JDBC は、SQL ベースのデータベースアクセス用のコールレベルの API を提供します。

Java HotSpot virtual machine (Java HotSpot 仮想マシン)

サーバー環境で実行しているアプリケーションのプログラム実行速度を最大にするように設計された Java 仮想マシン実装。この仮想マシンは、実行中のアプリケーションのパフォーマンスを動的に最適化する適応型コンパイラを備えています。

Java IDL

「Java Interface Development Language (Java インターフェイス開発言語 (Java IDL))」を参照してください。

Java Interface Development Language (Java インターフェイス開発言語 (Java IDL))

CORBA と Java 相互運用性をサポートしているライブラリ。詳細情報については、Sun Microsystems Java IDL ドキュメント (http://java.sun.com/products/jdk/idl/index.html) を参照してください。

Java Naming and Directory Interface（JNDI）

Java プラットフォームに対する標準の拡張。Java テクノロジー対応のアプリケーションプログラムに、複数のネーミングサービスおよびディレクトリサービスに対する統一インターフェイスを提供します。

Java Native Interface (Java ネイティブ インターフェイス (JNI))

Java クラスによって呼び出される C 関数が使用する C 言語インターフェイス。C プログラムから Java 仮想マシンを起動する呼び出し API が含まれています。

Java Platform Standard Edition (Java SE 6)

コア Java テクノロジープラットフォーム。デスクトップおよびサーバーでのアプリケーション開発と、内蔵環境での展開に完全な環境を提供します。詳細情報については、Sun Microsystems JDK 6.0 のマニュアルを参照してください。

Java runtime (Java ランタイム)

「Java SE Runtime Environment (Java SE ランタイム環境)」を参照してください。

Java SE Development Kit (JDK)

Java SE プラットフォームに付属の開発キット。Java SE Runtime Environment (JRE) と比べてください。「Java Platform Standard Edition 6.0 (Java SE)」も参照してください。

Java SE Runtime Environment (JRE)

Java 仮想マシンおよびコアパッケージ。これは Java コマンドが呼び出す標準の Java 環境です。Java SE Development Kit (JDK) と比べてください。「Java Platform Standard Edition 6.0 (Java SE)」も参照してください。

Java Transaction API (Java トランザクション API (JTA))

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わる要素 (リソースマネージャー、アプリケーションサーバー、トランザクションアプリケーション) との間の標準 Java インターフェイスを指定する Sun Microsystems 製品。詳細情報については、Sun Microsystems JTA ドキュメント (http://java.sun.com/products/jta/index.html) を参照してください。

Java Transaction Service (Java トランザクションサービス (JTS))

トランザクション API。OMG の OTS でモデル化されています。NonStop Server for Java 6.0 には、jts.Current インターフェイスのインプリメンテーションが含まれています。

Java virtual machine (Java 仮想マシン (JVM))

Java バイトコードをロード、リンク、確認、および解釈するプロセス。NonStop Server for Java 6.0 では、Java HotSpot サーバー仮想マシンを実装しています。

Java Virtual Machine Tool Interface (Java 仮想マシンツールインターフェイス (JVM TI))

開発および監視ツールで使用されるプログラミング インターフェイス。状態を検査し、JVM で実行しているアプリケーションの実行を制御するために使用されます。それによって VM が提供するデバッグ サービスを定義します。

`javac`

Java コンパイラ。Java ソースコードをバイトコードでコンパイルします。また、Java コンパイラを実行するコマンドも指します。

`javachk`

Java Checker。Java 仮想マシンでの問題が TCP/IP 構成の誤りによるものかどうかを判断します。また、Java Checker を実行するコマンドも指します。

`javadoc`

Java API ドキュメント ジェネレーター。Java ソースコードから、HTML 形式で API ドキュメントを生成します。また、Java API ドキュメント ジェネレーターを実行するコマンドも指します。

`javah`

C ヘッダーとスタブファイルジェネレーター。Java クラスから C ヘッダーファイルと C ソースファイルを生成し、Java および C コードがやり取りするための接続を提供します。また、C ヘッダーおよびスタブ ファイル ジェネレーターを実行するコマンドも指します。

`javap`

Java クラスファイル逆アセンブラー。コンパイルした Java ファイルを逆アセンブルし、Java バイトコードの表現を出力します。また、Java クラスファイル逆アセンブラーを実行するコマンドも指します。

JCK

「Java Conformance Kit (JCK)」を参照してください。

`jdb`

Java デバッガー。Java プログラムのエラーの検出と修正に役立ちます。また、Java デバッガーを実行するコマンドも指します。`jdb` は、Java デバッガー API を使用します。

JDBC

「Java Database Connectivity (JDBC)」を参照してください。

JDBC/MP

「HPE JDBC Driver for SQL/MP (JDBC/MP)」を参照してください。

JDBC/MX

「HPE JDBC Driver for SQL/MX (JDBC/MX)」を参照してください。

JDK

「Java SE Development Kit (Java SE 開発キット (JDK))」を参照してください。

JNDI

「Java Naming and Directory Interface (JNDI)」を参照してください。

JNI

「Java Native Interface (Java ネイティブ インターフェイス (JNI))」を参照してください。

`jre`

Java ランタイム環境。Java バイトコードを実行します。「Java SE Runtime Environment (JRE)」も参照してください。

JTA

「Java Transaction API (Java トランザクション API (JTA))」を参照してください。

JTS

「Java Transaction Service (Java トランザクションサービス (JTS))」を参照してください。

`jts.Current`

トランザクション境界を定義するための JTS インタフェース。NonStop Server for Java 6.0 には、`jts.Current` のインプリメンテーションが含まれています。

JVM

「Java virtual machine (Java 仮想マシン (JVM))」を参照してください。

JVM TI

「Java Virtual Machine Tool Interface (Java 仮想マシンツールインターフェイス (JVM TI))」を参照してください。

K～M

key (キー)

1. データベース内のレコードを識別するために使用する値。レコードに固定関数を適用することによって導出されます。キーは多くの場合、単なるフィールドの 1 つです (レコードが行になっているテーブルであるとデータベースが見なされる場合は、列)。また、1 つまたは複数のフィールドに関数を適用することによりキーを取得することができます。
2. 暗号化されたメッセージを復号して元のプレーンテキストを再現するために使用するアルゴリズムで指定しなければならない値です。一部の暗号化方式では同じ (秘密) キーを使用してメッセージを暗号化および解読しますが、公開キー暗号化ではプライベート (秘密) キーと、すべての当事者によって知られている公開キーを使用します。

LAN

「ローカルエリアネットワーク (LAN)」を参照してください。

local area network (ローカルエリアネットワーク (LAN))

地理的に限定されているデータ通信ネットワーク (通常は半径 1 キロメートル)。端末、マイクロプロセッサー、コンピューターを隣接する建物内で容易に相互接続できます。イーサーネットは LAN の一例です。

LOB

ラージ オブジェクトの略称。`CLOB` または `BLOB` データのいずれかを表します。

macro (マクロ)

仮引き数を含むことができるコマンドのシーケンス。マクロを実行するとき、仮引き数に実パラメーターが代入されます。

MXCI

SQL/MX 会話型インターフェイス

N

native (ネイティブ)

Java プログラミングにおいて、特定のプラットフォームで Java 以外の言語 (C または C++ など) で書かれたもの。

native method (ネイティブ メソッド)

Java クラスによってコールされる、Java 以外のルーチン (C または C++ などの言語で書かれている)。

`native2ascii`

ネイティブ - ASCII コンバーター。Unicode でエンコードされた文字を含むファイルの中のネイティブ エンコードされた文字を含むファイルを変換します。ネイティブ - ASCII コンバーターを実行するコマンドでもあります。

node (ノード)

1. コンピューター ネットワークに接続されているアドレス指定可能なデバイス。
2. ハイパーテキスト ドキュメント。

NonStop operating system (NonStop オペレーティングシステム)

NonStop システムのオペレーティングシステム。

NonStop Server for Java, based on Java Standard Edition 5.0 (Java 用 HPE NonStop サーバー (Java Standard Edition 5.0 に基づく))

NonStop Server for Java 2 プラットフォーム標準エディション (J2SE) 5.0 に準拠する Java 仮想マシンを備えた NonStop Server for Java 製品の正式名。また、「NonStop Server for Java 5」を参照してください。

NonStop Server for Java 5

Java 2 プラットフォーム標準エディション 5.0 製品に基づいた Java 用 NonStop サーバーの非公式名。この製品は、エンタープライズサーバーのコンパクトで並列型、ダイナミックで移植可能なプログラムをサポートする Java 環境です。

NonStop system (NonStop システム)

NonStop オペレーティングシステムをサポートする HPE コンピューター (ハードウェアおよびソフトウェア)。

NonStop Technical Library (NonStop テクニカルライブラリ)

NonStop コ ンピューティング技術情報へのブラウザーベースのインターフェイス。NonStop テクニカルライブラリは、HPE Total Information Manager (TIM) に代わるものです。

NSK

「NonStop operating system (NonStop オペレーティングシステム)」を参照してください。

NSKCOM

スワップ領域のプログラム管理ツール。

O

Object Management Group (オブジェクトマネージメントグループ (OMG))

CORBA を定義した標準団体。

Object Serialization (オブジェクト直列化)

以下をサポートすることによりオブジェクトのサポートを備えた、コア Java 入力/出力クラスを拡張する Sun Microsystems の手続き。

- オブジェクトと、そのオブジェクトから到達可能なオブジェクトの、バイトストリームでのエンコーディング。
- ストリームからのオブジェクト グラフの補足的な再構築。

オブジェクト直列化は、軽量の永続化と、ソケットまたは RMI による通信に使用されます。デフォルトのオブジェクト エンコーディングが非公開の一時的データを保護し、クラスの展開をサポートします。クラスが独自の外部エンコーディングを実装でき、外部形式について全責任を負います。

Object Transaction Service (オブジェクトトランザクションサービス (OTS))

OMG により採用されたトランザクション サービス標準。JTS のモデルとして使用されます。

ODBC

「Open Database Connectivity (ODBC)」を参照してください。

OLTP

「online transaction processing (オンライントランザクション処理(OLTP))」を参照してください。

OMG

「Object Management Group (オブジェクトマネージメントグループ (OMG))」を参照してください。

online transaction processing (オンライントランザクション処理(OLTP))

入力されたトランザクションがすぐにデータベースに適用されるトランザクションを処理する方法。データベース内の情報は常に会社の状態によって最新であり、オンライン画面と印刷されたレポートを通じてすべてのユーザーがすぐに入手できます。トランザクションはリクエスターが待機している間に処理されます。これに対して、キューに入ったトランザクションやバッチトランザクションは、後で処理されます。オンライン トランザクション処理は、注文処理、インベントリ管理、アカウンティング機能、銀行業務など、さまざまな種類のビジネス タスクで使用できます。

Open Database Connectivity (ODBC)

データベースにアクセスするための標準の Microsoft 製品。

Open System Services (オープンシステムサービス(OSS))

NonStop オペレーティングシステムと共に対話型かつプログラミング的に利用可能な環境。OSS 環境で実行されるプロセスは OSS API を使用します。OSS 環境の双方向型ユーザーは、そのコマンドインタープリターとして OSS シェルを使用します。Guardian と比較します。

OSS

「Open System Services (オープンシステムサービス(OSS))」を参照してください。

OTS

「Object Transaction Service (オブジェクトトランザクションサービス (OTS))」を参照してください。

P

package (パッケージ)

関連するクラスのコレクション (例：JDBC)。

Pathsend API

Pathway システムへのアプリケーションプログラムインターフェイス。Pathsend プロセスがサーバープロセスと通信可能になります。

Pathsend process (Pathsend プロセス)

Pathsend インターフェイスを使用してサーバープロセスと通信するクライアント (リクエスター) プロセス。Pathsend プロセスは、標準的リクエスターとネストサーバーのどちらでも構いません。標準的リクエスターはアプリケーション要求を開始します。ネストサーバーは、サーバークラスとして構成されていますが、リクエスターとして、他のサーバーに要求します。Pathsend リクエスターとも呼ばれます。

Pathway

クライアント/サーバーモデルを使用する OLTP プログラムを開発および監視するためのソフトウェアツールのグループです。サーバーは、要求された処理を実行するサーバー クラスにグループ化されます。NonStop システムでは、このツールのグループは、TS/MP および Pathway/TS という 2 つの別々の製品にパッケージ化されています。

Pathway CGI

Pathway サーバー クラスへの CGI 式アクセスを提供する、iTP Secure WebServer の拡張機能です。NonStop Server for Java で拡張されているため、Java サーブレットを特殊な Pathway CGI サーバー、`ServletServerClass` から呼び出すことができます。

Pathway/TS

NonStop サーバーの Guardian 環境で OLTP プログラムをサポートするためスクリーン プログラムを開発およびインタープリトするためのツールを提供する HPE 製品。Pathway/TS スクリーン プログラムは、端末およびインテリジェント デバイスと通信します。Pathway/TS では TS/MP 製品のサービスが必要です。

persistence (永続性)

1. 作成したオブジェクトと変数が存在し続け、プログラム実行が切り替わってもそれぞれの値を維持し続ける、プログラミング言語のプロパティです。
2. プロセスとして動作しているプログラムのように、存在し続けられる機能です。

portability (ポータビリティ)

プログラムをあるプラットフォームから別のプラットフォームへ、書き換えることなく移植できること。オープンシステムの特性です。ポータビリティは、C などの標準プログラミング言語を使用していることを意味します。

Portable Operating System Interface X (POSIX)

ANSI と IEEE によって定義されている、相関したインターフェイス標準のファミリです。各 POSIX インターフェイスは、番号が付いた ANSI/IEEE 標準またはドラフト標準で個別に定義されています。この標準は、ユーザーインターフェイスの移植性、相互運用性、一様性の問題を扱います。

POSIX

「Portable Operating System Interface X (POSIX)」を参照してください。

private key (プライベートキー)

一部の関係者にしか知られていない暗号キー。

protocol (プロトコル)

データを、特にネットワーク全体で送信するための形式上規則のセット。低レベルのプロトコルでは、電気的および物理的な基準、ビット順序、バイト順序、転送、エラー検出、ビットストリームのエラー訂正を定義します。高レベルのプロトコルでは、データの形式設定 (メッセージの構文、ターミナル-コンピューター ダイアログ、キャラクターセット、メッセージのシーケンスなど) を定義します。

Pthread

POSIX スレッドです。

public key (公開キー)

すべての関係者に知られている暗号キーです。

Pure Java

コアパッケージだけに依存する Java。どこでも動作できることを意味します。

R

_RLD_LIB_PATH

Java VM および他の Java プログラム検索 TNS/E jdbcMx PIC ファイルの場所です。_RLD_LIB_PATH を明示的に、または、`_RLD_LIB_PATH` 環境変数と共に設定します。

RDF

「リモート複製データベース機能 (RDF)」を参照してください。

リモート複製データベース機能 (RDF)

以下を行うヒューレット・パッカード エンタープライズ ソフトウェア製品:

- OLTP 本稼働データベースの障害回復を支援する
- プライマリ システム上の TMF サブシステムにより監査されるデータベース アップデートを監視し、これらのアップデートをリモートシステムのデータベースコピーに適用します。

リモートメソッド呼び出し (RMI)

すべての Java 環境で同質の分散オブジェクトに使用される Java パッケージ。

requester (リクエスター)

「クライアント」を参照してください。

RMI

「リモートメソッド呼び出し (RMI)」を参照してください。

`rmic`

Java RMI スタブ コンパイラ。リモート オブジェクトのスタブおよびスケルトンを生成します。

rmicregistry

Java リモート オブジェク トレジストリ。現在のホスト上の指定されるポートでリモート オブジェク トレジストリを開始します。

S

拡張性 (scalability)

オンライントランザクション処理システムのサイズと処理能力を、プロセッサーやデバイスをシステムに追加し、システムをネットワークに追加することで、高められる機能。そしてそれを、システムを低下させることなく、簡単かつ透過的に行なうこと。拡張性とも呼ばれます。

Scalable TCP/IP (スケーラブル TCP/IP (SIP))

Java で記述されたネットワークサーバーに拡張性と持続性を提供する方法を透過的に提供する NonStop Server for Java の機能。

serialization (直列化)

「Object Serialization (オブジェクト直列化)」を参照してください。

serialized object (直列化されたオブジェクト)

オブジェクト直列化を経たオブジェクト。

serialver

シリアル バージョン コマンド。1 つ または複数のクラスの `serialVersionUID` を返します。あるいは、シリアル バージョン コマンドを実行するコマンドです。

server (サーバー)

1. スタンドアロンシステムとして、あるいは Expand ネットワーク内のノードとして使用されるシステムの実装。
2. クライアントからネットワーク経由で受信した要求に応えてサービスを提供することを目的としたコンピューターシステムのハードウェアコンポーネント。例えば、NonStop システムサーバーは、トランザクション処理、データベースアクセス、および他のサービスを提供します。
3. クライアントにサービスを提供するプロセスまたはプログラム。サーバーの目的は、クライアントから要求メッセージを受信する、データベース照会または更新、セキュリティ検証、数値計算、他のコンピューターシステムへのデータルーティングなどの操作を必要に応じて実行する、そしてクライアントに応答メッセージを返すことです。

servlet (サーブレット)

あらゆる Web ブラウザーからアクセスできる、サーバー サイドの Java プログラム。このサーブレットを管理する Pathway CGI サーバーの拡張性と永続性を継承します。

サーブレットという名前の Java クラスは、World Wide Web サーバーなどのサーバー環境で実行されます。サーブレット API は、Sun Microsystems によるドラフト標準で定義されています。`servlets` クラスは、JDK のコアパッケージにはありません。

set() メソッド

データ項目を変更するために使用されるメソッドです。例えば、`SQLMPConnection.setAutoCommit()` メソッドは、SQL/MP または SQL/MX データベース用に、JDBC ドライバーの接続のトランザクションモードを変更します。`get()` メソッドと比較してください。

shell (シェル)

コマンドをオペレーティングシステムに渡すために使用するコマンドインタープリター。外部ネットワークへのインターフェイスとなる、オペレーティングシステムの一部分です。

SIP

「Scalable TCP/IP (スケーラブル TCP/IP (SIP))」を参照してください。

skeleton (スケルトン)

RMI での、スタブの補完部分。スケルトンとスタブが、RMI サービスと、リモートオブジェクトをコールして実装するコードとの間のインターフェイスを形成します。

SQLJ

SQLJ パート 0「データベース言語 SQL—パート 10」とも呼ばれます。静的 SQL 文を直接 Java プログラムに埋め込み可能にする ANSI SQL-2002 標準のオブジェクト言語バインディング (SQL/OLB) の部分。

SQL/MP

「HPE NonStop SQL/MP」を参照してください。

SQL/MX

「HPE NonStop SQL/MX」を参照してください。

Standard Extension API

コアパッケージの外側の API。Sun Microsystems, Inc. が API 標準を定義および公開しました。標準拡張機能の一部はコアパッケージに移行しています。標準拡張機能の例は、サーブレットや JTS です。

stored procedure (ストアド プロシージャー)

NonStop SQL/MX に登録されていて、CALL 文の実行中に NonStop SQL/MX により呼び出される手続き。ストアド プロシージャーが特に重要なのは、クライアント/サーバー データベースシステムの場合です。プロシージャーをサーバー側に保管することですべてのクライアントが使用可能になるからです。プロシージャーが変更されると、すべてのクライアントが新しいバージョンを自動的に取得します。

Stored Procedure in Java (SPJ)

本体が静的 Java メソッドであるストアド プロシージャー。

stub (スタブ)

1. プログラムをランタイムライブラリとリンクするときに使用するダミー プロシージャー。スタブにはコードを含める必要がありません。その目的は、リンク時に「未定義ラベル」エラーを防ぐことだけです。
2. リモートプロシージャーコール (RPC) の中のローカル プロシージャー。クライアントはスタブを呼び出してタスクを実行しますが、RPC が関与していることを必ずしも意識しているわけではありません。スタブはパラメーターをネットワーク経由でサーバーに送信し、結果を呼び出し元に返します。

T

TACL

「HPE Tandem Advanced Command Language (TACL)」を参照してください。

TCP/IP

「Transmission Control Protocol/Internet Protocol (TCP/IP)」を参照してください。

Technical Documentation (テクニカル ドキュメント)

HPE のテクニカル ドキュメントは、ビジネスサポートセンターにあります。

thread (スレッド)

個別にディスパッチされるタスク。プロセス内の連続した制御のフローを表します。

threads

Sun Microsystems Java SE 6.0 に付属している非ネイティブスレッドパッケージ。

throw

例外を発生させるために使用する Java キーワード。

throws

メソッドが発生させることができる例外を定義するために使用する Java キーワード。

TMF

「HPE NonStop トランザクション管理ファシリティ (TMF)」を参照してください。

TNS/E

Intel® Itanium® アーキテクチャーと NonStop オペレーティングシステムをベースにしたハードウェアプラットフォームおよびそのプラットフォームに固有のソフトウェア。すべてのコードが PIC (位置独立コード) です。

TNS/R

MIPS アーキテクチャーと NonStop オペレーティングシステムをベースにしたハードウェアプラットフォームおよびそのプラットフォーム固有のソフトウェア。コードは、PIC (位置独立コード) または非 PIC です。

transaction (トランザクション)

クライアントプログラム (通常はワークステーション上で実行中) が、サーバーに要求するユーザー定義動作。

Transaction Management Facility (トランザクション管理ファシリティ (TMF))

NonStop システム用の HPE ソフトウェア製品のセット。データベースの更新が不完全にならないようにすることで、データベースの完全性を確保します。データベースに行なわれる変更を継続的に (リアルタイムで) バックアウトして、これらの変更を必要なときに取り消すことができます。データベースのオンライン「スナップショット」バックアップを取り、これらのバックアップからデータベースを復元することもできます。

Transmission Control Protocol/Internet Protocol (TCP/IP)

最も広く使われている、ベンダーに限定されないプロトコルの 1 つ。システムの種類の違いを超えた大規模なネットワークをサポートするように設計されています。

trigger (トリガー)

トリガーは、指定された実テーブルで、削除、挿入、または更新操作が発生するたびに自動的に実行されるアクションを定義します。

U ～ Z

Unicode

ASCII 拡張となるべく設計された文字コーディング体系。各文字で 16 ビットを使用することで (ASCII の 7 ビットではなく)、Unicode はすべての言語のほぼすべての文字と多数の記号 ("&"など) を国際的標準の方法で表すことができるため、拡張された文字セットとコードページが非互換であるという複雑さが解消されます。Unicode の最初から 128 番目までのコードは、標準 ASCII 文字に対応します。

Uniform Resource Locator (URL)

ネットワーク上のオブジェクト (ファイルやニュースグループ、あるいは JDBC の場合はデータベースなど) を指定するためのドラフト標準。URL は、World Wide Web 上で広範に使用されます。HTML ドキュメントはこれらを使用してハイパーリンクのターゲットを指定します。

URL

「Uniform Resource Locator (URL)」を参照してください。

virtual machine (仮想マシン (VM))

自己完結型のオペレーティング環境。別のコンピューターであるかのように動作します。「Java 仮想マシン」および「Java Hotspot 仮想マシン」も参照してください。

VM

「仮想マシン (VM)」を参照してください。

World Wide Web (WWW)

インターネットクライアント - サーバーハイパーテキスト分散情報検索システム。スイス、ジュネーブにある CERN 高エネルギー物理学研究所で生まれました。WWW では、すべてのもの (ドキュメント、メニュー、インデックス) がユーザーに対して、HTML 形式のハイパーテキストオブジェクトとして表現されます。ハイパーテキストリンクは他のドキュメントを URL で参照します。URL が参照できるローカルまたはリモートリソースは、FTP、Gopher、Telnet、ニュースによってアクセスできるほか、ハイパーテキストドキュメントを転送するために使用する HTTP プロトコルを使ってアクセスできます。クライアントプログラム (ブラウザと呼ばれる) は、ユーザーのコンピューター上で実行され、リンクをたどる、あるいはサーバーにクエリを送信するという、2 つの基本的なナビゲーション操作を提供します。

wrapper (ラッパー)

適切な実行環境をセットアップしてから、シェルの名前に対応するバイナリファイルを実行するシェルスクリプト。

WWW

「World Wide Web (WWW)」を参照してください。