



IBM Systems - iSeries

プログラミング

IBM Developer Kit for Java

バージョン 5 リリース 4







**IBM Systems - iSeries**

**プログラミング**

**IBM Developer Kit for Java**

バージョン 5 リリース 4

**ご注意！**

本書および本書で紹介する製品をご使用になる前に、563 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM Developer Kit for Java (製品番号 5722-JV1) のバージョン 5、リリース 4、モディフィケーション 0 に適用されます。また、改訂版で断りがない限り、それ以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼働するとは限りません。また CISC モデルでは稼働しません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

<http://www.ibm.com/jp/manuals/main/mail.html>

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

<http://www.ibm.com/jp/manuals/> の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原 典： IBM Systems - iSeries  
Programming  
IBM Developer Kit for Java  
Version 5 Release 4

発 行： 日本アイ・ビー・エム株式会社

担 当： ナショナル・ランゲージ・サポート

第1刷 2006.2

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体\*)は、(財)日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注\* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、  
平成角ゴシック体™W5、平成角ゴシック体™W7

© Copyright International Business Machines Corporation 1998, 2006. All rights reserved.

© Copyright IBM Japan 2006

# 目次

|  |          |
|--|----------|
| <b>IBM Developer Kit for Java</b> . . . . .                                      | <b>1</b> |
| 新機能 . . . . .  | 1        |
| 印刷可能 PDF . . . . .   | 2        |
| IBM Developer Kit for Java をインストールして構成<br>する . . . . .                           | 2        |
| IBM Developer Kit for Java をインストールする . . . . .                                   | 3        |
| Hello World Java プログラムを初めて実行する . . . . .   | 7        |
| ネットワーク・ドライブを iSeries サーバーに割り<br>当てる . . . . .                                    | 8        |
| iSeries サーバー上にディレクトリーを作成する . . . . .   | 8        |
| HelloWorld Java プログラムを作成、コンパイ<br>ル、および実行する . . . . .                             | 10       |
| Java ソース・ファイルを作成および編集する . . . . .  | 11       |
| IBM Developer Kit for Java 用に iSeries サーバーを<br>カスタマイズする . . . . .                | 12       |
| Java クラスパス . . . . .   | 12       |
| Java システム・プロパティー . . . . .   | 14       |
| 国際化対応 . . . . .  | 25       |
| リリース間の互換性 . . . . .  | 34       |
| IBM Developer Kit for Java によるデータベース・ア<br>クセス . . . . .                          | 34       |
| IBM Developer Kit for Java の JDBC ドライバー<br>を使用して iSeries データベースにアクセスする . . . . . | 35       |
| IBM Developer Kit for Java の DB2 SQLJ サポ<br>ートを使用するデータベースへのアクセス . . . . .        | 190      |
| Java SQL ルーチン . . . . .  | 202      |
| Java と他のプログラム言語 . . . . .  | 222      |
| ネイティブ・メソッドのために Java ネイティ<br>ブ・インターフェースを使用する . . . . .                            | 223      |
| Java 用の IBM i5/OS PASE ネイティブ・メソッ<br>ド . . . . .                                  | 234      |
| Java 用 Teraspace ストレージ・モデル・ネイテ<br>ィブ・メソッド . . . . .                              | 241      |
| 統合言語環境と Java との比較 . . . . .  | 242      |
| java.lang.Runtime.exec() を使用する . . . . .   | 243      |
| プロセス間通信 . . . . .  | 247      |
| Java プラットフォーム . . . . .  | 253      |
| Java アプレットおよびアプリケーション . . . . .  | 253      |
| Java 仮想マシン . . . . .   | 254      |
| Java JAR とクラス・ファイル . . . . .   | 256      |
| Java スレッド . . . . .  | 256      |
| Sun Microsystems, Inc. Java Development Kit . . . . .                            | 257      |
| 高度なトピック . . . . .  | 258      |
| Java クラス、パッケージ、およびディレクトリ<br>ー . . . . .  | 258      |
| 統合ファイル・システム内のファイル . . . . .  | 260      |
| 統合ファイル・システム内の Java ファイル権限<br>バッチ・ジョブで Java を実行する . . . . .                       | 261      |
| グラフィカル・ユーザー・インターフェースを使用<br>しないホスト上で Java アプリケーションを実行す<br>る . . . . .             | 261      |

|  |     |
|--|-----|
| Native Abstract Windowing Toolkit . . . . .                            | 262 |
| Java セキュリティー . . . . .   | 275 |
| Java セキュリティー・モデル . . . . .   | 276 |
| Java Cryptography Extension . . . . .                                  | 276 |
| Java Secure Socket Extension . . . . .                                 | 279 |
| Java Authentication and Authorization Service . . . . .                | 308 |
| IBM Java Generic Security Service (JGSS) . . . . .                     | 344 |
| IBM Developer Kit for Java を使用して Java プロ<br>グラムのパフォーマンスを調整する . . . . . | 382 |
| Java イベント・トレースのパフォーマンス測定<br>ツール . . . . .                              | 383 |
| Java のパフォーマンスの考慮事項 . . . . .   | 383 |
| Java ガーベージ・コレクション . . . . .  | 390 |
| Java ネイティブ・メソッド呼び出しのパフォー<br>マンスに関する考慮事項 . . . . .                      | 391 |
| Java メソッドのインライン化のパフォーマンス<br>に関する考慮事項 . . . . .                         | 391 |
| Java の例外処理のパフォーマンスの考慮事項 . . . . .                                      | 392 |
| Java 呼び出しトレースのパフォーマンス測定ツ<br>ール . . . . .                               | 392 |
| Java プロファイルのパフォーマンス測定ツール . . . . .                                     | 392 |
| Java パフォーマンス・データを収集する . . . . .  | 393 |
| IBM Developer Kit for Java 用のコマンドとツール . . . . .                        | 397 |
| IBM Developer Kit for Java がサポートする<br>Java ツール . . . . .               | 397 |
| Java でサポートされる CL コマンド . . . . .  | 405 |
| Java でサポートされる iSeries ナビゲーター・コ<br>マンド . . . . .                        | 406 |
| サーバーで実行する Java プログラムをデバッグす<br>る . . . . .                              | 407 |
| i5/OS コマンド行から Java プログラムをデバッ<br>グする . . . . .                          | 408 |
| IBM Developer Kit for Java のコード例 . . . . .                             | 420 |
| IBM Developer Kit for Java のトラブルシューティ<br>ング . . . . .                  | 555 |
| 制限 . . . . .   | 555 |
| Java の問題分析用のジョブ・ログを検索する . . . . .                                      | 556 |
| Java の問題分析用のデータを収集する . . . . .   | 557 |
| プログラム一時修正を適用する . . . . .   | 557 |
| IBM Developer Kit for Java のサポート . . . . .                             | 558 |
| IBM Developer Kit for Java の関連情報 . . . . .                             | 558 |
| Java Naming and Directory Interface . . . . .                          | 558 |
| JavaMail . . . . .   | 559 |
| Java 印刷サービス . . . . .  | 559 |

|                              |            |
|------------------------------|------------|
| <b>付録. 特記事項</b> . . . . .    | <b>563</b> |
| プログラミング・インターフェース情報 . . . . . | 564        |
| 商標 . . . . .                 | 565        |
| 使用条件 . . . . .               | 565        |



---

# IBM Developer Kit for Java



IBM Developer Kit for Java™ は、iSeries™ サーバー環境で使用するために最適化されています。Java プログラミング・インターフェースおよびユーザー・インターフェースに関する互換性を利用することで、開発者は iSeries サーバー用の独自のアプリケーションを開発できます。

IBM® Developer Kit for Java を使うと、iSeries サーバー上で Java プログラムを作成し、実行することができます。IBM Developer Kit for Java は Sun Microsystems, Inc. の Java Technology と互換性のある製品なので、Sun Microsystems, Inc. の Java Development Kit (JDK) の資料を十分理解していることが前提となります。Sun Microsystems, Inc. と IBM の情報を利用しやすくするために、Sun Microsystems, Inc. の情報へのリンクが用意されています。

Sun Microsystems, Inc. の Java Development Kit 関連資料へのリンクが何らかの理由で機能しない場合は、必要な情報について、Sun Microsystems, Inc. の HTML の参照資料を利用してください。この情報は、WWW の The Source for Java Technology [java.sun.com](http://java.sun.com) にあります。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

---

## 新機能

このトピックでは、IBM Developer Kit for Java V5R4 の変更内容について説明します。

### 新規デバッグ・インターフェース

417 ページの『Java Platform Debugger Architecture』および 393 ページの『Java Virtual Machine Profiler Interface』のトピックでは、Java Virtual Machine Tool Interface (JVMTI) について説明しています。

### 新規 CL コマンド

557 ページの『プログラム一時修正を適用する』および 405 ページの『Java でサポートされる CL コマンド』のトピックでは、「Java 仮想マシンのジョブの表示 (DSPJVMJOB)」コマンドの使用について説明しています。

### 新規 Java Cryptography Extension プロバイダー

276 ページの『Java Cryptography Extension』のトピックでは、IBM JCE FIPS JCE プロバイダーについて説明しています。

### 新規プロパティ

15 ページの『Java システム・プロパティのリスト』では、更新された J2SE バージョン 5.0 のプロパティについて説明しています。

## 新規バージョンのオプション



4 ページの『複数の Java 2 Software Development Kit のサポート』のトピックでは、J2SE バージョン 5.0 を、他の JDK のバージョンと共にインストールする場合について説明しています。

### 新規 Java ツール:

- 399 ページの『Java apt ツール』
- 402 ページの『Java pack200 ツール』
- 404 ページの『Java unpack200 ツール』

## 新情報や変更情報を識別する方法

技術上の変更が加えられたことを識別するのに役立つように、以下の情報が使用されています。

-  は、新情報や変更情報の先頭を示すマークです。
-  は、新情報や変更情報の終了を示すマークです。

新着情報やこのリリースでの変更点に関する詳細は、プログラム資料説明書を参照してください。

---

## 印刷可能 PDF

このトピックの PDF を表示およびダウンロードするには、以下のステップに従います。

本書の PDF バージョンを表示またはダウンロードするには、IBM Developer Kit for Java (約 4.6 MB) を選択します。

### PDF ファイルの保存

表示用または印刷用のために PDF をワークステーションに保存するには、以下のようにします。

1. ブラウザーで PDF を右マウス・ボタン・クリックする (上にあるリンクを右マウス・ボタン・クリック)。
2. PDF をローカルに保管するオプションを選択する。
3. PDF を保存したいディレクトリーに進む。
4. 「保存」をクリックする。

### Adobe Reader のダウンロード

- 1 これらの PDF を表示または印刷するには、システム上に Adobe Reader がインストールされていることが
- 1 必要です。このプログラムの無償のコピーは Adobe Web サイト
- 1 ([www.adobe.com/products/acrobat/readstep.html](http://www.adobe.com/products/acrobat/readstep.html))  からダウンロードできます。

---

## IBM Developer Kit for Java をインストールして構成する

IBM Developer Kit for Java を初めて使用する場合は、ここに示す手順に従ってインストールと構成を行い、単純な Hello World Java プログラムを実行して、使い方に慣れてください。

1 ページの『新機能』

このトピックでは、IBM Developer Kit for Java V5R4 の変更内容について説明します。



12 ページの『IBM Developer Kit for Java 用に iSeries サーバーをカスタマイズする』

iSeries サーバーに IBM Developer Kit for Java をインストールした後、システムをカスタマイズすることができます。

6 ページの『Java パッケージをダウンロードしてインストールする』

iSeries サーバー上への Java パッケージのダウンロード、インストール、および使用を効率的に行うには、以下を参照してください。

34 ページの『リリース間の互換性』

Java クラス・ファイルは、Sun がサポートを停止または変更したフィーチャーを利用していない限り、上位互換です (JDK 1.1.x - 1.2.x - 1.3.x - 1.4.x - 1.5.x)。

## IBM Developer Kit for Java をインストールする

IBM Developer Kit for Java をインストールすると、iSeries サーバー上で Java プログラムを作成し、実行することができます。

ライセンス・プログラム 5722-JV1 は、システム CD と共に配送されるため、JV1 はプリインストールされています。「ライセンス・プログラムの処理 (GO LICPGM)」コマンドを入力し、オプション 10 (表示) を選択してください。このライセンス・プログラムがリストされていない場合は、以下のステップを実行してください。

1. コマンド行で、GO LICPGM コマンドを入力します。
2. オプション 11 (ライセンス・プログラムの導入) を選択します。
3. ライセンス・プログラム (LP) 5769-JV1 \*BASE についてオプション 1 (導入) を選択し、インストールしたい Java Development Kit (JDK) に適合するオプションを選択します。インストールしたいオプションがリストに表示されていない場合は、「オプション」フィールドにオプション 1 (導入) を入力することにより、それをリストに追加することができます。「ライセンス・プログラム」フィールドに 5722JV1 と入力し、「プロダクト・オプション」フィールドにオプション番号を入力します。

注: 一度に複数のオプションをインストールすることができます。

iSeries サーバーに IBM Developer Kit for Java をインストールした後、システムをカスタマイズすることができます。

IBM Developer Kit for Java の入門情報については、初めての Hello World Java プログラムを実行するを参照してください。

### 「ライセンス・プログラムの復元」コマンドを使ってライセンス・プログラムをインストールする

サーバーが新規の場合、「ライセンス・プログラムのインストール」画面でリストされているプログラムは、LICPGM インストール・システムによってサポートされています。時折、使用可能になった新しいプログラムが、サーバー上のライセンス・プログラムとしてリストされないことがあります。インストールしたいプログラムでこの状況になったときは、インストールするために「ライセンス・プログラムの復元 (RSTLICPGM)」コマンドを使用する必要があります。

「ライセンス・プログラムの復元」コマンドを使ってライセンス・プログラムをインストールする方法は、以下のとおりです

1. ライセンス・プログラムが含まれているテープまたは CD-ROM を、適切なドライブに入れる。
2. iSeries コマンド入力行で、次のように入力する。

RSTLICPGM

その後、Enter キーを押します。

「ライセンス・プログラムの復元 (RSTLICPGM)」画面が表示されます。

3. 「プロダクト (Product)」フィールドに、インストールしたいライセンス・プログラムの ID 番号を入力する。
4. 「装置 (Device)」フィールドで、インストール装置を指定する。

注: 磁気テープ・ドライブからインストールする場合は、装置 ID は常に **TAPxx** という形式になります。ここで **xx** は **01** のような番号です。

5. ライセンス・プログラムの復元画面の他のパラメーターはデフォルトの設定のままにする。Enter キーを押します。
6. さらにパラメーターが表示される。これもデフォルトの設定のままにします。Enter キーを押します。プログラムのインストールが開始されます。

ライセンス・プログラムのインストールが完了すると、「ライセンス・プログラムの復元」画面が再び表示されます。

## 複数の Java 2 Software Development Kit のサポート

iSeries サーバーでは、複数のバージョンの Java Development Kits (JDKs) および Java 2 Software Development Kit (J2SDK), Standard Edition がサポートされています。

注: この資料で、JDK という語は、状況に従い、JDK および J2SDK がサポートされる任意のバージョンを指します。通常、JDK が現れる状況では、特定のバージョンとリリース番号が参照されます。

iSeries サーバーは、複数の JDK の同時使用をサポートしていますが、それは、複数の Java 仮想マシンを通してのみ行われます。単一の Java 仮想マシンは、1 つの指定された JDK を実行します。

使用しているまたは使用したい JDK を見つけてから、インストールする調整オプションを選択します。一度に複数の JDK をインストールするには、3 ページの『IBM Developer Kit for Java をインストールする』を参照してください。システム・プロパティー `java.version` で、実行する JDK を決められます。Java 仮想マシンを起動、実行しているとき、その `java.` バージョンのシステム・プロパティーを変更しても有効になりません。

注: V5R3 およびそれ以降では、オプション 1 (JDK 1.1.6)、オプション 2 (JDK 1.1.7)、オプション 3 (JDK 1.2.2)、およびオプション 4 (JDK 1.1.8) はサポートされません。以下の表に、このリリースでサポートされる J2SDK をリストしています。

| オプション | JDK                   | java.home                     | java.version |
|-------|-----------------------|-------------------------------|--------------|
| 5     | 1.3                   | /QIBM/ProdData/Java400/jdk13/ | 1.3          |
| 6     | 1.4                   | /QIBM/ProdData/Java400/jdk14/ | 1.4          |
| 7     | 1.5 (J2SE 5.0 と呼ばれます) | /QIBM/ProdData/Java400/jdk15/ | 1.5          |

この複数の JDK の環境で選択されるデフォルトの JDK は、どの 5722-JV1 オプションがインストールされているかによって異なります。以下の表は、その例です。

| インストール        | 入力         | 結果  |
|---------------|------------|---|
| オプション 5 (1.3) | java Hello | J2SDK, Standard Edition バージョン 1.3 が実行される。 |

| インストール                         | 入力         | 結果  |
|--------------------------------|------------|---|
| オプション 6 (1.4)                  | java Hello | J2SDK, Standard Edition バージョン 1.4 が実行される。 |
| オプション 5 (1.3) およびオプション 6 (1.4) | java Hello | J2SDK, Standard Edition バージョン 1.4 が実行される。 |

注: JDK を 1 つだけインストールしている場合は、その JDK がデフォルトになります。複数の JDK をインストールしている場合は、優先順位は以下のとおりです。

1. オプション 6 (1.4)
2. オプション 5 (1.3)
3. オプション 7 (1.5)

## IBM Developer Kit for Java の拡張機能をインストールする

拡張機能は、コア・プラットフォームの機能を拡張するために使用できる Java クラスのパッケージです。拡張機能は、1 つまたは複数の ZIP ファイルまたは JAR ファイルのパッケージであり、拡張クラス・ローダーによって Java 仮想マシンにロードされます。

拡張機能のメカニズムにより、Java 仮想マシンでは、システム・クラスを使用するのと同じ方法で拡張クラスを使用することができます。また、拡張機能のメカニズムは、J2SDK バージョン 1.2 以上または Java 2 Runtime Environment, Standard Edition バージョン 1.2 以上で拡張機能がインストールされていない場合には、指定の URL からそれらを取り出す方法を提供します。

拡張機能のためのいくつかの JAR ファイルは、iSeries サーバーと共に出荷されます。これらの拡張機能のいずれかをインストールするときには、以下のコマンドを入力してください。

```
ADDLNK OBJ('/QIBM/ProdData/Java400/ext/extensionToInstall.jar')
NEWLNK('/QIBM/UserData/Java400/ext/extensionToInstall.jar')
LNKTYPE(*SYMBOLIC)
```

ここで、

`extensionToInstall.jar`

は、インストールしたい拡張機能を含む ZIP または JAR ファイルの名前です。

注: /QIBM/UserData/Java400/ext ディレクトリーに拡張機能のための JAR ファイルが格納されている可能性があります。これは IBM によって提供されたものではありません。

/QIBM/UserData/Java400/ext ディレクトリー内の拡張機能へのリンクの作成またはファイルの追加を行うと、iSeries サーバー上で実行されているそれぞれの Java 仮想マシン について、拡張クラス・ローダーによって検索されるファイルのリストが変更されます。iSeries サーバー上の他の Java 仮想マシンの拡張クラス・ローダーに影響を与えたくないが、拡張機能へのリンクを作成したり、IBM が iSeries サーバーと共に出荷したものではない拡張機能をインストールしたい場合は、以下のステップに従ってください。

1. 拡張機能をインストールするためのディレクトリーを作成します。iSeries コマンド行から「ディレクトリーの作成 (MKDIR)」コマンドを使用するか、または Qshell インタープリターから `mkdir` コマンドを使用します。
2. 作成したディレクトリーに拡張機能の JAR ファイルを置きます。

- 新しいディレクトリーを `java.ext.dirs` プロパティーに追加します。新しいディレクトリーを `java.ext.dirs` プロパティーに追加するには、`iSeries` コマンド行から `JAVA` コマンドの `PROP` フィールドを使用します。

新しいディレクトリーの名前が `"/home/username/ext"` で、拡張機能ファイルの名前が `extensionToInstall.jar`、`Java` プログラムの名前が `Hello` の場合は、入力するコマンドは次のようになります。

```
MKDIR DIR('/home/username/ext')

CPY OBJ('/productA/extensionToInstall.jar') TODIR('/home/username/ext') または
FTP (ファイル転送プロトコル) を使って、ファイルを /home/username/ext にコピーします。

JAVA Hello PROP((java.ext.dirs '/home/username/ext'))
```

## Java パッケージをダウンロードしてインストールする

`iSeries` サーバー上への `Java` パッケージのダウンロード、インストール、および使用を効率的に行うには、以下を参照してください。

### グラフィカル・ユーザー・インターフェースを使用するパッケージ

グラフィカル・ユーザー・インターフェース (GUI) と共に使用する `Java` プログラムでは、グラフィカル表示装置能力のある表示装置を使用する必要があります。たとえば、パーソナル・コンピューター、テクニカル・ワークステーション、またはネットワーク・コンピューターなどが使用できます。Native Abstract Windowing Toolkit (NAWT) を使用すれば、`Java` アプリケーションおよびサーブレットに、`Java 2 Software Development Kit (J2SDK) Standard Edition Abstract Windowing Toolkit (AWT)` のグラフィックス機能の全機能を備えることができます。詳しくは、Native Abstract Windowing Toolkit (NAWT)を参照してください。

### 大文字小文字の区別および統合ファイル・システム

統合ファイル・システムは、大文字小文字の区別をするファイル・システムと、ファイル名とは関係していないファイル・システムの両方を提供します。`QOpenSys` は、統合ファイル・システムにある大文字小文字の区別をするファイル・システムの例です。ルート `'/'` は、大文字小文字を区別しないファイル・システムの例です。詳しくは、統合ファイル・システムを参照してください。

`JAR` またはクラスは大文字小文字を区別しないファイル・システム上に配置されますが、`Java` は引き続き大文字小文字を区別する言語です。`wrklnk '/home/Hello.class'` と `wrklnk '/home/hello.class'` は同一の結果を生成しますが、`JAVA CLASS(Hello)` と `JAVA CLASS(hello)` は別々のクラスを呼び出します。

### ZIP ファイルの処理と JAR ファイルの処理

`ZIP` ファイルと `JAR` ファイルには、一連の `Java` クラスが格納されています。これらのファイルに対して「`Java` プログラムの作成 (CRTJVAPGM)」コマンドを使用すると、格納されているクラスが検査され、マシンの内部形式に変換されます。また、指定された場合は `iSeries` マシン・コードに変換されます。`ZIP` ファイルと `JAR` ファイルは、他の個々のクラス・ファイルと同じように扱うことができます。内部マシンの形式が、これらのファイルの 1 つと関係している場合は、そのファイルと関係した状態が続きます。パフォーマンスを向上するために、内部マシン形式は将来の実行のときにクラス・ファイルの代わりに使用されます。現在の `Java` プログラムがクラス・ファイルまたは `JAR` ファイルと関連付けられているかどうかの不確実な場合は、「`Java` プログラムの表示 (DSPJVAPGM)」コマンドを使用して、`iSeries` サーバー上の `Java` プログラムに関する情報を表示してください。

IBM Developer Kit for Java の前のリリースでは、JAR ファイルまたは ZIP ファイルをいずれかの方法で変更した場合、接続されている Java プログラムが使用できなくなったため、Java を再作成しなければなりません。このことは、現在は必要ありません。多くの場合、JAR ファイルまたは ZIP ファイルを変更する場合、Java プログラムは依然有効であるため、再作成する必要はありません。JAR ファイル内で 1 つのクラス・ファイルが更新されるなどの部分的な変更が行われる場合、JAR ファイル内にある影響を受けたクラス・ファイルを再作成するだけで済みます。

JAR ファイルに対するほとんどの典型的な変更の後、Java プログラムは JAR ファイルとの接続状態を保持します。たとえば次の場合、それらの Java プログラムは JAR ファイルとの接続を保持します。

- `ajar` ツールを使用して JAR ファイルを変更するか、再作成する。
- `jar` ツールを使用して、JAR ファイルを変更するか、再作成する。
- `OS/400 COPY` コマンドを使用するか、または `Qshell cp` ユーティリティを使用して JAR ファイルを置換する。

iSeries Access for Windows<sup>®</sup> を介して、またはパーソナル・コンピューター (PC) のマップ済みドライブから統合ファイル・システム内の JAR ファイルにアクセスする場合、以下のようなときは、それらの Java プログラムは JAR ファイルとの接続を保持します。

- 別の JAR ファイルを既存の統合ファイル・システム JAR ファイルにドラッグ・アンド・ドロップする。
- `jar` ツールを使用して統合ファイル・システム JAR ファイルを変更するか、再作成する。
- PC コピー・コマンドを使用して統合ファイル・システム JAR ファイルを置換する。

JAR ファイルが変更または置換されると、それに接続されている Java プログラムは現行のものではありません。

Java プログラムが JAR ファイルに接続されたままにならない、例外が 1 つあります。ファイル転送プロトコル (FTP) を使用して JAR ファイルを置換すると、接続されていた Java プログラムは破棄されます。たとえば、このことは `FTP put` コマンドを使用して JAR ファイルを置換するときに生じます。

JAR ファイルのパフォーマンス特性の詳細については、Java ランタイムのパフォーマンスを参照してください。

## Java 拡張フレームワーク

J2SDK では、拡張機能は、コア・プラットフォームの機能を拡張するために使用できる Java クラスのパッケージです。拡張機能またはアプリケーションは、1 つまたは複数の JAR ファイルにあります。拡張機能のメカニズムにより、Java 仮想マシンでは、システム・クラスを使用するのと同じ方法で拡張クラスを使用することができます。拡張機能機構はさらに、拡張機能がまだ J2SDK または Java 2 Runtime Environment, Standard Edition にインストールされていないときに、指定した URL から拡張機能を検索することを可能にします。

拡張機能のインストール方法については、IBM Developer Kit for Java の拡張機能をインストールするを参照してください。

## Hello World Java プログラムを初めて実行する

このトピックは、プログラムを初めて実行する場合に役立ちます。

Hello World Java プログラムを起動して実行するには、次の 2 つの方法があります。

1. IBM Developer Kit for Java に付属している Hello World Java プログラムを実行する。



付属のプログラムを実行する方法は次のとおりです。

- a. 「ライセンス・プログラムの処理 (GO LICPGM)」コマンドを入力し、IBM Developer Kit for Java がインストールされていることを確認する。その後、オプション 10 (導入済みライセンス・プログラムの表示) を選択する。ライセンス・プログラム 5722-JV1 \*BASE と、少なくとも 1 つのオプションがインストール済みとしてリストされていることを確認してください。
  - b. iSeries メイン・メニューのコマンド行に、java Hello と入力する。Enter キーを押すと、Hello World Java プログラムが実行されます。
  - c. IBM Developer Kit for Java が正しくインストールされていれば、Java シェル画面に Hello World と表示される。F3 (終了) または F12 (終了) を押すと、コマンド入力画面に戻ります。
  - d. Hello World クラスが実行されない場合は、インストールが正常に完了したことを確認するか、IBM Developer Kit for Java のサポートを受けるのサービス情報を参照する。
2. また、ユーザー独自の Hello Java プログラムも実行することができます。ユーザー独自の Hello Java プログラムの作成方法については、Hello World Java プログラムを作成し、コンパイルして、実行するを参照してください。

## ネットワーク・ドライブを iSeries サーバーに割り当てる

ネットワーク・ドライブを割り当てるには、以下のステップを実行してください。

1. サーバーおよびワークステーションに iSeries Access for Windows がインストールされていることを確認してください。iSeries Access for Windows のインストールおよび構成方法の詳細は、iSeries Access for Windows のインストールを参照してください。ネットワーク・ドライブを割り当てる前に、iSeries サーバー用に接続を構成する必要があります。
2. Windows エクスプローラを開く。
  - a. Windows タスクバーの「スタート」ボタンの上で右マウス・ボタン・クリックする。
  - b. メニューの中の「エクスプローラ」をクリックする。
3. 「ツール」メニューから「ネットワーク ドライブの割り当て」を選択する。
4. iSeries サーバーへの接続に使用したいドライブを選択する。
5. サーバーへのパス名を入力する。たとえば、¥MYSERVER とします。ここで MYSERVER は iSeries サーバーの名前です。
6. 「ログオン時に再接続」が選択されていない場合、ボックスをチェック・マークを入れる。
7. 「OK」をクリックして完了する。

割り当てたドライブが、Windows エクスプローラの「すべてのフォルダ」セクションに表示されます。

## iSeries サーバー上にディレクトリーを作成する

Java アプリケーションを保管するには、iSeries サーバー上にディレクトリーを作成する必要があります。

これには、2 つの方法があります。

- 9 ページの『iSeries ナビゲーターを使って、ディレクトリーを作成する』

iSeries Access for Windows がインストールされている場合には、このオプションを選択してください。Java プログラムを、iSeries ナビゲーターを使ってコンパイル、最適化、および実行する計画がある場合は、このオプションを選択して、これらの操作が行われる正しい位置にプログラムが保管されるようにしてください。

- 9 ページの『コマンド入力行を使用してディレクトリーを作成する』

iSeries Access for Windows がインストールされていない場合は、このオプションを選択してください。

インストールに関する情報も含む、iSeries ナビゲーターについて詳しくは、「iSeries ナビゲーターの理解」を参照してください。

## iSeries ナビゲーターを使って、ディレクトリーを作成する

iSeries サーバー上にディレクトリーを作成するには、以下の手順に従ってください。

1. iSeries ナビゲーターを開く。
2. 「**My Connections**」ウィンドウのサーバー名をダブルクリックして、サインオンする。サーバーが「**My Connections**」ウィンドウに表示されていない場合は、以下の手順に従って追加してください。
  - a. 「**ファイル (File)**」 --> 「**接続の追加... (Add Connection...)**」をクリックする。
  - b. 「**システム (System)**」 フィールドにサーバーの名前を入力する。
  - c. 「**次へ (Next)**」 をクリックする。
  - d. 「**デフォルトのユーザー ID を使用し、必要に応じて問い合わせる (Use default user ID, prompt as needed)**」 フィールドにユーザー ID がまだ入力されていない場合は入力する。
  - e. 「**次へ (Next)**」 をクリックする。
  - f. 「**接続の検証 (Verify Connection)**」 をクリックする。これにより、サーバーに接続できることが確認されます。
  - g. 「**完了 (Finish)**」 をクリックする。
3. 使用したい接続の下のフォルダーを展開する。「**File Systems**」という名前のフォルダーを探します。このフォルダーが見つからない場合は、iSeries ナビゲーターのインストールのときに、File Systems をインストールするオプションが選択されていません。「**スタート**」 --> 「**プログラム**」 --> 「**IBM iSeries Access for Windows**」 --> 「**選択セットアップ**」を選択して、iSeries Navigator の File Systems オプションをインストールする必要があります。
4. 「**File Systems**」フォルダーを展開し、「**Integrated File System**」フォルダーを探す。
5. 「**Integrated File System**」フォルダーを展開した後、「**Root**」フォルダーを展開する。「**Root**」フォルダーを展開すると、iSeries コマンド入力行で WRKLNK (') コマンドを実行したときと同じ構造が表示されます。
6. サブディレクトリーを追加したいフォルダーで右マウス・ボタン・クリックします。「**新規フォルダー**」を選択して、作成したいサブディレクトリーの名前を入力します。

## コマンド入力行を使用してディレクトリーを作成する

iSeries サーバー上にディレクトリーを作成するには、以下の手順に従ってください。

1. iSeries サーバーにサインオンする。
2. コマンド入力行で、次のように入力する。

```
CRDIR DIR('/mydir')
```

ここで *mydir* は作成するディレクトリーの名前です。

**Enter** キーを押す。

画面の最下部に、「**ディレクトリーが作成されました (Directory created)**」というメッセージが表示されます。

## HelloWorld Java プログラムを作成、コンパイル、および実行する

単純な Hello World Java プログラムを作成することは、IBM Developer Kit for Java を十分理解する上での第一歩となります。

独自の Hello World Java プログラムを作成し、コンパイルして実行する手順は、次のとおりです。

1. ネットワーク・ドライブを iSeries サーバーに割り当てる。
2. Java アプリケーション用に、iSeries サーバー上にディレクトリーを作成する。
3. 統合ファイル・システム内に、ASCII テキスト・ファイルとしてソース・ファイルを作成する。Java アプリケーションのコーディングには、統合開発環境 (IDE) 製品を使用することも、Windows のメモ帳のようなテキスト・エディターを使用することもできます。
  - a. テキスト・ファイルの名前を HelloWorld.java にする。ファイルの作成方法と編集方法の詳細については、Java ソース・ファイルを作成して編集するを参照してください。
  - b. ファイルに次のソース・コードが含まれていることを確認する。

```
class HelloWorld {  
    public static void main (String args[]) {  
        System.out.println("Hello World");  
    }  
}
```

4. Compile the source file.
  - a. 「環境変数の処理 (WRKENVVAR)」コマンドを入力し、CLASSPATH 環境変数を調べる。CLASSPATH 変数が存在しない場合は、CLASSPATH 変数を追加して '.' (現行ディレクトリー) に設定する。CLASSPATH 変数が存在する場合は、パス名リストの先頭が '.' になっていることを確認する。CLASSPATH 環境変数の詳細については、Java クラスパスを参照してください。
  - b. 「Qshell の開始 (STRQSH)」コマンドを入力して、Qshell インタープリターを開始する。
  - c. 「ディレクトリーの変更 (cd)」コマンドを使用して、現行ディレクトリーを HelloWorld.java ファイルが入っている統合ファイル・システムのディレクトリーに変更する。
  - d. javac に続けて、ディスクに保管したファイルの名前を入力する。たとえば、javac HelloWorld.java と入力する。
5. 統合ファイル・システムのクラス・ファイル上のファイル権限を設定する。
6. Java アプリケーションを最適化する。
  - a. QSH コマンド入力 行で、次のように入力する。

```
system "CRTJVAPGM '/mydir/myclass.class' OPTIMIZE(20)"
```

ここで、mydir は Java アプリケーションを保管したディレクトリーのパス名、myclass はコンパイルされた Java アプリケーションの名前です。

**注:** 最適化レベルは、最大で 40 まで設定できます。最適化レベルを 40 にすると、Java アプリケーションの効率が向上しますが、デバッグ機能が制限されます。Java アプリケーションの開発を始めた初期の段階では、最適化レベルを 20 に設定し、アプリケーションのデバッグが容易になるようにすることをお勧めします。OPTIMIZE パラメーターの詳細については、CRTJVAPGM コマンドを参照してください。

- b. **Enter** キーを押す。

メッセージが表示され、Java プログラムのクラスが作成されます。

7. クラス・ファイルを実行する。
  - a. Java クラスパスが正しく設定されていることを確認する。



- b. Qshell コマンド入力行で、java に続けて HelloWorld と入力し、Java 仮想マシンで HelloWorld.class を実行する。たとえば、java HelloWorld と入力します。iSeries サーバーで、「Java プログラムの実行 (RUNJAVA)」コマンドを使用して、HelloWorld.class を実行することもできます。
- c. すべてが正しく入力されていれば、画面に "Hello World" と表示されます。シェル・プロンプト (デフォルトでは \$) が表示され、Qshell が次のコマンドを受けける準備ができたことを示します。
- d. F3 (Exit (終了)) または F12 (Disconnect (切断)) を押して、コマンド入力画面に戻る。

iSeries サーバー上のタスクを動作させるためのグラフィカル・ユーザー・インターフェースである iSeries ナビゲーターを利用して、簡単に Java アプリケーションをコンパイル、最適化、および実行することもできます。手順については、406 ページの『Java でサポートされる iSeries ナビゲーター・コマンド』を参照してください。インストールに関する情報も含め、iSeries ナビゲーターについて詳しくは、iSeries ナビゲーターの理解を参照してください。

## Java ソース・ファイルを作成および編集する

Java ソース・ファイルを作成および編集するには、次のようないくつかの方法があります。

### iSeries Access for Windows を使用する

Java ソース・ファイルは、iSeries サーバーの統合ファイル・システムにある、ASCII テキスト・ファイルです。

iSeries Access for Windows や、ワークステーション・ベースのエディターを使用して、Java ソース・ファイルを作成および編集することができます。

### ワークステーション上で編集する

Java ソース・ファイルをワークステーション上で作成することができます。その場合は、作成したファイルをファイル転送プロトコル (FTP) を使って統合ファイル・システムに転送します。

ワークステーション上で Java ソース・ファイルを作成し、編集する方法は次のとおりです。

1. 任意のエディターを使って、ワークステーション上で ASCII ファイルを作成する。
2. iSeries サーバーに FTP 接続する。
3. ASCII 形式が維持されるように、ソース・ファイルをバイナリー・ファイルとして統合ファイル・システムのディレクトリーに転送する。

### EDTF を使用する

任意のファイル・システムからファイルを編集するには、EDTF CL コマンドを使用することができます。これは、ストリーム・ファイルやデータベース・ファイルを編集するための原始ステートメント入力ユーティリティー (SEU) と類似のエディターです。詳しくは、EDTF CL commandを参照してください。

### 原始ステートメント入力ユーティリティーを使用する

原始ステートメント入力ユーティリティー (SEU) を使うと、Java ソース・ファイルをテキスト・ファイルとして作成することができます。

SEU を使って Java ソース・ファイルをテキスト・ファイルとして作成する方法は次のとおりです。

1. SEU を使ってソース・ファイル・メンバーを作成する。

- 「ストリーム・ファイルへのコピー (CPYTOSTMF)」コマンドを使用して、ソース・ファイル・メンバーを統合ファイル・システム・ストリーム・ファイルにコピーする。このとき、データは ASCII コードに変換されます。

ソース・コードを変更する必要がある場合は、SEU を使ってデータベース・メンバーを変更し、ファイルを再びコピーしてください。

保管ファイルについては、統合ファイル・システム内のファイルを参照してください。

---

## IBM Developer Kit for Java 用に iSeries サーバーをカスタマイズする

iSeries サーバーに IBM Developer Kit for Java をインストールした後、システムをカスタマイズすることができます。

カスタマイズ可能な事項について詳しくは、以下を参照してください。

### Java クラスパス

Java<sup>(TM)</sup> 仮想マシンは、実行時にクラスを検索するために、Java クラスパスを使用します。また、Java のコマンドとツールも、クラスパスを使ってクラスの位置を判別します。デフォルトのシステム・クラスパス、CLASSPATH 環境変数、および classpath コマンド・パラメーターはすべて、クラスを探すときにどのディレクトリーを検索するかを指定するために使用されます。

Java 2 Software Development Kit (J2SDK) では、java.ext.dirs プロパティにより、ロードされる拡張のクラスパスが決定されます。詳細については、IBM Developer Kit for Java の拡張機能をインストールするを参照してください。

デフォルトのブートストラップ・クラスパスはシステムによって定義されており、変更できません。サーバーでは、IBM Developer Kit、Native Abstract Window Toolkit (NAWT)、および他のシステム・クラスに属するクラスをどこで検索するかを、デフォルトのブートストラップ・クラスパスによって指定します。

これら以外のクラスをシステム上で検出するには、CLASSPATH 環境変数または classpath パラメーターを使用して、検索対象のクラスパスを指定する必要があります。ツールやコマンドで classpath パラメーターを使用すると、CLASSPATH 環境変数で指定されている値は無効になります。

CLASSPATH 環境変数の設定には、「環境変数の処理 (WRKENVVAR)」コマンドを使用します。WRKENVVAR の画面から、CLASSPATH 環境変数の追加や変更を行うことができます。CLASSPATH 環境変数を追加する場合は「環境変数の追加 (ADDENVVAR)」コマンドを、CLASSPATH 環境変数を変更する場合は「環境変数の変更 (CHGENVVAR)」コマンドを使用します。

CLASSPATH 環境変数の値はパス名のリストであり、コロン (;) によって分けられています。これは、特定のクラスを探すために検索されます。パス名は、0 または複数の一連のディレクトリー名です。これらのディレクトリー名の後には、ディレクトリーの名前、ZIP ファイル、または JAR ファイル (統合ファイル・システムで検索する) が続きます。パス名のコンポーネントはスラッシュ (/) 文字によって分けられています。ピリオド (.) を使用して、現行作業ディレクトリーを示します。

Qshell インタープリターで使用可能なエクスポート・ユーティリティーを使うと、Qshell 環境で CLASSPATH 変数を設定できます。

これらのコマンドは、CLASSPATH 変数をユーザーの Qshell 環境に追加し、それを値 `"./myclasses.zip:/Product/classes"` に設定します。

- 次に、Qshell 環境で CLASSPATH 変数を設定するコマンドを示します。

```
export -s CLASSPATH=./myclasses.zip:/Product/classes
```

- 次に、コマンド行から CLASSPATH 変数を設定するコマンドを示します。

```
ADDENVVAR ENVVAR(CLASSPATH) VALUE("./myclasses.zip:/Product/classes")
```

J2SDK は最初にブートストラップ・クラスパスを検索してから、次に拡張ディレクトリーを検索し、その後クラスパスを検索します。上記の例のコードでの J2SDK の検索順序は次のようになります。

1. sun.boot.class.path プロパティのブートストラップ・クラスパス
2. java.ext.dirs プロパティの拡張ディレクトリー
3. 現行作業ディレクトリー
4. 「ルート」(/) ファイル・システムにある myclasses.zip ファイル
5. 「ルート」(/) ファイル・システムにある、プロダクト・ディレクトリーにあるクラス・ディレクトリー

Qshell 環境に入ると、CLASSPATH 変数が環境変数に設定されます。クラスパス・パラメーターはパス名のリストを指定します。この構文は CLASSPATH 環境変数の構文と同じです。クラスパス・パラメーターは、次のツールとコマンドで使用できます。

- Qshell の java コマンド
- javac ツール
- javah ツール
- javap ツール
- javadoc ツール
- rmic ツール
- 「Java プログラムの実行 (RUNJVA)」コマンド

これらのコマンドについて詳しくは、IBM Developer Kit for Java 用のコマンドとツールを参照してください。これらのコマンドやツールで classpath パラメーターを使用すると、CLASSPATH 環境変数は無視されます。

CLASSPATH 環境変数をオーバーライドするには、java.class.path プロパティを使用します。他のプロパティと同様に、java.class.path プロパティを変更するには、SystemDefault.properties ファイルを使用します。SystemDefault.properties ファイルの値は、CLASSPATH 環境変数をオーバーライドします。SystemDefault.properties ファイルについては、SystemDefault.properties ファイルを参照してください。

J2SDK では、さらに -Xbootclasspath オプションは、クラスの検索時にシステムがどのディレクトリーを検索するかに影響します。-Xbootclasspath/a:path を使用すると、デフォルトのブートストラップ・クラスパスの後に path が付加され、/p:path と指定すると、デフォルトのブートストラップ・クラスパスの前に path が付加され、:path と指定すると、ブートストラップ・クラスパスは path によって置き換えられます。

注: -Xbootclasspath を指定すると、システム・クラスが見つからなかったり、システム・クラスが誤ってユーザー定義クラスで置き換えられた場合に結果が保証されないため、注意が必要です。このため、ユーザー指定のクラスパスの前にデフォルトのシステム・クラスパスが検索されるように指定することを勧めます。

プログラムのランタイム環境を決定する Java システム・プロパティの詳細については、Java システム・プロパティを参照してください。

詳しくは、プログラムおよび CL コマンド API または統合ファイル・システムを参照してください。

## Java システム・プロパティー

Java システム・プロパティーにより、Java プログラムを実行する環境が決まります。Java システム・プロパティーは、i5/OS™ のシステム値や環境変数と似ています。

Java 仮想マシン (JVM) のインスタンスを開始すると、その JVM に影響するシステム・プロパティーの値が設定されます。

Java システム・プロパティーのデフォルト値を使用するか、以下の方法でそれらの値を指定できます。

- Java プログラムを開始するときにコマンド行 (または Java Native Interface (JNI) 呼び出し API) にパラメーターを追加する。
- QIBM\_JAVA\_PROPERTIES\_FILE ジョブ・レベル環境変数を使用して特定のプロパティー・ファイルを指し示す。以下に例を示します。

```
ADDENVVAR ENVVAR(QIBM_JAVA_PROPERTIES_FILE)
VALUE(/qibm/userdata/java400/mySystem.properties)
```

- user.home ディレクトリーに作成する SystemDefault.properties ファイルを作成する。
- /qibm/userdata/java400/SystemDefault.properties ファイルを使用する。

i5/OS および JVM が、以下の優先順序で Java システム・プロパティーの値を決定します。

1. コマンド行または JNI 呼び出し API
2. QIBM\_JAVA\_PROPERTIES\_FILE 環境変数
3. user.home SystemDefault.properties ファイル
4. /QIBM/UserData/Java400/SystemDefault.properties
5. デフォルト・システム・プロパティーの値

## SystemDefault.properties ファイル

SystemDefault.properties ファイルは、Java 環境のデフォルト・プロパティーを指定できる、標準の Java プロパティー・ファイルです。

ホーム・ディレクトリーにある SystemDefault.properties ファイルは、/QIBM/UserData/Java400 ディレクトリーにある SystemDefault.properties よりも優先されます。

/YourUserHome/SystemDefault.properties ファイルで設定するプロパティーは、以下の特定の Java 仮想マシンにのみ影響します。

- 別の user.home プロパティーを指定せずに開始する JVM
- プロパティー user.home = /YourUserHome/ を指定して他のユーザーが開始する JVM

## 例: SystemDefault.properties ファイル

以下の例では、いくつかの Java プロパティーを設定しています。

```
#Comments start with pound sign
#Use J2SDK 1.4
java.version=1.4
#This sets a special property
myown.propname=6
```

システム・プロパティーについて詳しくは、以下のページを参照してください。

Java システム・プロパティ

Java システム・プロパティのリスト

## Java システム・プロパティのリスト

Java システム・プロパティにより、Java プログラムのランタイム環境が決まります。Java システム・プロパティは、i5/OS のシステム値や環境変数と似ています。

Java 仮想マシン (JVM) を開始すると、JVM のそのインスタンスのシステム・プロパティが設定されます。Java システム・プロパティの値の指定方法について詳しくは、以下のページを参照してください。

Java システム・プロパティ

SystemDefault.properties ファイル

Java システム・プロパティについて詳しくは、Java Secure Socket Extension (JSSE) システム・プロパティを参照してください。

以下の表には、サポートされるバージョンの Java 2 Software Development Kit (J2SDK) Standard Edition の Java システム・プロパティをリストしています。

- J2SDK バージョン 1.3
- J2SDK バージョン 1.4
- J2SE バージョン 5.0

この表では、各プロパティごとに、プロパティの名前と、適用されるデフォルト値または簡略説明をリストしています。この表には、J2SDK のバージョンによって値が異なるシステム・プロパティが示されています。デフォルト値がリストされている列に、種々の J2SDK のバージョンが示されていない場合は、サポートされているすべてのバージョンの J2SDK でそのデフォルト値が使用されます。

|                   |  |
|-------------------|--|
| awt.toolkit       | sun.awt.motif.MToolkit<br><br>os400.awt.native=true または java.awt.headless=true が設定されていない場合、awt.toolkit は設定されません。   |
| file.encoding     | ISO8859_1 (デフォルト値)<br><br>コード化文字セット ID (CCSID) を、対応する ISO ASCII CCSID にマップします。また、file.encoding 値の集合を、その ISO ASCII (CCSID) を表す Java 値に設定します。file.encoding に指定可能な値と、それに最も近い iSeries CCSID との関係を示した表については、file.encoding の値と iSeries CCSIDを参照してください。 |
| file.encoding.pkg | sun.io   |
| file.separator    | / (スラッシュ)  |

|                    |  |
|--------------------|--|
| java.awt.headless  | <ul style="list-style-type: none"> <li>• J2SDK v1.3: このプロパティーは、J2SDK v.1.3 を実行している場合は使用できません。</li> <li>• J2SDK v1.4: false (デフォルト値)</li> <li>• J2SE 5.0: false</li> </ul> <p>このプロパティーは、Abstract Windowing Toolkit (AWT) API をヘッドレス・モードで操作するかどうかを指定します。デフォルト値の false では、os400.awt.native を true に設定して AWT を使用可能にしている場合にのみ、全 AWT 機能が使用可能になります。このプロパティーを true に設定する場合、ヘッドレス AWT モードがサポートされ、さらに明示的に os400.awt.native が true に設定されることとなります。</p>   |
| java.class.path    | <p>. (ピリオド) (デフォルト値)</p> <p>i5/OS がクラスを見つけるために使用するパスを指定します。デフォルトはユーザー指定の CLASSPATH です。</p>   |
| java.class.version | <ul style="list-style-type: none"> <li>• J2SDK v1.3: 47.0</li> <li>• J2SDK v1.4: 48.0</li> <li>• J2SE 5.0: 49.0</li> </ul>   |
| java.compiler      | <p>jite_de (デフォルト値)</p> <p>Just-In-Time (JIT) コンパイラーを使用してコードをコンパイルするか (jite)、JIT コンパイラーと直接処理の両方を使用してコンパイルするか (jite_de) を指定します。</p>   |
| java.ext.dirs      | <p>J2SDK v1.3:</p> <ul style="list-style-type: none"> <li>• /QIBM/ProdData/Java400/jdk13/lib/ext:</li> <li>• /QIBM/UserData/Java400/ext</li> </ul> <p>J2SDK v1.4:</p> <ul style="list-style-type: none"> <li>• /QIBM/ProdData/OS400/Java400/jdk/lib/ext:</li> <li>• /QIBM/ProdData/Java400/jdk14/lib/ext:</li> <li>• /QIBM/UserData/Java400/ext (デフォルト値)</li> </ul> <p>J2SE 5.0:</p> <ul style="list-style-type: none"> <li>• /QIBM/ProdData/Java400/jdk15/lib/ext:</li> <li>• /QIBM/UserData/Java400/ext</li> </ul> |
| java.home          | <p>J2SDK v1.3: /QIBM/Prodata/Java400/jdk13</p> <p>J2SDK v1.4: /QIBM/ProdData/Java400/jdk14 (デフォルト値)</p> <p>J2SDK v1.5: /QIBM/ProdData/Java400/jdk15</p> <p>詳細は、複数の Java Development Kits (JDKs) のサポートを参照してください。</p>  |
| java.library.path  | <ul style="list-style-type: none"> <li>• /QSYS.LIB/QSHELL.LIB:/QSYS.LIB/QGPL.LIB:</li> <li>• /QSYS.LIB/QTEMP.LIB:/QSYS.LIB/QDEVELOP.LIB:</li> <li>• /QSYS.LIB/QBLDSYS.LIB:/QSYS.LIB/QBLDSYSR.LIB</li> </ul> <p>(デフォルト値)</p> <ul style="list-style-type: none"> <li>• i5/OS ライブラリー・リスト</li> </ul>   |



|                              |   |
|------------------------------|---|
| java.net.preferIPv4Stack     | <ul style="list-style-type: none"> <li>• true (デフォルト値)</li> <li>• false (no's)</li> </ul> <p>デュアル・スタック・マシンでは、優先されるプロトコル・スタック (IPv4 または IPv6) と、優先されるアドレス・ファミリー・タイプ (inet4 または inet6) を設定するためのシステム・プロパティーが用意されています。デュアル・スタック・マシンの IPv6 ソケットは、IPv4 と IPv6 の両方のピアと対話できるので、デフォルトでは IPv6 スタックが優先されます。この設定は、このプロパティーを使用して変更できます。 java.net.preferIPv4Stack は J2SDK v1.4 に固有です。</p> <p>詳しくは、IPv6 プロトコル (IPv6 protocol) を参照してください。</p> |
| java.net.preferIPv6Addresses | <ul style="list-style-type: none"> <li>• true</li> <li>• false (no's) (デフォルト値)</li> </ul> <p>オペレーティング・システムで IPv6 が使用可能でも、デフォルト設定では、IPv6 アドレスよりも IPv4 マップ・アドレスが優先されます。このプロパティーによって、IPv6 (true) と IPv4 (false) のどちらのアドレスを使用するかが制御されます。 java.net.preferIPv4Stack は J2SDK v1.4 に固有です。</p> <p>詳しくは、IPv6 プロトコル (IPv6 protocol) を参照してください。</p>   |
| java.policy                  | <p>J2SDK v1.3: /QIBM/ProdData/ Java400/jdk13/lib/security/java.policy</p> <p>J2SDK v1.4: /QIBM/ProdData/OS400/ Java400/jdk/lib/security/java.policy (デフォルト値)</p> <p>J2SE v5.0: /QIBM/ProdData/ Java400/jdk15/lib/security/java.policy</p>   |
| java.specification.name      | <ul style="list-style-type: none"> <li>• Java プラットフォーム API の仕様 (デフォルト値)</li> <li>• Java 言語の仕様</li> </ul>  |
| java.specification.vendor    | Sun Microsystems, Inc.  |
| java.specification.version   | <ul style="list-style-type: none"> <li>• J2SDK v1.3: 1.3</li> <li>• J2SDK v1.4: 1.4 (デフォルト値)</li> <li>• J2SE v5.0: 1.5</li> </ul>   |
| java.use.policy              | true  |
| java.vendor                  | IBM Corporation   |
| java.vendor.url              | <a href="http://www.ibm.com">http://www.ibm.com</a>   |

|                               |   |
|-------------------------------|---|
| java.version                  | <ul style="list-style-type: none"> <li>• 1.3.1</li> <li>• 1.4.2 (デフォルト値)</li> <li>• 1.5.0</li> </ul> <p>使用する J2SDK のバージョンを決定します。</p> <p>単一の J2SDK のバージョンをインストールしている場合は、そのバージョンがデフォルトになります。インストールされていないバージョンを指定すると、エラー・メッセージが戻されます。バージョンを指定しない場合、最新の J2SDK のバージョンがデフォルトとして使用されます。</p> <p>注: java.version は、SystemDefault.properties ファイルに入っている場合、および Java Native Interface (JNI) を使用する場合は無視されます。詳しくは、複数の J2SDK のサポートを参照してください。</p> |
| java.vm.name                  | クラシック VM  |
| java.vm.specification.name    | Java 仮想マシンの仕様   |
| java.vm.specification.vendor  | Sun Microsystems, Inc.  |
| java.vm.specification.version | 1.0   |
| java.vm.vendor                | IBM Corporation   |
| java.vm.version               | <ul style="list-style-type: none"> <li>• J2SDK v1.3: 1.3</li> <li>• J2SDK v1.4: 1.4 (デフォルト値)</li> <li>• J2SE v5.0: 1.5</li> </ul>   |
| line.separator                | ¥n  |
| os.arch                       | PowerPC®  |
| os.name                       | i5/OS   |
| os.version                    | <p>V5R4M0 (デフォルト値)</p> <p>i5/OS のリリース・レベルを、Retrieve Product Information アプリケーション・プログラム・インターフェース (API) から取得します。</p>  |
| os400.awt.native              | Abstract Windowing Toolkit (AWT) API をサポートするかどうかを制御します。有効値は true および false です。java.awt.headless=true が設定されていない場合デフォルトは false で、その場合は os400.awt.native が true であることが暗黙的に示されます。   |
| os400.certificateContainer    | <p>Java Secure Sockets Layer (SSL) サポートで、開始済みの Java プログラムおよび指定済みのプロパティの証明書コンテナとして指定したものが使用されるようにします。os400.secureApplication システム・プロパティを指定すると、このプロパティは無視されます。たとえば、-Dos400.certificateContainer=/home/username/mykeyfile.kdb と入力するか、または統合ファイル・システム中の他のキー・ファイルを入力してください。</p>   |
| os400.certificateLabel        | <p>このシステム・プロパティは、os400.certificateContainer システム・プロパティと一緒に指定できます。このプロパティを指定すると、指定されているコンテナ中のどの証明書が、Secure Sockets Layer (SSL) で使用されるか選択できます。たとえば、-Dos400.certificateLabel=myCert (myCert は、証明書の作成時かインポート時にデジタル証明書マネージャー (DCM) によってその証明書に割り当てられるラベル名) と入力してください。</p>   |



|                                 |   |
|---------------------------------|---|
| os400.child.stdio.convert       | Java での stdin、stdout、および stderr に関するデータ変換を制御します。ASCII データと Extended Binary Coded Decimal Interchange Code (EBCDIC) 間のデータ変換は、Java 仮想マシンにデフォルトで存在します。このプロパティーを使用してこれらの変換をオンまたはオフにすると、それはこのプロセスが runtime.exec() メソッドを使用して開始するすべての子プロセスにも作用します。デフォルト値を参照してください。   |
| os400.class.path.security.check | 20 (デフォルト値)<br>有効値:<br><ul style="list-style-type: none"> <li>• 0<br/>セキュリティ検査なし</li> <li>• 10<br/>RUNJVA CHPATH(*IGNORE) と同等</li> <li>• 20<br/>RUNJVA CHPATH(*WARN) と同等</li> <li>• 30<br/>RUNJVA CHPATH(*SECURE) と同等</li> </ul>  |
| os400.class.path.tools          | 0 (デフォルト値)<br>有効値:<br><ul style="list-style-type: none"> <li>• 0<br/>java.class.path プロパティーに Sun ツールを含めない</li> <li>• 1<br/>java.class.path プロパティーの前に J2SDK の固有のツール・ファイルを付加する</li> </ul> <p>J2SDK v1.3 の場合、tools.jar のパスは /QIBM/ProdData/Java400/jdk13/lib/ です。</p> <p>J2SDK v1.4 の場合、tools.jar のパスは /QIBM/ProdData/OS400/Java400/jdk/lib/ です。</p> <p>J2SE v5.0 の場合、tools.jar のパスは /QIBM/ProdData/Java400/jdk15/lib/ です。</p> |
| os400.create.type               | <ul style="list-style-type: none"> <li>• interpret (デフォルト値)<br/>RUNJVA OPTIMIZE(*INTERPRET) および INTERPRET(*OPTIMIZE) または INTERPRET(*YES) と同等</li> <li>• direct<br/>その他</li> </ul>   |
| os400.define.class.cache.file   | デフォルト値は /QIBM/ProdData/Java400/QDefineClassCache.jar です。<br>JAR または ZIP ファイルの名前を指定します。Java のパフォーマンスの考慮事項の中の『ユーザー・クラス・ローダーのキャッシュの使用』を参照してください。   |

|   |   |
|---|---|
| os400.define.class.cache.hour                 | <ul style="list-style-type: none"> <li>デフォルト値 = 768</li> <li>最大 10 進値 = 9999</li> </ul> <p>10 進値を指定します。Java のパフォーマンスの考慮事項の中の『ユーザー・クラス・ローダーのキャッシュの使用』を参照してください。</p>  |
| os400.define.class.cache.maxpgms              | <ul style="list-style-type: none"> <li>デフォルト値 = 5000</li> <li>最大 10 進値 = 40000</li> </ul> <p>10 進値を指定します。Java のパフォーマンスの考慮事項の中の『ユーザー・クラス・ローダーのキャッシュの使用』を参照してください。</p>  |
| os400.defineClass.optLevel                    | 0   |
| os400.display.properties                      | この値が 'true' に設定されている場合、Java 仮想マシンのプロパティーのすべてが標準値に設定されます。その他の値は認識されません。  |
| os400.enbpfrcol                               | <ul style="list-style-type: none"> <li>0 (デフォルト値)<br/>CRTJVAPGM ENBPFRCOL(*NONE) と同等</li> <li>1<br/>CRTJVAPGM ENBPFRCOL(*ENTRYEXIT) と同等</li> <li>7<br/>CRTJVAPGM ENBPFRCOL(*FULL) と同等</li> </ul> <p>ゼロ以外の値の場合、JIT は *JVAENTRY、*JVAEXIT、*JVAPRECALL、および *JVAPOSTCALL イベントを生成します。</p> |
| os400.exception.trace                         | このプロパティーは、デバッグの目的に使用されます。このプロパティーを指定すると、JVM の終了時に、最後に出された例外が標準出力へ送信されます。  |
| os400.file.create.auth、 os400.dir.create.auth | これらのプロパティーは、ファイルやディレクトリーに割り当てる権限を指定します。このプロパティーに何も値を指定しない場合や、サポートされていない値を指定した場合、権限は共通権限 *NONE になります。  |
| os400.file.io.mode                            | このプロパティーにデフォルト (BINARY) ではなく TEXT を指定すると、ファイルの CCSID が file.encoding と異なる場合に、ファイルの CCSID を変換します。  |
| os400.gc.heap.size.init                       | これは、-Xms (初期 GC サイズを設定する) を使用する代わりに使用します。このプロパティーは i5/OS に固有なプロパティーなので、やむをえない場合以外は引き続き -Xms を使用することを強くお勧めします。このプロパティーは、主に SystemDefault.properties ファイルで初期 GC サイズを指定できるようにするために導入されました。   |

|                         |  |
|-------------------------|--|
| os400.gc.heap.size.max  | <p>これは、-Xmx (最大 GC サイズを設定する) を使用する代わりに使用します。このプロパティは i5/OS に固有なプロパティなので、やむをえない場合以外は引き続き -Xmx を使用することを強くお勧めします。このプロパティによって、SystemDefault.properties ファイルに最大 GC サイズを指定できます。</p> <p><b>注:</b> このプロパティは注意して使用してください。指定すると、-Xmx がオーバーライドされます。この値は、コンマのないキロバイト単位のサイズの整数でなければなりません。</p>   |
| os400.interpret         | <ul style="list-style-type: none"> <li>• 0 (デフォルト値)</li> </ul> <p>CRTJVAPGM INTERPRET(*NO) と同等</p> <ul style="list-style-type: none"> <li>• 1</li> </ul> <p>CRTJVAPGM INTERPRET(*YES) と同等</p>  |
| os400.jit.mmi.threshold | <p>i5/OS が JIT コンパイラーを使用してメソッドをコンパイルし、ネイティブ・マシン・インストラクションを作成するまでに、そのメソッドが Mixed-Mode Interpreter (MMI) を使用して実行する回数を設定します。通常、デフォルト値の 2000 は変更しないでください。</p> <ul style="list-style-type: none"> <li>• 値がゼロの場合、MMI は使用不可となり、メソッドは最初に呼び出されたときにコンパイルされます。</li> <li>• 値がデフォルトより低い場合、始動時間が長くなり、最終的なパフォーマンスが低下する傾向があります。</li> <li>• 値がデフォルトより高い場合、最初の、しきい値に達するまでのパフォーマンスは低下し、最終的な実行時パフォーマンスは向上する傾向があります。</li> </ul>   |
| os400.job.file.encoding | <p>このプロパティは、出力の目的だけに使用されます。これは、JVM が含まれる i5/OS ジョブのファイル・エンコードをリストします。</p>  |
| os400.optimization      | <ul style="list-style-type: none"> <li>• 0 (デフォルト値)</li> </ul> <p>CRTJVAPGM OPTIMIZE(*INTERPRET) と同等</p> <ul style="list-style-type: none"> <li>• 10</li> </ul> <p>CRTJVAPGM OPTIMIZE(10) と同等</p> <ul style="list-style-type: none"> <li>• 20</li> </ul> <p>CRTJVAPGM OPTIMIZE(20) と同等</p> <ul style="list-style-type: none"> <li>• 30</li> </ul> <p>CRTJVAPGM OPTIMIZE(30) と同等</p> <ul style="list-style-type: none"> <li>• 40</li> </ul> <p>CRTJVAPGM OPTIMIZE(40) と同等</p> |
| os400.pool.size         | <p>スレッド・ローカル・ヒープ内のそれぞれのヒープ・プールごとに使用可能になるスペース量 (キロバイト単位) を定義します。</p>  |

|                           |   |
|---------------------------|---|
| os400.run.mode            | <ul style="list-style-type: none"> <li>• interpret</li> </ul> <p>RUNJVA OPTIMIZE(*INTERPRET) および INTERPRET(*OPTIMIZE)、または INTERPRET(*YES) と同等</p> <ul style="list-style-type: none"> <li>• program_create_type</li> <li>• jitc_de (デフォルト値)</li> </ul> <p>その他</p>                                |
| os400.run.verbose         | <p>この値が 'true' に設定されている場合、バーボーズ・クラス・ローティングが標準値に設定されます。その他の値は認識されません。</p> <p>SystemDefault.properties ファイルで機能する点を除けば、このプロパティーには、CL コマンドの QSHELL または OPTION(*VERBOSE) に -verbose を指定するのと同じ効果があります。</p>  |
| os400.runtime.exec        | <ul style="list-style-type: none"> <li>• EXEC (デフォルト値)</li> </ul> <p>EXEC インターフェースを使用して、runtime.exec() 経由で機能呼び出します。</p> <ul style="list-style-type: none"> <li>• QSHELL</li> </ul> <p>Qshell インタープリターを使用して、runtime.exec() 経由で機能呼び出します。</p> <p>詳しくは、java.lang.Runtime.exec() を使用するを参照してください。</p> |
| os400.secureApplication   | <p>このシステム・プロパティー (os400.secureApplication) の使用中に開始される Java プログラムと、登録済みの保護アプリケーション名を関連付けます。デジタル証明書マネージャー (DCM) を使用すると、登録済みの保護アプリケーション名を参照できます。</p>   |
| os400.security.properties | <p>どの java.security ファイルを使用するかに関する全制御を許可します。このプロパティーを指定すると、J2SDK は、J2SDK 特有の java.security デフォルト・ファイルを含め、その他のどの java.security ファイルも使用しません。</p>  |
| os400.stderr              | <p>stderr をファイルまたはソケットにマッピングできます。デフォルト値を参照してください。</p>   |
| os400.stdin               | <p>stdin をファイルまたはソケットにマッピングできます。デフォルト値を参照してください。</p>  |
| os400.stdin.allowed       | <p>1 (デフォルト値)</p> <p>stdin を使用できる (1) か、使用できない (0) かを指定します。呼び出し元がバッチ・ジョブを実行しているときは、stdin は使用できません。</p>  |
| os400.stdio.convert       | <p>Java での stdin、stdout、および stderr に関するデータ変換を制御できます。デフォルトでは、Java 仮想マシンでは ASCII データと EBCDIC データの間のデータ変換が行われます。このプロパティーを指定して、この種の変換をオン/オフにすることができます。この指定は、現行の Java プログラムに反映されます。デフォルト値を参照してください。</p>  |
| os400.stdout              | <p>stdout をファイルまたはソケットにマッピングできます。デフォルト値を参照してください。</p>   |
| os400.xrun.option         | <p>このシステム・プロパティーは、Qshell -Xrun オプションに値を指定して使用することを許可します。JVM の始動時に実行するエージェント・プログラムを指定するには、これを使用します。</p>  |

|                             |   |
|-----------------------------|---|
| os400.verify.checks.disable | 65535 (デフォルト値)<br><br>このシステム・プロパティーの値は、数値の合計を表す文字列です。これらの値のリストについては、os400.verify.checks.disable 数値を参照してください。  |
| os400.vm.inputargs          | このプロパティーは、出力の目的だけに使用されます。これは、JVM が入力として受け取った引数を表示します。このプロパティーは、JVM 始動時に指定されたものをデバッグするために役立ちます。  |
| path.separator              | : (コロン)   |
| sun.boot.class.path         | デフォルトのブート・クラス・ローダーが必要とするすべてのファイルをリストします。この値は変更しないでください。   |
| user.dir                    | 現行作業ディレクトリーを、getcwd API を使用して取り出します。  |
| user.home                   | 初期作業ディレクトリーを、Get API (getpwnam) を使用して取り出します。user.home パス中に SystemDefault.properties ファイルを挿入して、/QIBM/UserData/Java400/SystemDefault.properties 中のデフォルト・プロパティーを一時変更できます。独自のデフォルト・プロパティー値のセットを指定するよう、iSeries サーバーをカスタマイズすることができます。   |
| user.language               | このシステム・プロパティーは Java 仮想マシンで使用され、ジョブの LANGID 値が読み取られ、その値を使って対応する言語が取り出されます。   |
| user.name                   | このシステム・プロパティーは Java 仮想マシンで使用され、有効なユーザー・プロファイル名が、承認コンピューティング・ベース (TCB) の Security セクション (Security.UserName) から取り出されます。   |
| user.region                 | このシステム・プロパティーは Java 仮想マシンで使用され、ジョブの CNTRYID 値が読み取られ、その値を使ってユーザー領域が判別されます。   |
| user.timezone               | Universal Time Coordinate (UTC) (デフォルト値): このシステム・プロパティーは Java 仮想マシンで使用され、時間帯名が、QlgRetrieveLocalInformation API を使用して取得されます。JVM は最初にシステム QLOCALE オブジェクトを探します。見つからない場合、JVM は QTIMZON システム値を参照します。QTIMZON システム値に、認識されない QTIMZON オブジェクトが含まれている場合、JVM は user.timezone をデフォルトの UTC にします。<br><br>詳しくは、WebSphere Software Information Center の Development Kit for Java でサポートされている user.timezone プロパティーの値を参照してください。 |

#### os400.stdio.convert および os400.child.stdio.convert システム・プロパティーの値:

以下の表は、os400.stdio.convert システム・プロパティーと os400.child.stdio.convert システム・プロパティーの両方のシステム値を示しています。

| 値         | 説明   |
|-----------|--|
| N (デフォルト) | 読み取りまたは書き込み中に stdio 変換は実行されません。                            |
| Y         | 読み取りまたは書き込み中の、file.encoding 値とジョブ CCSID との間のすべての stdio 変換。 |
| 1         | 読み取り中のジョブ CCSID から file.encoding への stdin データ変換のみ。         |

| 値 | 説明  |
|---|---|
| 2 | 書き込み中の file.encoding からジョブ CCSID への stdout データ変換のみ。 |
| 3 | stdin 変換と stdout 変換が両方とも実行されます。                     |
| 4 | 書き込み中の file.encoding からジョブ CCSID への stderr データ変換のみ。 |
| 5 | stdin 変換と stderr 変換が両方とも実行されます。                     |
| 6 | stdout 変換と stderr 変換が両方とも実行されます。                    |
| 7 | すべての stdio 変換が実行されます。                               |

**os400.stdin、os400.stdout、および os400.stderr システム・プロパティー値:**

以下の表は、os400.stdin、os400.stdout、および os400.stderr システム・プロパティーのシステム値を示しています。

| 値    | 例の名前         | 説明                                       | 例                                       |
|------|--------------|--|---|
| File | SomeFileName | SomeFileName は、現行ディレクトリーに対する絶対パスか相対パスです。 | file:/QIBM/UserData/Java400/Output.file |
| Port | HostName     | ポート・アドレス                                 | port:myhost:2000                        |
| Port | TCPAddress   | ポート・アドレス                                 | port:1.1.11.111:2000                    |

**os400.verify.checks.disable システム・プロパティーの値:** os400.verify.checks.disable システム・プロパティーの値は、以下のリストからの 1 つ以上の数値の合計を表す文字列です。

| 値 | 説明   |
|---|--|
| 1 | <b>ローカル・クラスのアクセス検査を省略する:</b> Java <sup>TM</sup> 仮想マシンは、ローカル・ファイル・システムからロードされたクラスについては、専用および保護対象のフィールドとメソッドに関するアクセス検査を省略します。この設定は、転送するアプリケーションに含まれている内部クラスが、そのスーパークラスの専用および保護対象のメソッドとフィールドを参照している場合に有効です。 |
| 2 | <b>早期ロード時の NoClassDefFoundError を抑制する:</b> Java 仮想マシンは、タイプ・キャストやフィールドおよびメソッドのアクセスの早期検査時に発生する NoClassDefFoundErrors を無視します。   |
| 4 | <b>LocalVariableTable 検査を省略できるようにする:</b> クラスの LocalVariableTable 中にエラーがあると、そのクラスは LocalVariableTable がないかのように稼働します。この値を指定しないと、LocaleVariableTable 中にエラーがあると ClassFormatError になります。                      |
| 7 | 実行時に使用される値。  |

値は、10 進、16 進、または 8 進形式のいずれかで指定します。ゼロより小さい値は無視されます。たとえば、このリスト中の最初の 2 つの項目を選択する場合は、以下の iSeries コマンド構文を使用します。

```
JAVA CLASS(Hello) PROP((os400.verify.checks.disable 3))
```

## 国際化対応

国際化 Java プログラムを作成することによって、Java プログラムを世界の特定の地域用にカスタマイズすることができます。時間帯、ロケール、および文字エンコード方式を使用することにより、Java プログラムが正しい時刻、場所、および言語を反映するようにできます。

詳しくは、以下を参照してください。

**時間帯** 時間帯に依存する Java プログラムが正しい時刻を使用するように、サーバー上で時間帯を構成する方法について確認してください。

**Java ロケール** この Java ロケールのリストを使用して、Java プログラムが地域の言語、文化データ、および特定の文字のサポートを提供するようにしてください。

**文字エンコード** Java プログラムが他のフォーマットのデータを変換して、アプリケーションが多様な国際文字セットの情報を転送および使用できるようにする方法についてお読みください。

**例** 時間帯、ロケール、および文字エンコード方式を使用して国際化 Java プログラムを作成するための例を参照してください。

国際化について詳しくは、以下を参照してください。

- i5/OS グローバリゼーション
- Sun Microsystems, Inc. による国際化対応

### 時間帯構成

時間帯に依存する Java プログラムがある場合は、その Java プログラムが正しい時間を使用するようにサーバー上で時間帯を構成する必要があります。

Java 仮想マシン (JVM) が地方時を正しく判断するためには、現行ユーザーまたはジョブの QUTCOFFSET i5/OS システム値と LOCALE ユーザー・パラメーターの時刻情報の両方が設定してある必要があります。

- JVM は、QUTCOFFSET 値をシステムの地方時と比較して、正しい協定世界時 (UTC) を判別します。
- JVM は、Java システム・プロパティ `user.timezone` を使用して正しい地方時をシステムに戻します。

注: QUTCOFFSET と LOCALE を設定する代わりに、QTIMZON システム値を使用することができます。JVM は最初にシステム QLOCALE オブジェクトを探します。見つからない場合、JVM は QTIMZON システム値を参照します。QTIMZON システム値に、認識されない QTIMZON オブジェクトが含まれている場合、JVM は `user.timezone` をデフォルトの UTC にします。

### QUTCOFFSET と `user.timezone`

QUTCOFFSET i5/OS システム値は、システムの協定世界時 (UTC) オフセットを表します。QUTCOFFSET は、UTC (またはグリニッジ標準時) と現行システムの時刻との間の差を示します。QUTCOFFSET のデフォルト値はゼロ (+00:00) です。

QUTCOFFSET 値によって、JVM は正確な地方時の値を判別することができます。たとえば、アメリカ中部標準時 (CST) を示す QUTCOFFSET の値は、-6:00 です。アメリカ中部夏時間 (CDT) を示す QUTCOFFSET の値は、-5:00 です。

`user.timezone` Java システム・プロパティは、UTC の時刻をデフォルト値として使用します。他の値を指定しない限り、JVM は UTC の時刻を現行時刻として認識します。



QUTCOFFSET および Java システム・プロパティについて詳しくは、以下のトピックを参照してください。

i5/OSシステム値: QUTCOFFSET

Java システム・プロパティ

## LOCALE

ユーザー・プロファイルの LOCALE パラメーターは、LANG 環境変数用に使用する \*LOCALE オブジェクトを指定します。 \*LOCALE オブジェクトを Java ロケールと混同しないでください。

ロケール情報を正しく設定すれば、JVM は user.timezone プロパティを正しい時間帯に設定できます。 \*LOCALE オブジェクトが指定するデフォルトの設定は、user.timezone プロパティを設定することによってオーバーライドできます。

ロケールの使用と Java システム・プロパティの設定について詳しくは、以下のページを参照してください。

ロケール

Java システム・プロパティ

LC\_TOD カテゴリは、ロケールの夏時間調整時刻の規則と時間帯を定義します。

注: 夏時間調整時刻を使用するには、QUTCOFFSET システム値が正しいオフセットになるように設定する必要があります。

次の例は、Java 用の正しい時間帯を構成するためにロケール・オブジェクトに組み込む必要がある LC\_TOD カテゴリ情報を示しています。

```
LC_TOD
% TZDIFF is number of minutes difference from UTC (or GMT)
tzdiff 360
% Timezone name (this is the value that you would have
% passed to the JVM as the user.timezone property.)
tname "<C><S><T>"
% Remember to adjust the value of QUTCOFFSET when using
% daylight savings time (DST)
% Name used for DST.
dstname "<C><D><T>"
% DST start in this part of the US is the first Sunday in
% April at 2am
dststart 4,1,1,7200
% DST End in this area of US is Last Sunday in October.
dstend 10,-1,1,7200
% shift in seconds
dstshift 3600

END LC_TOD
```

ロケールの LC\_TOD カテゴリには tname フィールドが含まれており、これを、時間帯と同じ値に設定しなければなりません。有効な時間帯ストリングについては、java.util.TimeZone クラス用の Javadoc 参照情報を参照してください。ロケールの処理について詳しくは、以下のページを参照してください。

ロケールの処理



## Java 文字のエンコード

Java プログラムは、他のフォーマットのデータを変換して、アプリケーションが多様な国際文字セットの情報を転送および使用できるようにします。

Java 仮想マシン (JVM) は、内部では常に Unicode 形式でデータを扱います。ただし、JVM が外部とやり取りするすべてのデータは、`file.encoding` プロパティと一致したフォーマットになっています。JVM が読み取るデータは `file.encoding` から Unicode に変換され、JVM から送信されるデータは Unicode から `file.encoding` へ変換されます。

Java プログラムのデータ・ファイルは、統合ファイル・システム (IFS) に保管されています。統合ファイル・システムの中のファイルは、コード化文字セット ID (CCSID) でタグ付けされており、これによってファイル内に含まれているデータの文字エンコード方式を識別します。iSeries サーバーにおける `file.encoding` と CCSID の相関関係については、`File.encoding` の値と `iSeriesSeries CCSID` の表を参照してください。

Java プログラムが読み取るデータは、`file.encoding` と一致する文字エンコード方式であることが要求されます。Java プログラムがファイルに書き込むデータは、`file.encoding` と一致する文字エンコード方式で書き込まれます。このことは、`javac` コマンドが処理する Java のソース言語ファイル (.java ファイル) や、`.net` パッケージを使用して伝送制御プロトコル/インターネット・プロトコル (TCP/IP) ソケットを介して送受信されるデータにも当てはまります。

`System.in`、`System.out`、および `System.err` で読み書きされるデータの処理方法は、`stdin`、`stdout`、および `stderr` に割り当てられた他のソースで読み書きされるデータの処理方法とは異なります。`stdin`、`stdout`、`stderr` は通常、iSeries サーバーの EBCDIC 装置に接続されているので、データは JVM によって通常の `file.encoding` の文字コード方式から iSeries ジョブの CCSID と一致する CCSID に変換されます。`System.in`、`System.out`、`System.err` のいずれかがファイルやソケットにリダイレクトされ、`stdin`、`stdout`、`stderr` のいずれにも送信されない場合、この付加的な変換は実行されず、データは `file.encoding` と一致するエンコード方式のままになります。

Java プログラムで、`file.encoding` 以外のエンコード方式を使ってデータを読み書きする必要がある場合は、プログラムで Java の IO クラス (`java.io.InputStreamReader`、`java.io.FileReader`、`java.io.OutputStreamReader`、および `java.io.FileWriter`) を使用することができます。Java クラスを使用すれば、JVM が現在使用しているデフォルトの `file.encoding` プロパティよりも優先される `file.encoding` 値を指定することができます。

JDBC API を介して DB2/400 データベースから iSeries に転送される場合、データは iSeries の CCSID に変換されます。逆に、iSeries から DB2/400 データベースに転送される場合、データは iSeries の CCSID から変換されます。


Java ネイティブ・インターフェースを介して他のプログラムとの間で転送されるデータは、変換されません。

国際化対応について詳しくは、i5/OS グローバリゼーションを参照してください。

また、Sun Microsystems, Inc. による国際化対応 も参照してください。

**file.encoding の値と iSeries CCSID:**

次の表は、file.encoding に指定可能な値と、それに最も近い iSeries コード化文字セット識別コード (CCSID) との関係を示したものです。

file.encoding サポートについて詳しくは、Supported encodings by Sun Microsystems, Inc.  を参照してください。

| file.encoding | CCSID | 説明  |
|---------------|-------|---|
| ASCII         | 367   | 情報交換用米国標準コード                                      |
| Big5          | 950   | 8 ビット ASCII 中国語 (繁体字) BIG-5                       |
| Big5_HKSCS    | 950   | Big5_HKSCS  |
| Big5_Solaris  | 950   | Solaris zh_TW.BIG5 ロケール用の 7 つの追加の繁体字マッピングを含む Big5 |
| CNS11643      | 964   | 中国語 (繁体字) の中国文字セット                                |
| Cp037         | 037   | IBM EBCDIC 米国、カナダ、オランダ、...                        |
| Cp273         | 273   | IBM EBCDIC ドイツ、オーストリア                             |
| Cp277         | 277   | IBM EBCDIC デンマーク、ノルウェー                            |
| Cp278         | 278   | IBM EBCDIC フィンランド、スウェーデン                          |
| Cp280         | 280   | IBM EBCDIC イタリア                                   |
| Cp284         | 284   | IBM EBCDIC ラテンアメリカ・スペイン語                          |
| Cp285         | 285   | IBM EBCDIC 英国                                     |
| Cp297         | 297   | IBM EBCDIC フランス                                   |
| Cp420         | 420   | IBM EBCDIC アラビア語                                  |
| Cp424         | 424   | IBM EBCDIC ヘブライ語                                  |
| Cp437         | 437   | 8 ビット ASCII US PC                                 |
| Cp500         | 500   | IBM EBCDIC 国際                                     |
| Cp737         | 737   | 8 ビット ASCII ギリシャ語 MS-DOS                          |
| Cp775         | 775   | 8 ビット ASCII バルト語 MS-DOS                           |
| Cp838         | 838   | IBM EBCDIC タイ語                                    |
| Cp850         | 850   | 8 ビット ASCII Latin-1 多国語                           |
| Cp852         | 852   | 8 ビット ASCII Latin-2                               |
| Cp855         | 855   | 8 ビット ASCII キリル文字使用言語                             |
| Cp856         | 0     | 8 ビット ASCII ヘブライ語                                 |
| Cp857         | 857   | 8 ビット ASCII Latin-5                               |
| Cp860         | 860   | 8 ビット ASCII ポルトガル語                                |
| Cp861         | 861   | 8 ビット ASCII アイスランド語                               |
| Cp862         | 862   | 8 ビット ASCII ヘブライ語                                 |
| Cp863         | 863   | 8 ビット ASCII カナダ                                   |
| Cp864         | 864   | 8 ビット ASCII アラビア語                                 |
| Cp865         | 865   | 8 ビット ASCII デンマーク、ノルウェー                           |
| Cp866         | 866   | 8 ビット ASCII キリル文字使用言語                             |
| Cp868         | 868   | 8 ビット ASCII ウルドゥー語                                |
| Cp869         | 869   | 8 ビット ASCII ギリシャ語                                 |
| Cp870         | 870   | IBM EBCDIC Latin-2                                |
| Cp871         | 871   | IBM EBCDIC アイスランド                                 |

| file.encoding | CCSID | 説明                          |
|---------------|-------|-----------------------------|
| Cp874         | 874   | 8 ビット ASCII タイ語             |
| Cp875         | 875   | IBM EBCDIC ギリシャ語            |
| Cp918         | 918   | IBM EBCDIC ウルドゥー語           |
| Cp921         | 921   | 8 ビット ASCII バルト語            |
| Cp922         | 922   | 8 ビット ASCII エストニア語          |
| Cp930         | 930   | IBM EBCDIC 日本語拡張カタカナ        |
| Cp933         | 933   | IBM EBCDIC 韓国語              |
| Cp935         | 935   | IBM EBCDIC 中国語 (簡体字)        |
| Cp937         | 937   | IBM EBCDIC 中国語 (繁体字)        |
| Cp939         | 939   | IBM EBCDIC 日本語拡張ローマ字        |
| Cp942         | 942   | 8 ビット ASCII 日本語             |
| Cp942C        | 942   | Cp942 の変種                   |
| Cp943         | 943   | 日本語オープン環境用混合 PC データ         |
| Cp943C        | 943   | 日本語オープン環境用混合 PC データ         |
| Cp948         | 948   | 8 ビット ASCII IBM 中国語 (繁体字)   |
| Cp949         | 944   | 8 ビット ASCII 韓国語 KSC5601     |
| Cp949C        | 949   | Cp949 の変種                   |
| Cp950         | 950   | 8 ビット ASCII 中国語 (繁体字) BIG-5 |
| Cp964         | 964   | EUC 中国語 (繁体字)               |
| Cp970         | 970   | EUC 韓国語                     |
| Cp1006        | 1006  | ISO 8 ビット ウルドゥー語            |
| Cp1025        | 1025  | IBM EBCDIC キリル文字            |
| Cp1026        | 1026  | IBM EBCDIC トルコ語             |
| Cp1046        | 1046  | 8 ビット ASCII アラビア語           |
| Cp1097        | 1097  | IBM EBCDIC ペルシア語            |
| Cp1098        | 1098  | 8 ビット ASCII ペルシア語           |
| Cp1112        | 1112  | IBM EBCDIC バルト語             |
| Cp1122        | 1122  | IBM EBCDIC エストニア語           |
| Cp1123        | 1123  | IBM EBCDIC ウクライナ            |
| Cp1124        | 0     | ISO 8 ビット ウクライナ             |
| Cp1140        | 1140  | ユーロ文字を含む Cp037 の変種          |
| Cp1141        | 1141  | ユーロ文字を含む Cp273 の変種          |
| Cp1142        | 1142  | ユーロ文字を含む Cp277 の変種          |
| Cp1143        | 1143  | ユーロ文字を含む Cp278 の変種          |
| Cp1144        | 1144  | ユーロ文字を含む Cp280 の変種          |
| Cp1145        | 1145  | ユーロ文字を含む Cp284 の変種          |
| Cp1146        | 1146  | ユーロ文字を含む Cp285 の変種          |
| Cp1147        | 1147  | ユーロ文字を含む Cp297 の変種          |
| Cp1148        | 1148  | ユーロ文字を含む Cp500 の変種          |
| Cp1149        | 1149  | ユーロ文字を含む Cp871 の変種          |
| Cp1250        | 1250  | MS-Win Latin-2              |

| file.encoding   | CCSID | 説明  |
|-----------------|-------|---|
| Cp1251          | 1251  | MS-Win キリル文字使用言語                                    |
| Cp1252          | 1252  | MS-Win Latin-1                                      |
| Cp1253          | 1253  | MS-Win ギリシャ語  |
| Cp1254          | 1254  | MS-Win トルコ語   |
| Cp1255          | 1255  | MS-Win ヘブライ語  |
| Cp1256          | 1256  | MS-Win アラビア語  |
| Cp1257          | 1257  | MS-Win バルト語   |
| Cp1258          | 1251  | MS-Win ロシア語   |
| Cp1381          | 1381  | 8 ビット ASCII 中国語 (簡体字) GB                            |
| Cp1383          | 1383  | EUC 中国語 (簡体字)                                       |
| Cp33722         | 33722 | EUC 日本語   |
| EUC_CN          | 1383  | EUC 中国語 (簡体字)                                       |
| EUC_JP          | 5050  | EUC 日本語   |
| EUC_JP_LINUX    | 0     | JISX 0201、0208、EUC エンコードの日本語                        |
| EUC_KR          | 970   | EUC 韓国語   |
| EUC_TW          | 964   | EUC 中国語 (繁体字)                                       |
| GB2312          | 1381  | 8 ビット ASCII 中国語 (簡体字) GB                            |
| GB18030         | 1392  | 中国語 (簡体字)、PRC 標準                                    |
| GBK             | 1386  | 8 ビット ASCII 9 中国語 (新簡体字)                            |
| ISCI91          | 806   | インド語文字の ISCI91 エンコード                                |
| ISO2022CN       | 965   | ISO 2022 CN、中国語 (Unicode への変換のみ)                    |
| ISO2022_CN_CNS  | 965   | ISO 2022 CN 形式の CNS11643、中国語 (繁体字) (Unicode への変換のみ) |
| ISO2022_CN_GB   | 1383  | ISO 2022 CN 形式の GB2312、中国語 (簡体字) (Unicode からの変換のみ)  |
| ISO2022CN_CNS   | 965   | 7 ビット ASCII 中国語 (繁体字)                               |
| ISO2022CN_GB    | 1383  | 7 ビット ASCII 中国語 (簡体字)                               |
| ISO2022JP       | 5054  | 7 ビット ASCII 日本語                                     |
| ISO2022KR       | 25546 | 7 ビット ASCII 韓国語                                     |
| ISO8859_1       | 819   | ISO 8859-1 Latin Alphabet No. 1                     |
| ISO8859_2       | 912   | ISO 8859-2 ISO Latin-2                              |
| ISO8859_3       | 0     | ISO 8859-3 ISO Latin-3                              |
| ISO8859_4       | 914   | ISO 8859-4 ISO Latin-4                              |
| ISO8859_5       | 915   | ISO 8859-5 ISO Latin-5                              |
| ISO8859_6       | 1089  | ISO 8859-6 ISO Latin-6 (アラビア語)                      |
| ISO8859_7       | 813   | ISO 8859-7 ISO Latin-7 (ギリシャ語/ラテン語)                 |
| ISO8859_8       | 916   | ISO 8859-8 ISO Latin-8 (ヘブライ語)                      |
| ISO8859_9       | 920   | ISO 8859-9 ISO Latin-9 (ECMA-128、トルコ語)              |
| ISO8859_13      | 0     | Latin Alphabet No. 7                                |
| ISO8859_15      | 923   | ISO8859_15  |
| ISO8859_15_FDIS | 923   | ISO 8859-15、Latin alphabet No. 9                    |
| ISO-8859-15     | 923   | ISO 8859-15、Latin Alphabet No. 9                    |
| JIS0201         | 897   | 日本工業規格 X0201  |

| file.encoding      | CCSID | 説明  |
|--------------------|-------|---|
| JIS0208            | 5052  | 日本工業規格 X0208  |
| JIS0212            | 0     | 日本工業規格 X0212  |
| JISAutoDetect      | 0     | Shift-JIS、EUC-JP、ISO 2022 JP を検出し、変換する (Unicode への変換のみ) |
| Johab              | 0     | 韓国構成ハングル・エンコード (全)                                      |
| K018_R             | 878   | キリル語  |
| KSC5601            | 949   | 8 ビット ASCII 韓国語   |
| MacArabic          | 1256  | Macintosh アラビア語   |
| MacCentralEurope   | 1282  | Macintosh Latin-2                                       |
| MacCroatian        | 1284  | Macintosh クロアチア語  |
| MacCyrillic        | 1283  | Macintosh キリル文字   |
| MacDingbat         | 0     | Macintosh Dingbat                                       |
| MacGreek           | 1280  | Macintosh ギリシャ語   |
| MacHebrew          | 1255  | Macintosh ヘブライ語   |
| MacIceland         | 1286  | Macintosh アイスランド語                                       |
| MacRoman           | 0     | Macintosh Roman   |
| MacRomania         | 1285  | Macintosh ルーマニア   |
| MacSymbol          | 0     | Macintosh シンボル  |
| MacThai            | 0     | Macintosh タイ  |
| MacTurkish         | 1281  | Macintosh トルコ語  |
| MacUkraine         | 1283  | Macintosh ウクライナ   |
| MS874              | 874   | MS-Win タイ   |
| MS932              | 943   | Windows 日本語   |
| MS936              | 936   | Windows 中国語 (簡体字)                                       |
| MS949              | 949   | Windows 韓国語   |
| MS950              | 950   | Windows 中国語 (繁体字)                                       |
| MS950_HKSCS        | NA    | 中国香港特別行政区拡張を含む WindowsWindows 中国語 (繁体字)                 |
| SJIS               | 932   | 8 ビット ASCII 日本語   |
| TIS620             | 874   | タイ工業規格 620  |
| US-ASCII           | 367   | 情報交換用米国標準コード  |
| UTF8               | 1208  | UTF-8 (IBM CCSID 1208、iSeries サーバー上ではまだ使用できません)         |
| UTF-16             | 1200  | 16 ビット UCS 変換フォーマット、オプションのバイト・オーダー・マークによって示されるバイト・オーダー  |
| UTF-16BE           | 1200  | 16 ビット Unicode 変換フォーマット、ビッグ・エンディアン・バイト・オーダー             |
| UTF-16LE           | 1200  | 16 ビット Unicode 変換フォーマット、リトル・エンディアン・バイト・オーダー             |
| UTF-8              | 1208  | 8 ビット UCS 変換フォーマット                                      |
| Unicode            | 13488 | UNICODE、UCS-2   |
| UnicodeBig         | 13488 | Unicode と同じ   |
| UnicodeBigUnmarked |       | Unicode (バイト・オーダー・マークなし)                                |
| UnicodeLittle      |       | Unicode (リトル・エンディアン・バイト・オーダー)                           |

| file.encoding         | CCSID | 説明                             |
|-----------------------|-------|--------------------------------|
| UnicodeLittleUnmarked |       | UnicodeLittle (バイト・オーダー・マークなし) |

デフォルト値については、file.encoding のデフォルト値を参照してください。

### file.encoding のデフォルト値:

次の表は、Java 仮想マシンの起動時に、file.encoding が iSeries コード化文字セット識別コード (CCSID) に応じてどの値に設定されるかを示したものです。

| iSeries CCSID | file.encoding のデフォルト | 説明  |
|---------------|----------------------|---|
| 37            | ISO8859_1            | 米国、カナダ、ニュージーランド、オーストラリアでは英語。ポルトガル、ブラジルではポルトガル語。オランダではオランダ語。 |
| 256           | ISO8859_1            | 各国共通 #1   |
| 273           | ISO8859_1            | ドイツ語/ドイツ、ドイツ語/オーストリア  |
| 277           | ISO8859_1            | デンマーク語/デンマーク、ノルウェー語/ノルウェー、ノルウェー語/ノルウェー、NY                   |
| 278           | ISO8859_1            | フィンランド語/フィンランド  |
| 280           | ISO8859_1            | イタリア語/イタリア  |
| 284           | ISO8859_1            | カタロニア語/スペイン、スペイン語/スペイン                                      |
| 285           | ISO8859_1            | 英語/英国、英語/アイルランド   |
| 290           | Cp943C               | 日本語 EBCDIC 混合 (CCSID 5026) の SBCS 部                         |
| 297           | ISO8859_1            | フランス語/フランス  |
| 420           | Cp1046               | アラビア語/エジプト  |
| 423           | ISO8859_7            | ギリシャ  |
| 424           | ISO8859_8            | ヘブライ語/イスラエル   |
| 500           | ISO8859_1            | ドイツ語/スイス、フランス語/ベルギー、フランス語/カナダ、フランス語/スイス                     |
| 833           | Cp970                | 韓国語 EBCDIC 混合 (CCSID 933) の SBCS 部                          |
| 836           | Cp1383               | 中国語 (簡体字) EBCDIC 混合 (CCSID 935) の SBCS 部                    |
| 838           | TIS620               | タイ語   |
| 870           | ISO8859_2            | チェコ語/チェコ共和国、クロアチア語/クロアチア、ハンガリー語/ハンガリー、ポーランド語/ポーランド          |
| 871           | ISO8859_1            | アイスランド語/アイスランド  |
| 875           | ISO8859_7            | ギリシャ語/ギリシャ  |
| 880           | ISO8859_5            | ブルガリア (ISO 8859_5)  |

| iSeries CCSID | file.encoding のデフォルト | 説明  |
|---------------|----------------------|---|
| 905           | ISO8859_9            | トルコ拡張   |
| 918           | Cp868                | ウルドゥー語  |
| 930           | Cp943C               | 日本語 EBCDIC 混合 (CCSID 5026 と類似)                  |
| 933           | Cp970                | 韓国語/韓国  |
| 935           | Cp1383               | 中国語 (簡体字)                                       |
| 937           | Cp950                | 中国語 (繁体字)                                       |
| 939           | Cp943C               | 日本語 EBCDIC 混合 (CCSID 5035 と類似)                  |
| 1025          | ISO8859_5            | ベロルシア語/ベラルーシ、ブルガリア語/ブルガリア、マケドニア語/マケドニア、ロシア語/ロシア |
| 1026          | ISO8859_9            | トルコ語/トルコ  |
| 1027          | Cp943C               | 日本語 EBCDIC 混合 (CCSID 5035) の SBCS 部             |
| 1097          | Cp1098               | ペルシア語   |
| 1112          | Cp921                | リトアニア語/リトアニア、ラトビア語/ラトビア、バルト語                    |
| 1388          | GBK                  | 中国語 (簡体字) EBCDIC 混合 (GBK を含む)                   |
| 5026          | Cp943C               | 日本語 EBCDIC 混合 CCSID (拡張カタカナ)                    |
| 5035          | Cp943C               | 日本語 EBCDIC 混合 CCSID (拡張ローマ字)                    |
| 8612          | Cp1046               | アラビア語 (基本形のみ) (または ASCII 420 および 8859_6)        |
| 9030          | Cp874                | タイ語 (ホスト拡張 SBCS)                                |
| 13124         | GBK                  | 中国語 (簡体字) EBCDIC 混合 (GBK を含む) の SBCS 部          |
| 28709         | Cp948                | 中国語 (繁体字) EBCDIC 混合 (CCSID 937) の SBCS 部        |

## 例: 国際化 Java プログラムを作成する

特定の地域に Java プログラムをカスタマイズする必要がある場合は、Java ロケールを使用して、国際化 Java プログラムを作成できます。

Java ロケール。

国際化 Java プログラムの作成には、以下のようないくつかのタスクが関係します。

1. ロケール依存のコードとデータを分離する。たとえば、プログラム内のストリング、日付、数値などです。
2. Locale クラスを使って、ロケールを設定または取得する。
3. デフォルト・ロケールを使用したくない場合は、日付と数値をフォーマットしてロケールを指定する。
4. ストリングとその他のロケール依存データを処理するためのリソース・バンドルを作成する。



以下の例を参照してください。国際化 Java プログラムを作成するために必要なタスクを完了するための方法が参照できます。

- 例: java.util.DateFormat クラスを使用して日付を国際化する
- 例: java.util.NumberFormat クラスを使用して数値表示を国際化する
- 例: java.util.ResourceBundle クラスを使用してロケール固有データを国際化する

国際化について詳しくは、以下を参照してください。

- i5/OS グローバリゼーション
- Sun Microsystems, Inc. による国際化対応

---

## リリース間の互換性

Java クラス・ファイルは、Sun がサポートを停止または変更したフィーチャーを利用していない限り、上位互換です (JDK 1.1.x - 1.2.x - 1.3.x - 1.4.x - 1.5.x)。

リリースとリリースの間の有効性に関する情報については、The Source for Java Technology [java.sun.com](http://java.sun.com) を参照してください。

「Java プログラムの作成 (CRTJVAPGM)」コマンドを使用して iSeries 上の Java プログラムを最適化すると、クラス・ファイルに「Java プログラム (JVAPGM)」が追記されます。これらの JVAPGM の内部構造は、V4R4 で変更されました。つまり、V4R4 より前に作成された JVAPGM は、V4R4 以降のリリースでは無効です。JVAPGM を再作成する必要があります。そうしないと、システムが以前と同じ最適化レベルで JVAPGM を自動的に作成します。ただし、JAR または ZIP ファイルの場合には特に、CRTJVAPGM を手操作で実行するようお勧めします。これにより、最小のプログラム・サイズで、最良の最適化が行われます。

最適化レベル 40 で最善のパフォーマンスを得るために、i5/OS のリリースまたは JDK のバージョンの変更のたびに、CRTJVAPGM を実行することをお勧めします。CRTJVAPGM 上で JDKVER 機能を使用する場合には、これにより Sun JDK のメソッドが JVAPGM に埋め込まれるため、特にあてはまります。この実行により、パフォーマンスが大きく向上する可能性があります。ただし、以降のリリースの JDK に変更が加えられて、これらの埋め込みが無効になると、プログラムの実際の実行速度は、低レベルの最適化プログラムよりも遅くなる可能性があります。これは、適切に機能させるために、特殊なケース・コードが必要になるためです。

詳細については、Java 実行時のパフォーマンスを参照してください。

---

## IBM Developer Kit for Java によるデータベース・アクセス

IBM Developer Kit for Java を使用することにより、Java プログラムは、次のような 3 つの方法でデータベース・ファイルにアクセスすることができます。

- JDBC ドライバー。IBM Developer Kit for Java の JDBC ドライバーを使って Java プログラムがデータベース・ファイルにアクセスする方法を説明します。
- SQLJ サポート。IBM Developer Kit for Java を使用して、Java アプリケーションに組み込まれている SQL ステートメントを使用する方法を説明します。
- Java SQL ルーチン。Java ストアド・プロシージャおよび Java ユーザー定義関数を使用して、Java プログラムにアクセスする方法を説明します。



# IBM Developer Kit for Java の JDBC ドライバーを使用して iSeries データベースにアクセスする

IBM Developer Kit for Java の JDBC ドライバー (「ネイティブ」ドライバーとしても知られる) は、iSeries データベース・ファイルへのプログラマチックなアクセスを提供します。Java Database Connectivity (JDBC) API を使用すれば、Java 言語で作成されたアプリケーションは、組み込まれた構造化照会言語 (SQL) を使って JDBC データベースの機能にアクセスしたり、SQL ステートメントを実行したり、結果を検索したり、変更をデータベースに戻したりできます。また、JDBC API を使用して、分散した異機種混合環境内の複数のデータ・ソースと対話できます。

JDBC API のベースである SQL99 コマンド言語インターフェース (CLI) は、ODBC の基本となるものです。JDBC は、Java プログラム言語から、SQL 標準で定義されている抽象および概念への、自然な使いやすいまッピングを提供します。

JDBC ドライバーを使用する場合、以下を参照してください。

**JDBC 入門** iSeries サーバーでの JDBC プログラムの作成および実行に関するチュートリアルを活用できます。

**接続** アプリケーション・プログラムは、一度に複数の接続を持つことができます。Connection オブジェクトを使用することにより、JDBC 内のデータ・ソースへの接続を表すことができます。SQL ステートメントを処理するために作成された Statement オブジェクトは、Connection オブジェクトを介してデータベースに接続します。

**JVM プロパティ** ネイティブ JDBC ドライバーが使用する設定の一部は、接続プロパティを使用しては設定できません。これらの設定は、ネイティブ JDBC ドライバーが実行する JVM に対して設定しなければならないものです。

**DatabaseMetaData** DatabaseMetaData インターフェースは、提供されているデータ・ソースとの対話方法を決定するため、アプリケーション・サーバーとツールによって使用されます。アプリケーションは、DatabaseMetaData メソッドを使用しても特定のデータ・ソースの情報を入手することができます。

**例外** Java 言語では、プログラムのエラー処理機能を提供するために、例外を使用します。例外は、プログラムを実行しているときに、命令の通常フローが中断されたときに発生するイベントです。

**トランザクション** トランザクションは作業の論理単位です。トランザクションは、同時アクセスの間、データの整合性、正しいアプリケーション・セマンティクス、および矛盾のないデータのビューを提供するために使用されます。JDBC 準拠のドライバーはすべて、トランザクションをサポートしていなければなりません。

**ステートメントのタイプ** Statement インターフェースとその PreparedStatement および CallableStatement サブクラスは、データベースに対して SQL コマンドを処理するために使用されます。SQL ステートメントが処理されると、ResultSet オブジェクトが生成されます。

**ResultSets** ResultSet インターフェースは、照会の実行によって生成された結果へのアクセスを提供します。ResultSet のデータは、特定の数の列および特定の数の行を含むテーブルとして考えることができます。デフォルトでは、テーブル行は順番に検索されます。検索の対象が 1 行であれば、列の値は、任意の順序でアクセスできます。

**JDBC オブジェクト・プーリング** JDBC で使用されるオブジェクト (Connection、Statement、および ResultSet オブジェクトなど) の多くは作成に費用がかかるので、JDBC オブジェクト・プーリングを使用することにより、パフォーマンス上の大きな利点を得ることができます。オブジェクト・プールを使用すると、これらを必要になるたびに作成するのではなく、それらのオブジェクトを再利用できます。

**バッチ更新** バッチ更新サポートを使用することにより、データベースへの多くの更新を、ユーザー・プログラムとデータベースの間の単一トランザクションとして渡すことができます。一度に多くの更新を実行しなければならない場合、バッチ更新を行うことにより、パフォーマンスが大幅に向上します。

**拡張データ・タイプ** SQL3 データ・タイプという、いくつかの新しいデータ・タイプが、iSeries データベースで用意されています。SQL3 データ・タイプでは、非常に幅広い柔軟性が提供されています。これは、シリアル化された Java オブジェクト、XML (Extensible Markup Language) 文書、および音楽、製品の画像、従業員の写真やムービー・クリップといったマルチメディア・データを格納するのに理想的です。以下のような SQL3 データ・タイプがあります。

- 特殊タイプ
- ラージ・オブジェクト (バイナリー・ラージ・オブジェクト、文字ラージ・オブジェクト、2 バイト文字ラージ・オブジェクトなど)
- データ・リンク

**RowSets** RowSet の仕様は、実際のインプリメンテーションよりもフレームワークのために設計されています。RowSet インターフェースは、すべての RowSets に含まれているコア機能のセットを定義します。

**分散トランザクション** Java Transaction API (JTA) は、複雑なトランザクションをサポートします。また、Connection オブジェクトからのトランザクションの分離もサポートします。JTA と JDBC がともに機能することにより、Connection オブジェクトからトランザクションを分離します。また、複数のトランザクションが並行している状況で、単一の接続を機能させることができます。逆に、単一のトランザクションで複数の接続を機能させることもできます。

**パフォーマンス上のヒント** これらのパフォーマンス上のヒントを活用することにより、JDBC アプリケーションから、おそらく最高のパフォーマンスを得られます。

JDBC の詳細については、Sun Microsystems, Inc. の JDBC 資料を参照してください。

iSeries ネイティブ JDBC ドライバーについては詳しくは、iSeries native JDBC Driver FAQs を参照してください。

## JDBC 入門

Developer Kit for Java に付属している Java Database Connectivity (JDBC) ドライバーのことを、IBM Developer Kit for Java の JDBC ドライバーと呼びます。このドライバーは、一般にネイティブ JDBC ドライバーとも呼ばれます。

どの JDBC が必要にかなうかを選択する場合、以下の提案を考慮してください。

- データベースが置かれているサーバーで直接実行するプログラムは、パフォーマンス上の理由で、ネイティブ JDBC ドライバーを使用すべきです。これには、ほとんどのサーブレットおよび JavaServer Pages (JSP) ソリューション、および iSeries サーバーでローカルに実行するように作成されているアプリケーションが含まれます。

- リモート iSeries サーバーに接続しなければならないプログラムは、IBM Toolbox for Java JDBC ドライバーを使用します。この IBM Toolbox for Java JDBC ドライバーは JDBC の堅固なインプリメンテーションであり、IBM Toolbox for Java の一部として提供されています。Pure Java であるため、IBM Toolbox for Java JDBC ドライバーはクライアント用にセットアップするのが容易であり、サーバーのセットアップがほとんど必要ありません。
- iSeries サーバー上で実行されるプログラム、およびリモートの非 iSeries データベースに接続しなければならないプログラムは、ネイティブ JDBC ドライバーを使用し、そのリモート・サーバーへの™分散リレーショナル・データベース体系 (DRDA®) 接続をセットアップします。

JDBC の入門資料として、以下を参照してください。

**JDBC ドライバーのタイプ** このトピックでは、JDBC ドライバーのタイプを定義します。ドライバーのタイプは、データベースに接続するために使用されるテクノロジーを分類するために定義されます。

**要件** このトピックでは、以下にアクセスするために必要な要件について説明しています。

- コア JDBC
- JDBC 2.0 オプション・パッケージ
- Java Transaction API (JTA)

**JDBC チュートリアル** これは、JDBC プログラムの作成、およびネイティブ JDBC ドライバーを使用した iSeries サーバー上での実行に関する、重要な最初のステップです。

## JDBC ドライバーのタイプ:

このトピックでは、Java Database Connectivity (JDBC) ドライバーのタイプを定義します。ドライバーのタイプは、データベースに接続するために使用されるテクノロジーを分類するために使用されます。JDBC ドライバーのベンダーは、製品の動作方法を記述するためにこれらのタイプを使用します。一部のアプリケーションには、一部の JDBC ドライバーのタイプの方が、他のタイプよりも向いています。

### タイプ 1

タイプ 1 のドライバーは、「ブリッジ」ドライバーです。これらのドライバーは、データベースと通信するために Open Database Connectivity (ODBC) などの別のテクノロジーを使用します。これは利点となります。多くのリレーショナル・データベース管理システム (RDBMS) プラットフォーム用の ODBC ドライバーが存在するからです。JDBC ドライバーから ODBC 機能呼び出すために、Java Native Interface (JNI) が使用されます。

タイプ 1 のドライバーで JDBC を使用するには、その前にブリッジ・ドライバーがインストールおよび構成されている必要があります。これは、実動アプリケーションにとって重大な欠点となり得ます。アプレットはネイティブ・コードをロードできないので、タイプ 1 のドライバーをアプレットで使用することはできません。

### タイプ 2

タイプ 2 のドライバーは、データベース・システムと通信するためにネイティブ API を使用します。データベース操作を実行する API 関数を呼び出すために、Java ネイティブ・メソッドが使用されます。タイプ 2 のドライバーは、一般にタイプ 1 のドライバーよりも高速です。

タイプ 2 のドライバーが機能するためには、ネイティブ・バイナリー・コードがインストールおよび構成されている必要があります。タイプ 2 のドライバーも JNI を使用します。アプレットはネイティブ・コー

ドをロードできないので、タイプ 2 のドライバーをアプレットで使用することはできません。タイプ 2 の JDBC ドライバーでは、何らかのデータベース管理システム (DBMS) ネットワーキング・ソフトウェアがインストールされていなければならない場合があります。

Developer Kit for Java の JDBC ドライバーは、タイプ 2 の JDBC ドライバーです。

## タイプ 3

これらのドライバーは、サーバーと通信するためにネットワーク・プロトコルとミドルウェアを使用します。次いでサーバーは、プロトコルを DBMS 固有の DBMS 関数呼び出しに変換します。

タイプ 3 の JDBC ドライバーは、クライアント上にネイティブ・バイナリー・コードを必要としないので、最も柔軟な JDBC ソリューションです。タイプ 3 のドライバーは、クライアント側でのインストールを必要としません。

## タイプ 4

タイプ 4 のドライバーは、DBMS ベンダー・ネットワーク・プロトコルを実装するために Java を使用します。通常、プロトコルはメーカー独自仕様なので、一般に DBMS のベンダーはタイプ 4 の JDBC ドライバーの唯一の提供元です。

タイプ 4 のドライバーは、すべて Java ドライバーです。つまり、クライアント側でインストールや構成が行われれないという意味です。ただし、タイプ 4 のドライバーは、基礎プロトコルがセキュリティーやネットワーク接続性などの問題をうまく処理できない場合、一部のアプリケーションには向いていません。

IBM Toolbox for Java JDBC ドライバーはタイプ 4 の JDBC ドライバーであり、このことは、この API が純粋な Java ネットワーキング・プロトコル・ドライバーであることを示しています。

### JDBC の要件:

JDBC アプリケーションを作成および展開する前に、特定の JAR ファイルをクラスパスに組み込むことが必要になる場合があります。

## コア JDBC

ローカル・データベースへのコア Java Database Connectivity (JDBC) アクセスの場合、要件はありません。すべてのサポートが組み込まれ、プリインストールされ、構成済みとなっています。

## JDBC 2.0 オプション・パッケージ

JDBC 2.0 オプション・パッケージのクラスを使用する必要がある場合は、jdbc2\_0-stdext.jar ファイルをクラスパスに組み込む必要があります。この Java ARchive (JAR) ファイルには、JDBC 2.0 オプション・パッケージを使用するようにアプリケーションを作成するために必要な、すべての標準インターフェースが入っています。JAR ファイルを拡張クラスパスに追加するには、UserData 拡張ディレクトリーから JAR ファイルが置かれている場所へのシンボリック・リンクを作成します。これを行う必要があるのは一度だけです。アプリケーションは実行時に常に JDBC 2.0 オプション・パッケージの JAR ファイルを使用できます。オプション・パッケージを拡張クラスパスに追加するには、以下のコマンドを使用してください。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jdbc2_0-stdext.jar')
NEWLNK('/QIBM/UserData/Java400/ext/jdbc2_0-stdext.jar')
```

注: この要件は、J2SDK 1.3 にのみ適用されます。J2SDK 1.4 は、JDBC 3.0 をサポートするようになった最初のリリースであり、JDBC のすべて (つまり、コア JDBC とオプション・パッケージ) が基本

J2SDK ランタイム JAR ファイルに移動されています。このファイルは、常にプログラムによって検出されます。

## Java Transaction API

アプリケーションで Java Transaction API (JTA) を使用する必要がある場合は、jta-spec1\_0\_1.jar ファイルをクラスパスに組み込む必要があります。この JAR ファイルには、JTA を使用するようにアプリケーションを作成するために必要な、すべての標準インターフェースが入っています。JAR ファイルを拡張クラスパスに追加するには、UserData 拡張ディレクトリから JAR ファイルが置かれている場所へのシンボリック・リンクを作成します。これは一度限りの操作なので、いったん完了すれば、アプリケーションは実行時に常に JTA JAR ファイルを使用できます。JTA を拡張クラスパスに追加するには、以下のコマンドを使用してください。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jta-spec1_0_1.jar')
NEWLNK('/QIBM/UserData/Java400/ext/jta-spec1_0_1.jar')
```

## JDBC の準拠

ネイティブ JDBC ドライバーは、すべての関連した JDBC 仕様に準拠しています。JDBC ドライバーの準拠レベルは、i5/OS のリリースとは無関係ですが、使用する JDK のリリースに依存しています。各種 JDK のネイティブ JDBC ドライバーの準拠レベルは、以下のリストのとおりです。

| J2SDK のリリース         | JDBC ドライバーの準拠レベル  |
|---------------------|---|
| JDK 1.1             | この JDK は JDBC 1.0 に準拠しています。   |
| JDK 1.2             | この JDK は JDBC 2.0 に準拠していて、JDBC 2.1 オプション・パッケージをサポートしています。   |
| JDK 1.3             | この JDK は JDBC 2.0 に準拠していて、JDBC 2.1 オプション・パッケージをサポートしています (JDK 1.3 については、JDBC 関連の変更はありません)。        |
| JDK 1.4 および後続のバージョン | これらの JDK のバージョンは JDBC 3.0 に準拠していますが、JDBC オプション・パッケージは存在しなくなりました (このパッケージのサポートはコア JDK の一部となっています)。 |

## JDBC チュートリアル:

以下は、Java Database Connectivity (JDBC) プログラムを作成し、ネイティブ JDBC ドライバーが組み込まれた iSeries サーバー上で、そのプログラムを実行する方法についてのチュートリアルです。このチュートリアルは、プログラムで JDBC を実行するために必要な基本的なステップを示すように設計されています。

41 ページの『例: JDBC』は、テーブルを作成してデータを挿入します。プログラムは、そのデータをデータベースから取り出して画面に表示するための照会を処理します。

## サンプル・プログラムの実行

サンプル・プログラムを実行するには、以下のステップを実行してください。

1. プログラムをワークステーションにコピーする。
  - a. 41 ページの『例: JDBC』をコピーして、ワークステーション上のファイルにペーストする。
  - b. 提供されている共通クラスと同じ名前を使用し、.java 拡張子を付けてファイルを保管する。この場合、ローカル・ワークステーション上で、ファイルに BasicJDBC.java という名前を付ける必要があります。



2. ファイルをワークステーションから iSeries サーバーに転送する。コマンド・プロンプトから、以下のコマンドを入力します。

```
ftp <iSeries server name>
<Enter your user ID>
<Enter your password>
cd /home/cujo
put BasicJDBC.java
quit
```

これらのコマンドが機能するには、ファイルを書き込むディレクトリが存在していなければなりません。この例では、/home/cujo が書き込み先の場所ですが、任意の場所を使用できます。

**注:** 上記の FTP コマンドは、サーバーがどのようにセットアップされているかによって異なる場合がありますが、類似のコマンドとなります。ファイルを統合ファイル・システム内に転送する限り、どのような方法でファイルを iSeries サーバーに転送しても構いません。 VisualAge<sup>®</sup> for Java などのツールがあれば、この処理を完全に自動化できます。

3. Java コマンドの実行時にそれらの Java コマンドがファイルを発見できるように、そのファイルが置かれているディレクトリを示すクラスパスが設定されていることを確認する。 CL コマンド行から WRKENVVAR を使用して、ユーザー・プロファイルに設定されている環境変数を調べることができます。

- CLASSPATH という名前の環境変数を見つけたら、そこにリストされている一連のディレクトリの中に .java ファイルを置くようにするか、場所が指定されていない場合は追加する。
- CLASSPATH 環境変数がない場合は、追加する必要がある。これを行うには、以下のコマンドを使用します。

```
ADDENVVAR ENVVAR(CLASSPATH)
VALUE('/home/cujo:/QIBM/ProdData/Java400/jdk13/lib/tools.jar')
```

**注:** Java コードを CL コマンドからコンパイルするには、tools.jar ファイルを組み込む必要があります。この JAR ファイルには、javac コマンドが入っています。

4. Java ファイルをクラス・ファイルにコンパイルする。 CL コマンド行から、以下のコマンドを入力します。

```
java class(com.sun.tools.javac.Main) prop(BasicJDBC)
java BasicJDBC
```

以下のようにして、Java ファイルを QSH からコンパイルすることもできます。

```
cd /home/cujo
javac BasicJDBC.java
```

QSH は、自動的に tools.jar ファイルが検出されるようにします。その結果、このファイルをクラスパスに追加する必要はなくなります。現行ディレクトリもクラスパス内にあります。ディレクトリの変更 (cd) コマンドを発行することによっても、BasicJDBC.java ファイルを検出できます。

**注:** ファイルをワークステーション上でコンパイルし、FTP を使用してバイナリー・モードでクラス・ファイルを iSeries サーバーに送信することもできます。これは、どのプラットフォーム上でも実行できるという Java の機能の一例です。

CL コマンド行または QSH から以下のコマンドを使用して、プログラムを実行する。

```
java BasicJDBC
```

出力は以下のようになります。

|   |               |
|---|---------------|
| 1 | Frank Johnson |
| 2 | Neil Schwartz |
| 3 | Ben Rodman    |
| 4 | Dan Gloore    |

There were 4 rows returned.  
Output is complete.  
Java program completed.

Java および JDBC の詳細については、以下のリソースを調べてください。

- IBM Toolbox for Java JDBC ドライバーの外部 Web サイト
- Sun の JDBC ページ
- iSeries および iSeries ユーザーのための Java/JDBC フォーラム
- IBM JDBC の E メール・アドレス

#### 例: JDBC:

以下に、BasicJDBC プログラムの使用法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
//
// BasicJDBC example. This program uses the native JDBC driver for the
// Developer Kit for Java to build a simple table and process a query
// that displays the data in that table.
//
// Command syntax:
//   BasicJDBC
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////

// Include any Java classes that are to be used. In this application,
// many classes from the java.sql package are used and the

```



```

// java.util.Properties class is also used as part of obtaining
// a connection to the database.
import java.sql.*;
import java.util.Properties;

// Create a public class to encapsulate the program.
public class BasicJDBC {

    // The connection is a private variable of the object.
    private Connection connection = null;

    // Any class that is to be an 'entry point' for running
    // a program must have a main method. The main method
    // is where processing begins when the program is called.
    public static void main(java.lang.String[] args) {

        // Create an object of type BasicJDBC. This
        // is fundamental to object-oriented programming. Once
        // an object is created, call various methods on
        // that object to accomplish work.
        // In this case, calling the constructor for the object
        // creates a database connection that the other
        // methods use to do work against the database.
        BasicJDBC test = new BasicJDBC();

        // Call the rebuildTable method. This method ensures that
        // the table used in this program exists and looks
        // correct. The return value is a boolean for
        // whether or not rebuilding the table completed
        // successfully. If it did no, display a message
        // and exit the program.
        if (!test.rebuildTable()) {
            System.out.println("Failure occurred while setting up " +
                " for running the test.");
            System.out.println("Test will not continue.");
            System.exit(0);
        }

        // The run query method is called next. This method
        // processes an SQL select statement against the table that
        // was created in the rebuildTable method. The output of
        // that query is output to standard out for you to view.
        test.runQuery();

        // Finally, the cleanup method is called. This method
        // ensures that the database connection that the object has
        // been hanging on to is closed.
        test.cleanup();
    }

    /**
    This is the constructor for the basic JDBC test. It creates a database
    connection that is stored in an instance variable to be used in later
    method calls.
    */
    public BasicJDBC() {

        // One way to create a database connection is to pass a URL
        // and a java Properties object to the DriverManager. The following
        // code constructs a Properties object that has your user ID and
        // password. These pieces of information are used for connecting
        // to the database.
        Properties properties = new Properties ();
        properties.put("user", "cujo");
        properties.put("user", "newtiger");
    }
}

```

```

// Use a try/catch block to catch all exceptions that can come out of the
// following code.
try {
    // The DriverManager must be aware that there is a JDBC driver available
    // to handle a user connection request. The following line causes the
    // native JDBC driver to be loaded and registered with the DriverManager.
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

    // Create the database Connection object that this program uses in all
    // the other method calls that are made. The following code specifies
    // that a connection is to be established to the local database and that
    // that connection should conform to the properties that were set up
    // previously (that is, it should use the user ID and password specified).
    connection = DriverManager.getConnection("jdbc:db2:*local", properties);

} catch (Exception e) {
    // If any of the lines in the try/catch block fail, control transfers to
    // the following line of code. A robust application tries to handle the
    // problem or provide more details to you. In this program, the error
    // message from the exception is displayed and the application allows
    // the program to return.
    System.out.println("Caught exception: " + e.getMessage());
}
}

/**
Ensures that the qgpl.basicjdbc table looks you want it to at the start of
the test.

@returns boolean    Returns true if the table was rebuild successfully;
                    returns false if any failure occurred.
**/
public boolean rebuildTable() {
    // Wrap all the functionality in a try/catch block so an attempt is
    // made to handle any errors that may happen within this method.
    try {

        // Statement objects are used to process SQL statements against the
        // database. The Connection object is used to create a Statement
        // object.
        Statement s = connection.createStatement();

        try {
            // Build the test table from scratch. Process an update statement
            // that attempts to delete the table if it currently exists.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // Do NOT perform anything if an exception occurred. Assume
            // that the problem is that the table that was dropped does not
            // exist and that it can be created next.
        }

        // Use the statement object to create our table.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Use the statement object to populate our table with some data.
        s.executeUpdate("insert into qgpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qgpl.basicjdbc values(2, 'Neil Schwartz')");
        s.executeUpdate("insert into qgpl.basicjdbc values(3, 'Ben Rodman')");
        s.executeUpdate("insert into qgpl.basicjdbc values(4, 'Dan Gloore')");

        // Close the SQL statement to tell the database that it is no longer
        // needed.
        s.close();

        // If the entire method processed successfully, return true. At this point,

```

```

        // the table has been created or refreshed correctly.
        return true;
    } catch (SQLException sqle) {
        // If any of our SQL statements failed (other than the drop of the table
        // that was handled in the inner try/catch block), the error message is
        // displayed and false is returned to the caller, indicating that the table
        // may not be complete.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}

/**
Runs a query against the demonstration table and the results are displayed to
standard out.
**/
public void runQuery() {
    // Wrap all the functionality in a try/catch block so an attempts is
    // made to handle any errors that might happen within this
    // method.
    try {
        // Create a Statement object.
        Statement s = connection.createStatement();

        // Use the statement object to run an SQL query. Queries return
        // ResultSet objects that are used to look at the data the query
        // provides.
        ResultSet rs = s.executeQuery("select * from qqpl.basicjdbc");

        // Display the top of our 'table' and initialize the counter for the
        // number of rows returned.
        System.out.println("-----");
        int i = 0;

        // The ResultSet next method is used to process the rows of a
        // ResultSet. The next method must be called once before the
        // first data is available for viewing. As long as next returns
        // true, there is another row of data that can be used.
        while (rs.next()) {

            // Obtain both columns in the table for each row and write a row to
            // our on-screen table with the data. Then, increment the count
            // of rows that have been processed.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Place a border at the bottom on the table and display the number of rows
        // as output.
        System.out.println("-----");
        System.out.println("There were " + i + " rows returned.");
        System.out.println("Output is complete.");
    } catch (SQLException e) {
        // Display more information about any SQL exceptions that are
        // generated as output.
        System.out.println("SQLException exception: ");
        System.out.println("Message:....." + e.getMessage());
        System.out.println("SQLState:...." + e.getSQLState());
        System.out.println("Vendor Code:.. " + e.getErrorCode());
        e.printStackTrace();
    }
}

```

```

/**
The following method ensures that any JDBC resources that are still
allocated are freed.
**/
public void cleanup() {
    try {
        if (connection != null)
            connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## 例での JNDI の使用:

DataSource は JNDI (Java Naming and Directory Interface) と協調して動作します。 JDBC がデータベースの抽象化層であるのと同じように、JNDI はディレクトリー・サービスの Java 抽象化層です。

ほとんどの場合、JNDI は LDAP (Lightweight Directory Access Protocol) と共に使用されますが、CORBA Object Services (COS)、Java Remote Method Invocation (RMI) レジストリー、またはベースのファイル・システムと共に使用されることもあります。この多様化された使用法は、一般的な JNDI 要求を、特定のディレクトリー・サービスの要求に転換する、各種のディレクトリー・サービス・プロバイダーの方式によって実現されます。Java 2 SDK v 1.3 には、LDAP サービス・プロバイダー、COS 命名サービス・プロバイダー、および RMI レジストリー・サービス・プロバイダーという 3 つのサービス・プロバイダーが含まれています。

**注:** RMI を使用することは、複雑な作業になる可能性があるということを銘記しておいてください。RMI をソリューションとして選ぶ前に、これを選ぶと及ぶ影響について必ず理解しておいてください。RMI の評価を開始するのに適した場所は、Java Remote Method Invocation (RMI) です。

この DataSource のサンプルは、JNDI ファイル・システム・サービス・プロバイダーを使用するように設計されています。提供された例を実行する場合は、JNDI サービス・プロバイダーが適切な場所になければなりません。

ファイル・システム・サービス・プロバイダーを使用する環境をセットアップするには、以下の手順に従ってください。

1. Sun Microsystems の JNDI サイトから、ファイル・システム JNDI サポートをダウンロードする。
2. (FTP または他のメカニズムを使って、) fscontext.jar および providerutil.jar をシステムに転送し、/QIBM/UserData/Java400/ext に配置する。ここは拡張機能ディレクトリーで、ここに配置された JAR ファイルはアプリケーションの実行時に自動的に検索されます (クラスパスに追加する必要はありません)。

JNDI のサービス・プロバイダーがサポートされたら、アプリケーションのコンテキスト情報をセットアップする必要があります。これは、SystemDefault.properties ファイル内に必要な情報を書き込むことによって行えます。デフォルト・プロパティーを指定できる場所はシステム上にいくつかありますが、最善の方法は、ユーザーのホーム・ディレクトリー (/home/) に SystemDefault.properties という名前のテキスト・ファイルを作成することです。

ファイルを作成するには、以下の行を使用するか、既存のファイルに以下の行を追加します。

```
# Needed env settings for JNDI.  
java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory  
java.naming.provider.url=file:/DataSources/jdbc
```

これらの行は、JNDI 要求をファイル・システム・サービス・プロバイダーが処理し、/DataSource/jdbc が JNDI を使用するタスクのルートであることを指定しています。この場所を変更することもできますが、ユーザーが指定したディレクトリは必ず存在していなければなりません。ユーザーが指定した場所に、例で使用する DataSource がバインドおよび展開されます。

## Connections

Connection オブジェクトは、Java Database Connectivity (JDBC) 内のデータ・ソースへの接続を表しています。SQL ステートメントを処理するために作成された Statement オブジェクトは、Connection オブジェクトを介してデータベースに接続します。アプリケーション・プログラムは、一度に複数の接続を持つことができます。これらの Connection オブジェクトは、すべて同じデータベースに接続することも、別々のデータベースに接続することもできます。

JDBC 内で接続を取得するには、2 つの方法があります。

- DriverManager クラスを通して取得する。
- DataSource を使用して取得する。

アプリケーションの移植性と保守容易性を高めることができるため、接続を取得するためには DataSource を使用するほうが便利です。これはまた、アプリケーションが接続およびステートメント・プーリング、および分散トランザクションを容易に使用することができるようにします。

接続の取得について詳しくは、以下のセクションを参照してください。

**DriverManager** DriverManager は、アプリケーションが使用できるよう、使用可能な JDBC ドライバーのセットを管理する静的クラスです。

**接続プロパティ** 以下の表は、JDBC ドライバーの接続プロパティとその値、およびその説明を示しています。

**DataSource を UDBDataSource と共に使用する DataSource** は、特定のプロパティを持つようセットアップし、JNDI (Java Naming and Directory Interface) を使っていくつかのディレクトリ・サービスにバインドすることにより、UDBDataSource クラスと共に展開することができます。

**DataSource プロパティ** 以下の表は、有効な DataSource の接続プロパティとその値、およびその説明を示しています。

**その他の DataSource インプリメンテーション** ネイティブ JDBC ドライバーと共に、DataSource インターフェースのその他のインプリメンテーションが提供されています。これらは、UDBDataSource および関連した機能が採用されるまでのつなぎとして提供されています。

一度接続が取得されると、次のような JDBC タスクを完了するために使用することができます。

- さまざまなタイプの Statement オブジェクトを作成し、データベースを操作する。
- データベースに対するトランザクションを制御する。
- データベースに関するメタデータを取得する。

### DriverManager:

DriverManager は、Java 2 Software Development Kit (J2SDK) にある静的クラスです。 DriverManager は、アプリケーションで使用可能な複数の JDBC ドライバーのセットを使用できるように管理するものです。

アプリケーションは、必要があれば、複数の JDBC ドライバーを同時に利用することができます。各アプリケーションは Uniform Resource Locator (URL) を使用して、JDBC ドライバーを指定します。特定の JDBC ドライバーの URL を DriverManager に渡すことにより、アプリケーションは DriverManager に対して、どの種類の JDBC 接続をアプリケーションに戻すべきかを通知します。

これが完了するまでは、DriverManager が接続を提供することのできる、利用可能な JDBC ドライバーを認知していなければなりません。 Class.forName メソッドへの呼び出しを行うことによって、メソッドの中に渡されたストリング名に基づいて実行中の Java 仮想マシン (JVM) にクラスがロードされます。ネイティブ JDBC ドライバーをロードするために class.forName メソッドを使用する例を以下に示します。

#### 例: ネイティブ JDBC ドライバーをロードする

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Load the native JDBC driver into the DriverManager to make it
// available for getConnection requests.
```

```
Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
```

JDBC ドライバーは、ドライバーのインプリメンテーション・クラスがロードされると、自動的に自分自身について DriverManager に通知するように設計されています。前述のコードの行が一度処理されると、そのネイティブ JDBC ドライバーは DriverManager と共に動作できるようになります。以下のコードは、ネイティブ JDBC URL を使って、Connection オブジェクトを要求しています。

#### 例: Connection オブジェクトを要求する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Get a connection that uses the native JDBC driver.
```

```
Connection c = DriverManager.getConnection("jdbc:db2:*local");
```

最も単純な JDBC URL の形式は、コロンで区切られた 3 つの値のリストです。リストの最初の値はプロトコルを示しており、JDBC URL では常に jdbc になります。2 番目の値はサブプロトコルで、ネイティブ JDBC ドライバーを指定するために db2 または db2iSeries を使用しています。3 番目の値は、指定したシステムへの接続を確立するためのシステム名です。ローカル・データベースに接続するために、2 つの特殊値があります。それは、\*LOCAL と localhost です (どちらも大文字小文字を区別しません)。特定のシステム名は次のようにも指定できます。

```
Connection c =
    DriverManager.getConnection("jdbc:db2:rchasmop");
```

これは、rchasmop システムへの接続を作成します。システムがリモート・システムへの接続を試行する場合 (たとえば、分散リレーショナル・データベース体系) は、リレーショナル・データベース・ディレクトリーにあるシステム名を使用する必要があります。

注: 指定されない場合、サインインに使用されているユーザー ID とパスワードが、データベースへの接続の確立にも使用されます。

注: IBM DB2® JDBC Universal ドライバーも db2 サブプロトコルを使用します。ネイティブ JDBC ドライバーが URL を処理するようにするためには、アプリケーションで jdbc:db2:xxxx の URL ではなく



jdbc:db2iSeries:xxxx の URL を使用する必要があります。アプリケーションで、db2 サブプロトコルを含む URL をネイティブ・ドライバーが受け入れないようにしたい場合は、そのアプリケーションで com.ibm.db2.jdbc.app.DB2Driver ではなく、com.ibm.db2.jdbc.app.DB2iSeriesDriver クラスをロードする必要があります。このクラスをロードすると、ネイティブ・ドライバーは db2 サブプロトコルを含む URL を処理しなくなります。

## プロパティー

DriverManager.getConnection メソッドは DriverManager 上で Connection オブジェクトを取得する唯一のメソッドで、前述の単一のストリング URL を取ります。DriverManager.getConnection メソッドの別のバージョンでは、ユーザー ID とパスワードを取ります。このバージョンの例は次のとおりです。

**例:** ユーザー ID とパスワードを取る DriverManager.getConnection メソッド

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Get a connection that uses the native JDBC driver.  
  
Connection c = DriverManager.getConnection("jdbc:db2:*local", "cujo", "newtiger");
```

このコードは、誰がこのアプリケーションを実行しているかにかかわらず、ユーザー cujo、パスワード newtiger としてローカル・データベースに接続することを想定しています。DriverManager.getConnection メソッドの別のバージョンでは、さらにカスタマイズを行うため、java.util.Properties オブジェクトを取ります。以下に、この例を示します。

**例:** java.util.Properties オブジェクトを取る DriverManager.getConnection メソッド

```
// Get a connection that uses the native JDBC driver.  
  
Properties prop = new java.util.Properties();  
prop.put("user", "cujo");  
prop.put("password", "newtiger");  
Connection c = DriverManager.getConnection("jdbc:db2:*local", prop);
```

このコードは、前述のバージョンと機能的には同等ですが、ユーザー ID とパスワードをパラメーターとして渡しています。

ネイティブ JDBC ドライバーの接続プロパティーの完全なリストは、接続プロパティーを参照してください。

## URL プロパティー

プロパティーを指定する別の方法は、それらのプロパティーを URL オブジェクトのリストに格納することです。リスト内のそれぞれのプロパティーはセミコロンで区切られ、リストはプロパティー名 = プロパティー値という形式になっている必要があります。これは単なるショートカットであり、処理される方法には違いはありません。次の例のように記述されます。

**例:** URL プロパティーを指定する

```
// Get a connection that uses the native JDBC driver.  
  
Connection c = DriverManager.getConnection("jdbc:db2:*local;user=cujo;password=newtiger");
```

このコードも、前述の例と機能的には同等です。

プロパティー値が properties オブジェクトと URL オブジェクトの両方で指定された場合は、URL バージョンの指定が properties オブジェクトよりも優先されます。以下に、この例を示します。



## 例: URL プロパティー

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Get a connection that uses the native JDBC driver.
Properties prop = new java.util.Properties();
prop.put("user", "someone");
prop.put("password", "something");
Connection c = DriverManager.getConnection("jdbc:db2:*local;user=cujo;password=newtiger",
prop);
```

この例では、Properties オブジェクトで指定されたユーザー ID とパスワードではなく、URL スtring で指定されたユーザー ID とパスワードが使用されます。結果として、前述のコードと機能的に完全に同等になります。

詳細については、下記の例を参照してください。

- ネイティブ JDBC と IBM Toolbox for Java JDBC を同時に使用する
- Access プロパティー
- 無効なユーザー ID とパスワード

## 例: ネイティブ JDBC と IBM Toolbox for Java JDBC を同時に使用する:

以下に、プログラム中でネイティブ JDBC 接続と IBM Toolbox for Java JDBC 接続を使用する方法を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
//
// GetConnections example.
//
// This program demonstrates being able to use both JDBC drivers at
// once in a program. Two Connection objects are created in this
// program. One is a native JDBC connection and one is a IBM Toolbox for Java
// JDBC connection.
//
// This technique is convenient because it allows you to use different
// JDBC drivers for different tasks concurrently. For example, the
// IBM Toolbox for Java JDBC driver is ideal for connecting to remote iSeries servers
// and the native JDBC driver is faster for local connections.
// You can use the strengths of each driver concurrently in your
// application by writing code similar to this example.
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
```

```

// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
///////////////////////////////////////////////////////////////////
import java.sql.*;
import java.util.*;

public class GetConnections {

    public static void main(java.lang.String[] args)
    {
        // Verify input.
        if (args.length != 2) {
            System.out.println("Usage (CL command line): java GetConnections PARM(<user> <password>");
            System.out.println(" where <user> is a valid iSeries user ID");
            System.out.println(" and <password> is the password for that user ID");
            System.exit(0);
        }

        // Register both drivers.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            Class.forName("com.ibm.as400.access.AS400JDBCdriver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: One of the JDBC drivers did not load.");
            System.exit(0);
        }

        try {
            // Obtain a connection with each driver.
            Connection conn1 = DriverManager.getConnection("jdbc:db2://localhost", args[0], args[1]);
            Connection conn2 = DriverManager.getConnection("jdbc:as400://localhost", args[0], args[1]);

            // Verify that they are different.
            if (conn1 instanceof com.ibm.db2.jdbc.app.DB2Connection)
                System.out.println("conn1 is running under the native JDBC driver.");
            else
                System.out.println("There is something wrong with conn1.");

            if (conn2 instanceof com.ibm.as400.access.AS400JDBCCConnection)
                System.out.println("conn2 is running under the IBM Toolbox for Java JDBC driver.");
            else
                System.out.println("There is something wrong with conn2.");

            conn1.close();
            conn2.close();
        } catch (SQLException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }
}

```

## リンク集

コードのサンプルに関する特記事項

### 例: Access プロパティ:

以下に、Access プロパティの使用法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

// Note: This program assumes directory cujosql exists.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class AccessPropertyTest {
    public String url = "jdbc:db2:*local";
    public Connection connection = null;

    public static void main(java.lang.String[] args)
    throws Exception
    {
        AccessPropertyTest test = new AccessPropertyTest();

        test.setup();

        test.run();
        test.cleanup();
    }

    /**
    Set up the DataSource used in the testing.
    **/
    public void setup()
    throws Exception
    {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        connection = DriverManager.getConnection(url);
        Statement s = connection.createStatement();
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.TEMP");
        } catch (SQLException e) { // Ignore it - it doesn't exist
        }

        try {
            String sql = "CREATE PROCEDURE CUJOSQL.TEMP "
                + " LANGUAGE SQL SPECIFIC CUJOSQL.TEMP "
                + " MYPROC: BEGIN"
                + " RETURN 11;"
                + " END MYPROC";
            s.executeUpdate(sql);
        } catch (SQLException e) {
            // Ignore it - it exists.
        }
        s.executeUpdate("create table cujosql.temp (col1 char(10))");
        s.executeUpdate("insert into cujosql.temp values ('compare')");
        s.close();
    }

    public void resetConnection(String property)
    throws SQLException
    {
        if (connection != null)
            connection.close();

        connection = DriverManager.getConnection(url + ";access=" + property);
    }

    public boolean canQuery() {
        Statement s = null;
        try {
            s = connection.createStatement();
            ResultSet rs = s.executeQuery("SELECT * FROM cujosql.temp");
        }
    }
}

```

```

        if (rs == null)
            return false;

        rs.next();

        if (rs.getString(1).equals("compare  "))
            return true;

        return false;
    } catch (SQLException e) {
        // System.out.println("Exception: SQLState(" +
        //                      e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}

```

```

public boolean canUpdate() {
    Statement s = null;
    try {
        s = connection.createStatement();
        int count = s.executeUpdate("INSERT INTO CUJOSQL.TEMP VALUES('x')");
        if (count != 1)
            return false;

        return true;
    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}

```

```

public boolean canCall() {
    CallableStatement s = null;
    try {
        s = connection.prepareCall("? = CALL CUJOSQL.TEMP()");
        s.registerOutParameter(1, Types.INTEGER);
        s.execute();
        if (s.getInt(1) != 11)
            return false;

        return true;
    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    }
}

```

```

    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}

public void run()
throws SQLException
{
    System.out.println("Set the connection access property to read only");
    resetConnection("read only");

    System.out.println("Can run queries -->" + canQuery());
    System.out.println("Can run updates -->" + canUpdate());
    System.out.println("Can run sp calls -->" + canCall());

    System.out.println("Set the connection access property to read call");
    resetConnection("read call");

    System.out.println("Can run queries -->" + canQuery());
    System.out.println("Can run updates -->" + canUpdate());
    System.out.println("Can run sp calls -->" + canCall());

    System.out.println("Set the connection access property to all");
    resetConnection("all");

    System.out.println("Can run queries -->" + canQuery());
    System.out.println("Can run updates -->" + canUpdate());
    System.out.println("Can run sp calls -->" + canCall());

}

public void cleanup() {
    try {
        connection.close();
    } catch (Exception e) {
        // Ignore it.
    }
}
}

```

**例: 無効なユーザー ID とパスワード:**

以下は、SQL 命名モードでの Connection プロパティの使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
//
// InvalidConnect example.
//
// This program uses the Connection property in SQL naming mode.
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to

```

```

// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////
import java.sql.*;
import java.util.*;

public class InvalidConnect {

    public static void main(java.lang.String[] args)
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: JDBC driver did not load.");
            System.exit(0);
        }

        // Attempt to obtain a connection without specifying any user or
        // password. The attempt works and the connection uses the
        // same user profile under which the job is running.
        try {
            Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
            c1.close();
        } catch (SQLException e) {
            System.out.println("This test should not get into this exception path.");
            e.printStackTrace();
            System.exit(1);
        }

        try {
            Connection c2 = DriverManager.getConnection("jdbc:db2:*local",
                "notvalid", "notvalid");
        } catch (SQLException e) {
            System.out.println("This is an expected error.");
            System.out.println("Message is " + e.getMessage());
            System.out.println("SQLSTATE is " + e.getSQLState());
        }

    }
}

```

#### 接続プロパティ:

以下の表は、JDBC ドライバーの接続プロパティとその値、およびその説明を示しています。

| プロパティ       | 値                       | 意味  |
|-------------|-------------------------|---|
| access      | all、read call、read only | この値を使用すると、特定の接続で実行できる操作のタイプを制限できません。デフォルト値は all であり、これは基本的に、その接続が JDBC API への完全アクセスを持つことを意味します。read call 値は、照会の実行とストアード・プロシージャの呼び出しだけをその接続に許可します。SQL ステートメントを介してデータベースを更新しようとしても、その処理は停止します。read only 値を使用すれば、接続を照会だけに制限できます。ストアード・プロシージャ呼び出しと更新ステートメントは停止しません。 |
| 自動コミット      | true、false              | この値は接続の自動コミット設定の設定に使用されます。トランザクション分離プロパティが none 以外の値に設定されていない場合は、デフォルト値は true です。設定されている場合は、デフォルト値は false になります。  |
| batch style | 2.0、2.1                 | JDBC 2.1 仕様では、バッチ処理での更新で例外が発生した場合の処理として、2 番目のメソッドが定義されています。ドライバーは、どちらかの仕様に準拠できます。デフォルトでは、JDBC 2.0 仕様で定義された方法で処理されます。  |



| プロパティ            | 値                        | 意味   |
|------------------|--------------------------|--|
| block size       | 0、8、16、32、64、128、256、512 | <p>これは、ResultSet として一度に取り出される行数です。順方向に限られた通常の ResultSet の処理では、このサイズのブロックが取得されます。その後、ユーザーのアプリケーションが各行を処理するので、データベースはアクセスされません。ブロックの終わりに達したときに初めて、データベースは別のデータ・ブロックを要求します。</p> <p>この値は、blocking enabled プロパティが true に設定されているときにのみ使用されます。</p> <p>block size プロパティを 0 に設定すると、blocking enabled プロパティを "false" に設定した場合と同じ効果が得られます。</p> <p>デフォルトでは、32 のブロック・サイズでブロック化が行われます。これは、非常に独断的な決定であり、デフォルトは将来に変更される可能性があります。</p> <p>スクロール可能な ResultSet でブロック化は行われません。</p> |
| blocking enabled | true、false               | <p>この値は、接続で ResultSet 行を検索するときにブロック化を行うかどうかを決定します。ブロック化により、ResultSet の処理のパフォーマンスは大きく改善される可能性があります。</p> <p>このプロパティは、デフォルトでは true に設定されています。</p>   |

| プロパティ             | 値                                  | 意味   |
|-------------------|------------------------------------|--|
| cursor hold       | true、false                         | <p>この値は、トランザクションがコミットされた後もResultSet をオープンにしたままにするかどうかを設定します。この値を true にすると、コミットが呼び出された後でも、アプリケーションはオープンしている ResultSet にアクセスすることができます。この値を false にすると、その接続の中でオープンしているカーソルはコミット時にすべてクローズされます。</p> <p>このプロパティは、デフォルトでは true に設定されています。</p> <p>この値プロパティは、その接続で作成されたすべての ResultSet のデフォルト値となります。JDBC 3.0 では、カーソルの保持機能がサポートされました。このデフォルトは、アプリケーションによって別の保持機能が指定されると、置き換えられます。</p> <p>以前のバージョンから JDBC 3.0 にマイグレーションする場合、カーソルの保持機能のサポートが JDBC 3.0 までは追加されていなかったことに留意してください。以前のバージョンでは、デフォルト値「true」が接続時に送信されていたものの、JVM では認識されていませんでした。したがって、カーソル保持プロパティが JDBC 3.0 までのデータベース機能に影響を与えることはありません。</p> |
| data truncation   | true、false                         | <p>この値は、文字データが切り捨てられた場合に警告および例外を生成するか (true)、黙って切り捨てるか (false) を設定します。デフォルトは true で、文字フィールドのデータ切り捨てが有効です。</p>  |
| date format       | julian、mdy、dmy、ymd、usa、iso、eur、jis | <p>このプロパティは、日付のフォーマット方法を変更することができます。</p>   |
| date separator    | /(スラッシュ)、-(ダッシュ)、.(ピリオド)、,(コンマ)、b  | <p>このプロパティを使って、日付区切り文字を変更することができます。このプロパティは、(システム規則に従って) 一部の dateFormat 値との組み合わせでのみ有効です。</p>   |
| decimal separator | .(ピリオド)、,(コンマ)                     | <p>このプロパティを使って、小数点を変更できます。</p>   |

| プロパティ                | 値          | 意味   |
|----------------------|------------|--|
| do escape processing | true、false | <p>このプロパティは、接続でステートメントがエスケープ処理を実行するかどうかを示すフラグを設定します。エスケープ処理の使用は、SQL ステートメントをすべてのプラットフォームで汎用および類似のものとなるようにコード化する 1 つの方法です。データベースはエスケープ文節を読み取ると、ユーザーのシステム固有のバージョンに置き換えます。</p> <p>これは、システムに余分な作業を強制する点を除けば、優れた機能です。既に有効な iSeries SQL 構文が含まれている SQL ステートメントだけを使用することが分かっているなら、パフォーマンスの向上のために、この値を false に設定することを勧めます。</p> <p>JDBC 仕様に準拠するため、このプロパティのデフォルト値は true になっています (エスケープ処理はデフォルトでアクティブになっています)。</p> <p>この値は、JDBC 仕様の欠点を補うために追加されています。Statement クラスでは、エスケープ処理は off に設定することしかできません。単純なステートメントを処理する場合は、これでもうまくいきます。ステートメントを作成し、エスケープ処理を off にしてから、ステートメントの処理を開始することができます。ただし、PreparedStatement や呼び出し可能ステートメントの場合、この方式ではうまくいきません。PreparedStatement や呼び出し可能ステートメントの構成時に指定した SQL ステートメントは、後から変更されることはありません。そのため、ステートメントを準備し、その後からエスケープ処理を変更しても、効果はありません。この接続プロパティを設定することは、余分なオーバーヘッドを回避するための 1 つの手段となります。</p> |

| プロパティ     | 値  | 意味   |
|-----------|--|--|
| errors    | basic、full   | このプロパティは、全システムの 2 次レベル・エラー・テキストを <code>SQLException</code> オブジェクト・メッセージに戻すかどうかを指定します。デフォルトは <code>basic</code> になっており、標準メッセージ・テキストのみを戻します。   |
| libraries | スペースで区切られたライブラリーのリスト。(ライブラリーのリストは、コロンまたはコンマで区切ることもできます。) | <p>このプロパティを使うと、ライブラリーのリストをサーバー・ジョブのライブラリー・リスト、または指定されたデフォルト・ライブラリーに設定することができます。</p> <p>このプロパティがどのように動作するかは、<code>naming</code> プロパティによって影響されます。デフォルトでは、<code>naming</code> は <code>"sql"</code> に設定されており、<code>JDBC</code> は <code>ODBC</code> と同様の動作をします。ライブラリー・リストは、どのように接続処理が行われるかには影響しません。これは、すべての非修飾テーブルのデフォルト・ライブラリーになります。デフォルトでは、接続しているユーザー・プロファイルと同じ名前を持つライブラリーです。 <code>libraries</code> プロパティが指定されると、リストで最初に指定されているライブラリーがデフォルト・ライブラリーとなります。デフォルト・ライブラリーが接続 URL (<code>jdbc:db2:*local/mylibrary</code> 内) で指定されている場合は、このプロパティの設定よりも優先されます。</p> <p><code>naming</code> が <code>system</code> に設定されている場合は、このプロパティで指定したそれぞれのライブラリーがユーザーのライブラリー・リストに追加され、非修飾テーブル参照を解決するために検索されます。</p> |

| プロパティ                | 値              | 意味  |
|----------------------|----------------|---|
| lob threshold        | 500000 未満の任意の値 | <p>このプロパティは、ドライバーに対して、lob 列がしきい値よりも小さかった場合、lob 列のロケータの代わりに ResultSet 記憶域に実際の値を格納することを指定します。このプロパティは、列のサイズに対してのもので、それ自身の lob データ・サイズに対するものではありません。たとえば、各 lob 列のサイズが最大 1 MB として定義されていても、各列の値が 500 KB 以下であれば、引き続きロケータが使用されます。</p> <p>このサイズ制限は、常にデータ・ブロックが 16 MB の最大割り振りサイズよりも大きくなる危険がなく、ブロックをフェッチできるように指定されることに注意してください。大きな ResultSet では簡単にこの制限を超えてしまい、その結果、フェッチが失敗します。データ・ブロックと相互に作用するため、ブロック・サイズ・プロパティとこのプロパティの設定は慎重に行ってください。</p> <p>デフォルトでは 0 に設定され、lob データには常にロケータが使用されます。</p> |
| maximum precision    | 31、63          | この値は、結果のデータ・タイプで戻される最大精度 (長さ) を指定します。デフォルト値は 31 です。   |
| maximum scale        | 0 から 63        | この値は、結果のデータ・タイプで戻される最大位取り (小数点の右側の小数点以下の桁数) を指定します。値の範囲は 0 から最大精度までです。デフォルト値は 31 です。  |
| minimum divide scale | 0 から 9         | この値は、中間と結果の両方のデータ・タイプで戻される最小分割位取り (小数点の右側の小数点以下の桁数) を指定します。この値は、最大位取りを超えない 0 から 9 の範囲の値です。0 が指定されている場合は、最小分割位取りは使用されません。デフォルト値は 0 です。   |

| プロパティ    | 値          | 意味   |
|----------|------------|--|
| naming   | sql、system | <p>このプロパティを使用すると、従来の iSeries 命名構文か、標準の SQL 命名構文のいずれかを使用できます。system 命名は、/ (スラッシュ) 文字を使用してコレクション値とテーブル値を区切ることを意味し、SQL 命名は、. (ピリオド) 文字を使用してそれらの値を区切ることを意味します。</p> <p>この値には、デフォルトのライブラリーによってさまざまな設定があります。詳しくは、上記の libraries プロパティを参照してください。</p> <p>デフォルトでは SQL 命名が使用されます。</p>                                      |
| password | 任意の値       | <p>このプロパティでは、接続用のパスワードを指定することができます。このプロパティは、user プロパティが指定されていないと、正しく動作しません。これらのプロパティを使うと、iSeries ジョブを実行しているユーザーではないユーザーとして、データベースに接続することができるようになります。</p> <p>user および password プロパティを指定すると、シグニチャー getConnection(String url, String userId, String password) の接続メソッドを使用した場合と同様の効果が得られます。</p>                     |
| prefetch | true、false | <p>このプロパティは、ドライバーが処理の直後に ResultSet の最初のデータをフェッチするのか、データが要求されてからフェッチするのかを指定します。デフォルトは true で、データはプリフェッチされます。</p> <p>ネイティブ JDBC ドライバーを使用したアプリケーションの場合、まれにこのことが問題になることがあります。このプロパティは本来、Java ストアード・プロシージャおよびユーザー定義関数での内部使用のために存在しています。それらにおいては、要求されるまではデータベース・エンジンがユーザーのために ResultSet からデータをフェッチしないことが重要となります。</p> |

| プロパティ                 | 値   | 意味  |
|-----------------------|---|---|
| reuse objects         | true, false   | このプロパティは、一度クローズされた、ある種類のオブジェクトをドライバーが再利用するかどうかを指定します。これはパフォーマンスを向上させます。デフォルトは true です。  |
| server trace          | 整数として表記されるストリング   | このプロパティで、JDBC サーバー・ジョブをトレースすることが可能になります。サーバー・トレースが使用可能な場合、クライアントがサーバーに接続したときにトレースが開始され、接続が切断されたときにトレースが終了します。<br><br>トレース・データは、サーバー上のスプール・ファイル内に収集されます。複数レベルのサーバー・トレースは、定数を加算することと、set メソッドでその合計を渡すことの組み合わせによってオンになります。<br><b>注:</b> このプロパティは、一般にはサポート担当者によって使用されるもので、この値に関するこれ以上の説明はありません。 |
| time format           | hms, usa, iso, eur, jis   | このプロパティを使って、時刻のフォーマット方法を変更することができます。  |
| time separator        | :(コロン)、.(ピリオド)、,(コンマ)、b   | このプロパティを使って、時刻区切り文字を変更することができます。このプロパティは、(システム規則に従って) 一部の timeFormat 値との組み合わせでのみ有効です。   |
| trace                 | true, false   | このプロパティによって、接続のトレースをオンにすることができます。これは、単純なデバッグ支援機能として使用することができます。<br><br>デフォルト値は false で、トレースを使用しません。   |
| transaction isolation | none, read committed, read uncommitted, repeatable read, serializable | このプロパティを使って、接続のトランザクション分離レベルを設定することができます。このプロパティで特定のレベルを設定することは、Connection インターフェースの setTransactionIsolation メソッドでレベルを指定するのと同じことです。<br><br>JDBC のデフォルトは自動コミット・モードなので、このプロパティのデフォルト値は none です。   |



| プロパティ            | 値                 | 意味  |
|------------------|-------------------|---|
| translate binary | true, false       | <p>このプロパティを使用すれば、JDBC ドライバーで binary データ値や varbinary データ値を強制的に char データ値や varchar データ値と同様に扱うことができます。</p> <p>このプロパティのデフォルトは false で、バイナリー・データは文字データと同様には扱われません。</p>   |
| translate hex    | binary, character | <p>この値を使って、SQL 式の 16 進数を使用するデータ・タイプを選択することができます。binary が設定されている場合、それは 16 進数が BINARY データ・タイプを使用することを意味します。character が設定されている場合、それは 16 進数が CHARACTER FOR BIT DATA データ・タイプを使用することを意味します。デフォルト設定は character です。</p>   |
| use block insert | true, false       | <p>このプロパティは、データベースにデータのブロックを挿入するために、ネイティブ JDBC ドライバーがブロック挿入モードになることを可能にします。これはバッチ更新の最適化バージョンです。この最適化モードは、あるシステム制約への違反、またデータ挿入の障害およびデータの破壊が発生する恐れのないアプリケーション内でのみ使用できます。</p> <p>このプロパティをオンにしたアプリケーションは、バッチ更新を行おうとするときにはローカル・システムにのみ接続します。ブロック挿入は DRDA 上では管理できないため、DRDA を使ってリモート接続を確立します。</p> <p>また、アプリケーションでは SQL 挿入ステートメントで PreparedStatements が使われており、すべての挿入値パラメーターが VALUES 文節によって指定されている必要があります。値リストで定数を使うことは許可されていません。これは、システムのブロック挿入エンジンの要件です。</p> <p>デフォルトは false です。</p> |

| プロパティ | 値    | 意味  |
|-------|------|---|
| user  | 任意の値 | <p>このプロパティでは、接続用のユーザー ID を指定することができます。このプロパティは、password プロパティも指定されていないと機能しません。これらのプロパティを使うと、iSeries ジョブを実行しているユーザーではないユーザーとして、データベースに接続することができるようになります。</p> <p>user および password プロパティを指定すると、シグニチャー getConnection(String url, String userId, String password) の接続メソッドを使用した場合と同様の効果が得られます。</p> |

**例: UDBDataSource を作成して JNDI でバインドする:**

次に、UDBDataSource を作成し、それを JNDI でバインドする方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS", ds);
    }
}
```

**例: UDBDataSourceBind を作成して DataSource プロパティを設定する:**

次に、UDBDataSource を作成し、DataSource のプロパティとしてユーザー ID とパスワードを設定する方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind2
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource " +
            "with cujo as the default " +
            "profile to connect with.");

        // Provide a user ID and password to be used for
        // connection requests.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS2", ds);
    }
}
```

**例: UDBDataSource をバインドする前に初期コンテキストを取得する:**

次の例では、UDBDataSource をバインドするにあたって、その前に初期コンテキストを取得します。その後、そのコンテキスト上で lookup メソッドを使用して、アプリケーションが使用する DataSource タイプのオブジェクトを戻します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Import the required packages. There is no
// driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();
```

```

// Retrieve the bound UDBDataSource object using the
// name with which it was previously bound. At runtime,
// only the DataSource interface is used, so there
// is no need to convert the object to the UDBDataSource
// implementation class. (There is no need to know what
// the implementation class is. The logical JNDI name is
// only required).
DataSource ds = (DataSource) ctx.lookup("SimpleDS");

// Once the DataSource is obtained, it can be used to establish
// a connection. This Connection object is the same type
// of object that is returned if the DriverManager approach
// to establishing connection is used. Thus, so everything from
// this point forward is exactly like any other JDBC
// application.
Connection connection = ds.getConnection();

// The connection can be used to create Statement objects and
// update the database or process queries as follows.
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
while (rs.next()) {
    System.out.println(rs.getString(1) + "." + rs.getString(2));
}

// The connection is closed before the application ends.
connection.close();
}
}

```

#### **例: UDBDataSource の作成、およびユーザー ID とパスワードの取得:**

以下は、UDBDataSource を作成し、実行時に getConnection メソッドを使用してユーザー ID とパスワードを取得する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

/// Import the required packages. There is
// no driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know
        // what the implementation class is. The logical JNDI name
        // is only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");
    }
}

```

```

// Once the DataSource is obtained, it can be used to establish
// a connection. The user profile cujo and password newtiger
// used to create the connection instead of any default user
// ID and password for the DataSource.
Connection connection = ds.getConnection("cujo", "newtiger");

// The connection can be used to create Statement objects and
// update the database or process queries as follows.
Statement statement = connection.createStatement();
ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
while (rs.next()) {
    System.out.println(rs.getString(1) + "." + rs.getString(2));
}

// The connection is closed before the application ends.
connection.close();
}
}

```

### **DataSource を UDBDataSource と共に使用する:**

DataSource インターフェースは、Java Database Connectivity (JDBC) ドライバーを使用する際に柔軟性が向上するように設計されました。

DataSource の使用法は、以下の 2 つの段階に分けることができます。

- **配置**

配置 (デプロイメント) とは、JDBC アプリケーションを実際に行うセットアップの段階のことです。一般的には、配置は特定のプロパティで DataSource をセットアップすること、その後 Java Naming and Directory Interface (JNDI) を使用してそれをディレクトリー・サービスにバインドすることを意味します。最も一般的な Lightweight Directory Access Protocol (LDAP) はディレクトリー・サービスですが、他にも Common Object Request Broker Architecture (CORBA) オブジェクト・サービス、Java リモート・メソッド呼び出し (RMI)、または基礎となるファイル・システムなど多数あります。

- **使用**

DataSource の実行時使用と配置を分離することにより、多数のアプリケーションで DataSource のセットアップを再使用できます。配置の一部の局面を変更すると、DataSource を使用するすべてのアプリケーションに、自動的に変更内容が反映されます。

**注:** RMI を使用することは、複雑な作業になる可能性があるということを銘記しておいてください。RMI をソリューションとして選ぶ前に、それにより多方面に波及する影響について必ず理解しておいてください。

DataSource の利点には、アプリケーション開発プロセスに直接影響を与えずに、アプリケーションに代わって JDBC ドライバーを稼働できることがあります。詳しくは、以下を参照してください。

- 接続プーリング
- ステートメント・プーリング
- 分散トランザクション

### **UDBDataSourceBind**

UDBDataSource を作成して JNDI とのバインドを取得する例として、UDBDataSourceBind プログラムがあります。このプログラムにより、要求されている基本タスクがすべて実行されます。つまり、

UDBDataSource オブジェクトがインスタンス化され、このオブジェクトに関するプロパティーが設定され、JNDI コンテキストが取り出され、このオブジェクトが JNDI コンテキスト中の名前とバインドされます。

配置時のコードは、ベンダーごとに固有です。このアプリケーションは、処理したい特定の DataSource の実装をインポートしなければなりません。インポート・リスト中で、パッケージ修飾 UDBDataSource クラスがインポートされます。このアプリケーションの最も知られていない点として、JNDI との協働がありません (Context オブジェクトを取り出してバインド呼び出しするなど)。追加情報については、Sun Microsystems, Inc. の JNDI を参照してください。

このプログラムを実行して正常に完了すると、JNDI ディレクトリー・サービス中に SimpleDS という新しい項目が入れられます。この項目は、JNDI コンテキストによって指定されている場所に入れられます。これで、DataSource のインプリメンテーションが配置されたこととなります。アプリケーション・プログラムは、この DataSource を使用して、データベース接続と JDBC 関連作業を取り出します。

## UDBDataSourceUse

前述の配置したアプリケーションを使用する JDBC アプリケーションの例として、UDBDataSourceUse プログラムがあります。

前述の例で、JDBC アプリケーションは、UDBDataSource をバインドする前の状態の初期コンテキストを入手します。その後、そのコンテキスト上で lookup メソッドを使用して、アプリケーションが使用する DataSource タイプのオブジェクトを戻します。

注: ランタイム・アプリケーションは、DataSource インターフェースのメソッドだけを作業対象とするので、インプリメンテーション・クラスを認識する必要はありません。このようにして、アプリケーションは移植可能になります。

UDBDataSourceUse は、ある企業で大規模な運用を実行する複雑なアプリケーションとします。その企業には、十指に余る大規模アプリケーションがあります。ネットワークにあるシステムの 1 つの名前を変更する必要があります。配置ツールを実行して、1 つの UDBDataSource プロパティーに変更を加えれば、すべてのアプリケーションのコードに変更を加えなくても、それらのアプリケーションにこの新しい名前を認識させることができます。DataSources の利点の 1 つに、システム・セットアップ情報を統合できることがあります。別の大きな利点は、接続プーリング、ステートメント・プーリング、および分散トランザクション・サポートなどの、アプリケーションに認識されない機能をドライバーが実装できることです。

UDBDataSourceBind と UDBDataSourceUse について綿密に分析したので、DataSource オブジェクトが実行内容を認識する方法を知りたいと思われることでしょう。どちらのプログラムにも、システム、ユーザー ID、またはパスワードを指定するコードはありません。UDBDataSource クラスのすべてのプロパティーにデフォルト値があります。デフォルトでは、実行中のアプリケーションのユーザー・プロファイルとパスワードを使用して、ローカル iSeries サーバーに接続します。その代わりにユーザー・プロファイル cujo を使用して確実に接続したい場合は、次の 2 つの方法で行うことができます。

- ユーザー ID とパスワードを DataSource プロパティーとして設定する。この手法の使用例については、例: UDBDataSourceBind の作成と DataSource プロパティーの設定を参照してください。
- DataSource getConnection メソッドを使用して、実行時にユーザー ID とパスワードを取得する。この手法の使用例については、例: UDBDataSource の作成、およびユーザー ID とパスワードの取得を参照してください。

UDBDataSource のプロパティを多数指定できるのと同様に、DriverManager を使用して作成する接続のプロパティも指定できます。ネイティブ JDBC ドライバーの、サポートされているプロパティのリストについては、DataSource プロパティを参照してください。

これらのリストは類似していますが、今後のリリースでも同様かどうかは保証しません。DataSource インターフェースのコーディングを始めることをお勧めします。

注: ネイティブ JDBC ドライバーにはその他の DataSource インプリメンテーションが 2 つありますが、これらのインプリメンテーションを直接使用することはお勧めしません。

- DB2DataSource
- DB2StdDataSource

#### DataSource プロパティ:

以下の表は、データ・ソース・プロパティとその値、およびその説明を示しています。

| Set メソッド (データ・タイプ)    | 値                             | 説明  |
|-----------------------|-------------------------------|---|
| setAccess(String)     | "all"、"read call"、"read only" | <p>このプロパティを使用すると、特定の接続で実行できる操作のタイプを制限できます。デフォルト値は "all" で、これは基本的に、その接続が Java Database Connectivity (JDBC) API への完全アクセスを持つことを意味します。</p> <p>"read call" 値は、その接続で照会の実行とストアード・プロシージャの呼び出しだけを許可します。SQL ステートメントを介してデータベースを更新しようとする、SQLException が発生します。</p> <p>"read only" 値は、その接続では照会の実行のみに制限します。ストアード・プロシージャの呼び出しや更新ステートメントを実行しようとする、SQLException が発生します。</p> |
| setBatchStyle(String) | "2.0"、"2.1"                   | <p>JDBC 2.1 仕様では、バッチ処理での更新で例外が発生した場合の処理として、2 番目のメソッドが定義されています。ドライバーは、どちらかの仕様に準拠できます。デフォルトでは、JDBC 2.0 仕様で定義された方法で処理されます。</p>   |



| Set メソッド (データ・タイプ)      | 値  | 説明   |
|-------------------------|--|--|
| setUseBlocking(boolean) | "true"、"false"                           | <p>このプロパティは、接続で <b>ResultSet</b> 行を検索するときにブロック化を行うかどうかを判別します。ブロック化により、<b>ResultSet</b> の処理のパフォーマンスは大きく改善される可能性があります。</p> <p>このプロパティは、デフォルトでは <b>true</b> に設定されています。</p>  |
| setBlockSize(int)       | "0"、"8"、"16"、"32"、"64"、"128"、"256"、"512" | <p>このプロパティは、<b>ResultSet</b> として一度に取り出される行数です。順方向に限られた通常の <b>ResultSet</b> の処理で、照会に合致する多くの行がデータベース内にあった場合は、このサイズのブロックが取得されます。JDBC ドライバーの内部記憶域にあるブロックの終わりに達したときにのみ、別のデータのブロックに対する要求がデータベースに送られます。</p> <p>この値は、<b>useBlocking</b> プロパティが <b>true</b> になっているときにのみ使用されます。詳細については、上記の <b>setUseBlocking</b> を参照してください。</p> <p>ブロック・サイズ・プロパティを "0" に設定すると、<b>SetUseBlocking(false)</b> と同じ効果が得られます。</p> <p>デフォルトでは、"32" のブロック・サイズでブロック化が行われます。これは、非常に独断的な決定であり、デフォルトは将来のリリースで変更される可能性があります。</p> <p>スクロール可能な <b>ResultSet</b> でブロック化は行われません。</p> <p>ブロック化は、ユーザーのアプリケーションのカーソルの感度に影響しません。感度の高いカーソルは、他の SQL ステートメントによる変更を認識します。しかし、データ・キャッシュのため、その変更はデータベースからデータをフェッチする必要がある場合にのみ、検出されます。</p> |

| Set メソッド (データ・タイプ)         | 値  | 説明   |
|----------------------------|--|--|
| setCursorHold(boolean)     | "true"、"false"   | <p>このプロパティは、トランザクションがコミットされた後も <code>ResultSet</code> をオープンにしたままにするかどうかを設定します。この値を <code>true</code> にすると、コミットが呼び出された後でも、アプリケーションはオープンしている <code>ResultSet</code> にアクセスすることができます。この値を <code>false</code> にすると、その接続の中でオープンしているカーソルはコミット時にすべてクローズされます。</p> <p>このプロパティは、デフォルトでは <code>true</code> に設定されています。</p> <p>このプロパティは、その接続で作成されたすべての <code>ResultSet</code> のデフォルト値となります。JDBC 3.0 では、カーソル・サポートが追加されました(詳しくは、<code>ResultSet</code> 特性セクションを参照してください)。このデフォルトは、アプリケーションによって別のカーソル・サポートを指定されると、置き換えられます。</p> |
| setDataTruncation(boolean) | "true"、"false"   | <p>このプロパティは、次のことを指定します。</p> <ul style="list-style-type: none"> <li>文字データが切り捨てられた場合に、警告および例外を生成する (<code>true</code>)</li> <li>文字データを黙って切り捨てる (<code>false</code>)</li> </ul> <p>詳細については、<code>DataTruncation</code> を参照してください。</p>   |
| setDatabaseName(String)    | 任意の名前  | <p>このプロパティでは、<code>DataSource</code> が接続しようとするデータベースを指定します。デフォルトは <code>*LOCAL</code> です。データベース名はアプリケーションが動作しているシステム上のリレーショナル・データベース・ディレクトリーに存在しているか、ローカル・システムを指定する特殊値 <code>*LOCAL</code> か <code>localhost</code> でなければなりません。</p>  |
| setDataSourceName(String)  | 任意の名前  | <p>このプロパティは、接続プーリングをサポートするために、<code>ConnectionPoolDataSource</code> JNDI 名を渡すことを許可します。</p>   |
| setDateFormat(String)      | "julian"、"mdy"、"dmy"、"ymd"、<br>"usa"、"iso"、"eur"、"jis" | <p>このプロパティは、日付のフォーマット方法を変更することができます。</p>   |

| Set メソッド (データ・タイプ)             | 値                       | 説明   |
|--------------------------------|-------------------------|--|
| setDateSeparator(String)       | "/", "-", ".", ":", "b" | このプロパティを使って、日付区切り文字を変更することができます。このプロパティは、(システム規則に従って) 一部の dateFormat 値との組み合わせでのみ有効です。                    |
| setDecimalSeparator(String)    | ","; "                  | このプロパティを使って、小数点を変更することができます。   |
| setDescription(String)         | 任意の名前                   | このプロパティでは、DataSource オブジェクトの記述を設定します。  |
| setDoEscapeProcessing(boolean) | "true"、"false"          | このプロパティは、SQL ステートメントのエスケープ処理が行われているかどうかを指定します。<br><br>このプロパティのデフォルト値は true です。                           |
| setFullErrors(boolean)         | "true"、"false"          | このプロパティは、全システムの 2 次レベル・エラー・テキストを SQLException オブジェクト・メッセージに戻すかどうかを指定します。デフォルトは false です。                 |
| setLibraries(String)           | スペースで区切られたライブラリーのリスト    | このプロパティを使うと、ライブラリーのリストをサーバー・ジョブのライブラリー・リストに格納することができます。このプロパティは setSystemNaming(true) が使用されているときのみ使用します。 |
| setLobThreshold(int)           | 500000 未満の任意の値          | このプロパティは、ドライバーに対して、LOB 列がしきい値サイズよりも小さかった場合に、Locator Object (LOB) ロケーターの代わりに実際の値を格納することを指定します。            |
| setLoginTimeout(int)           | 任意の値                    | このプロパティは、現在は無視されますが、今後使用される予定です。   |
| setNetworkProtocol(int)        | 任意の値                    | このプロパティは、現在は無視されますが、今後使用される予定です。   |
| setPassword(String)            | 任意のストリング                | このプロパティでは、接続用のパスワードを指定することができます。ユーザー ID が設定されていない場合、このプロパティは無視されます。                                      |
| setPortNumber(int)             | 任意の値                    | このプロパティは、現在は無視されますが、今後使用される予定です。   |
| setPrefetch(boolean)           | "true"、"false"          | このプロパティは、ドライバーが処理の直後に ResultSet の最初のデータをフェッチするのか、データが要求されてからフェッチするのかを指定します。デフォルトは true です。               |

| Set メソッド (データ・タイプ)            | 値                             | 説明  |
|-------------------------------|-------------------------------|---|
| setReuseObjects(boolean)      | "true"、"false"                | このプロパティは、一度クローズされた、ある種類のオブジェクトをドライバーが再利用するかどうかを指定します。これはパフォーマンスを向上させます。デフォルトは true です。  |
| setServerName(String)         | 任意の名前                         | このプロパティは、現在は無視されますが、今後使用される予定です。  |
| setServerTraceCategories(int) | 整数として表記されるストリング               | このプロパティで、JDBC サーバー・ジョブをトレースすることが可能になります。サーバー・トレースが使用可能な場合、クライアントがサーバーに接続したときにトレースが開始され、接続が切断されたときにトレースが終了します。<br><br>トレース・データは、サーバー上のスプール・ファイル内に収集されます。複数レベルのサーバー・トレースは、定数を加算することと、set メソッドでその合計を渡すことの組み合わせによってオンになります。<br><b>注:</b> このプロパティは、一般にはサポート担当者によって使用されるもので、この値に関するこれ以上の説明はありません。 |
| setSystemNaming(boolean)      | "true"、"false"                | このプロパティは、コレクションおよびテーブルがピリオドによって区切られるか (SQL 命名)、スラッシュによって区切られるか (システム命名) を指定します。このプロパティによって、デフォルト・ライブラリーを使用するか (SQL 命名)、それともライブラリー・リストを使用するか (システム命名) についても判断されます。デフォルトは、SQL 命名です。   |
| setTimeFormat(String)         | "hms"、"usa"、"iso"、"eur"、"jis" | このプロパティを使って、時刻のフォーマット方法を変更することができます。  |
| setTimeSeparator(String)      | ":", ".", ",", "b"            | このプロパティを使って、時刻区切り文字を変更することができます。このプロパティは、(システム規則に従って) 一部の timeFormat 値との組み合わせでのみ有効です。   |
| setTrace(boolean)             | "true"、"false"                | このプロパティは、単純トレースを使用可能にします。デフォルト値は false です。  |

| Set メソッド (データ・タイプ)                   | 値   | 説明   |
|--------------------------------------|---|--|
| setTransactionIsolationLevel(String) | "none"、"read committed"、"read uncommitted"、"repeatable read"、"serializable" | このプロパティーでは、トランザクション分離レベルを指定することができます。 JDBC のデフォルトは自動コミット・モードなので、このプロパティーのデフォルト値は "none" です。  |
| setTranslateBinary(Boolean)          | "true"、"false"  | このプロパティーを使用すれば、JDBC ドライバーで binary データ値や varbinary データ値を強制的に char データ値や varchar データ値と同様に扱うことができます。<br><br>このプロパティーのデフォルト値は false です。  |
| setUseBlockInsert(boolean)           | "true"、"false"  | このプロパティーは、データベースにデータのブロックを挿入するために、ネイティブ JDBC ドライバーがブロック挿入モードになることを可能にします。これはバッチ更新の最適化バージョンです。この最適化モードは、あるシステム制約への違反、またデータ挿入の障害およびデータの破壊が発生する恐れのないアプリケーション内でのみ使用できます。<br><br>このプロパティーをオンにしたアプリケーションは、バッチ更新を行おうとするときにはローカル・システムにのみ接続します。ブロック更新は DRDA 上では管理できないため、DRDA を使ってリモート接続を確立することはしません。<br><br>また、アプリケーションでは SQL 挿入ステートメントで PreparedStatements が使われており、すべての挿入値パラメーターが VALUES 文節によって指定されている必要があります。値リストで定数を使うことは許可されていません。これは、システムのブロック挿入エンジンの要件です。<br><br>デフォルトは false です。 |
| setUser(String)                      | 任意の値  | このプロパティーは、接続を取得するためのユーザー ID を設定できます。このプロパティーを設定する際には、password プロパティーも設定する必要があります。  |

その他の DataSource インプリメンテーション:

ネイティブ JDBC ドライバーに組み込まれる DataSource インターフェースのインプリメンテーションは 2 つあります。これらの DataSource インプリメンテーションは、使用しないようにする必要があります。それらを使用することはできますが、将来行われる改良には対応していません。たとえば、これらのインプリメンテーションには、堅固な接続およびステートメントのプーリングは追加されていません。これらのインプリメンテーションは、UDBDataSource インターフェースとその関連機能を採用するまで存続します。

## DB2DataSource

DB2DataSource は、DataSource インターフェースの初期のインプリメンテーションであり、完全な仕様に準拠していません（つまり、仕様に先立つものです）。これまで DB2DataSource が存続している理由は、WebSphere® ユーザーが、現行リリースにマイグレーションできるようにするためだけであり、それ以外に使用すべきではありません。

## DB2StdDataSource

DB2StdDataSource は、改訂されたバージョンの DB2DataSource インプリメンテーションであり、JDBC オプション・パッケージ仕様が最終になったら、仕様準拠になります。すでに DB2DataSource バージョンで作成されたコードを中断しないために、新しいバージョンが提供されています。

これらの DataSource インプリメンテーションを利用するアプリケーションを作成した場合、以前のプロパティーがすべてサポートされているため、UDBDataSource へマイグレーションすることは簡単な作業になります。UDBDataSource へマイグレーションして、新しい UDBDataSource クラスの機能を装備することをお勧めします。

## JDBC 用の JVM プロパティー

ネイティブ JDBC ドライバーが使用する設定の一部は、接続プロパティーを使用しては設定できません。これらの設定は、ネイティブ JDBC ドライバーが実行する JVM に対して設定しなければならないものです。これらの設定は、ネイティブ JDBC ドライバーで作成されるすべての接続に使用されます。

ネイティブ・ドライバーは以下の JVM プロパティーを認識します。

| プロパティー                     | 値             | 意味  |
|----------------------------|---------------|---|
| jdbc.db2.job.sort.sequence | デフォルト値 = *HEX | このプロパティーを true に設定すると、ネイティブ JDBC ドライバーは、デフォルト値の *HEX を使用する代わりに、ジョブを開始したユーザーのジョブ・ソート・シーケンスを使用ようになります。これを他の値に設定する場合や、設定しないでおく場合、JDBC は引き続きデフォルトの *HEX を使用します。これが意味することに十分注意してください。JDBC 接続が接続要求で異なるユーザー・プロファイルを渡す場合、すべての接続に対して、サーバーを開始したユーザー・プロファイルのソート・シーケンスが使用されます。これは始動時に設定される環境属性であり、動的な接続属性ではありません。 |

| プロパティ                 | 値  | 意味  |
|-----------------------|--|---|
| jdbc.db2.trace        | 1 または error = エラー情報をトレース。 2 または info = 情報およびエラー情報をトレース。 3 または verbose = 詳細、情報、およびエラー情報をトレース。 4、all、または true = 可能なすべての情報をトレース。  | このプロパティは JDBC ドライバーのトレースをオンにします。これは、問題を報告する場合に使用してください。 |
| jdbc.db2.trace.config | stdout = トレース情報は stdout に送信されます (デフォルト値)。 usrtc = トレース情報はユーザー・トレースに送信されます。トレース情報を入手するには、CL コマンド「ユーザー・トレース・バッファのダンプ (DMPUSRTRC)」を使用することができます。 file://<pathtofile> = トレース情報はファイルに送信されます。ファイル名に "%j" が含まれている場合、"%j" はジョブ名で置き換えられます。 <pathtofile> の例は、/tmp/jdbc.%j.trace.txt です。 | このプロパティは、トレースの出力先を指定するために使用します。                         |

## IBM Developer Kit for Java 用の DatabaseMetaData インターフェース

DatabaseMetaData インターフェースは、IBM Developer Kit for Java の JDBC ドライバーによって実装され、基礎となるデータ・ソースに関連した情報を提供します。これは、提供されているデータ・ソースとの対話方法を決定するため、主にアプリケーション・サーバーとツールによって使用されます。アプリケーションは、DatabaseMetaData メソッドを使用してもデータ・ソースの情報を入手することができますが、こちらはそれほど一般的ではありません。

DatabaseMetaData インターフェースには 150 を超えるメソッドが組み込まれており、これらのメソッドは提供する情報のタイプによって分類されています。これらについては以下で説明します。

DatabaseMetaData インターフェースには、40 を超えるフィールドも含まれており、それらのフィールドは、さまざまな DatabaseMetaData メソッドの戻り値として使用される定数となります。

DatabaseMetaData インターフェースのメソッドの変更点の詳細については、『JDBC 3.0 の変更点』を参照してください。

### DatabaseMetaData オブジェクトを作成する

DatabaseMetaData オブジェクトは、Connection メソッド getMetaData によって作成されます。オブジェクトが作成されると、これを用いて基礎となるデータ・ソースの情報を動的に発見することができます。以下の例では、DatabaseMetaData オブジェクトを作成し、これを使ってテーブル名の最大文字数を判別する方法を示します。

**例:** DatabaseMetaData オブジェクトを作成する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// con is a Connection object
DatabaseMetaData dbmd = con.getMetadata();
int maxLen = dbmd.getMaxTableNameLength();
```



## 一般情報を取得する

DatabaseMetaData の一部のメソッドは、データ・ソースに関する一般情報と、その実装に関する詳細を動的に発見するために使用されます。これらの重要なメソッドには、次のようなものがあります。

- getURL
- getUserName
- getDatabaseProductVersion、getDriverMajorVersion、および getDriverMinorVersion
- getSchemaTerm、getCatalogTerm、および getProcedureTerm
- nullsAreSortedHigh、および nullsAreSortedLow
- usesLocalFiles、および usesLocalFilePerTable
- getSQLKeywords

## フィーチャー・サポートを判別する

DatabaseMetaData メソッドの多くは、特定のフィーチャーやフィーチャー・セットが、ドライバーや基礎となっているデータ・ソースでサポートされているかどうかを判別するために使用できます。これに加えて、一部のメソッドは提供されているサポートのレベルを記述します。個々のフィーチャーのサポートを記述するメソッドには、次のようなものがあります。

- supportsAlterTableWithDropColumn
- supportsBatchUpdates
- supportsTableCorrelationNames
- supportsPositionedDelete
- supportsFullOuterJoins
- supportsStoredProcedures
- supportsMixedCaseQuotedIdentifiers

フィーチャー・サポートのレベルを記述するためのメソッドには、次のようなものがあります。

- supportsANSI92EntryLevelSQL
- supportsCoreSQLGrammar

## データ・ソースの制限

別のグループのメソッドは、特定のデータ・ソースによって課されている制限を提供します。このカテゴリのメソッドには次のようなものがあります。

- getMaxRowSize
- getMaxStatementLength
- getMaxTablesInSelect
- getMaxConnections
- getMaxCharLiteralLength
- getMaxColumnsInTable

このグループのメソッドは、制限値を `integer` として返します。戻り値ゼロは、制限がないか、不明であることを示します。

## SQL オブジェクトとその属性

DatabaseMetaData のいくつかのメソッドは、特定のデータ・ソースに移植される SQL オブジェクトに関する情報を提供します。これらのメソッドは SQL オブジェクトの属性を判別することができます。また、これらのメソッドは各行に特定のオブジェクトが記述された、ResultSet オブジェクトを返します。たとえば、getUDTs メソッドはデータ・ソース内で定義されている各 UDT に対応した行を持つ ResultSet オブジェクトを返します。このカテゴリーには、たとえば次のようなメソッドがあります。

- getSchemas および getCatalogs
- getTables
- getPrimaryKeys
- getProcedures および getProcedureColumns
- getUDTs

## トランザクション・サポート

少数のメソッドのグループは、データ・ソースがサポートするトランザクション・セマンティクスに関する情報を提供します。このカテゴリーには、たとえば次のようなメソッドがあります。

- supportsMultipleTransactions
- getDefaultTransactionIsolation

DatabaseMetaData インターフェースの使用法の例については、例: IBM Developer Kit for Java 用の DatabaseMetaData インターフェースを参照してください。

## JDBC 3.0 の変更点

JDBC 3.0 のいくつかのメソッドでは、戻り値が変更されています。JDBC 3.0 では、以下のメソッドが戻す ResultSet にフィールドが追加されました。

- getTables
- getColumns
- getUDTs
- getSchemas

注: Java Development Kit (JDK) 1.4 を使用してアプリケーションを開発している場合、テストの際に一定数の列が戻されることがあります。ユーザーが作成するアプリケーションで、すべての列にアクセスすることを想定しています。しかし、そのアプリケーションが以前のリリースの JDK でも動作するように設計した場合、これらのフィールドにアクセスしようとする、アプリケーションは SQLException を受け取ります。以前の JDK のリリースではこれらのフィールドが存在しないためです。

SafeGetUDTs では、いくつかの JDK のリリースで動作するアプリケーションの記述方法の例が示されています。

### 例: IBM Developer Kit for Java 用の DatabaseMetaData インターフェース:

次の例は、テーブルのリストを返す方法を示しています。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Connect to iSeries server.  
Connection c = DriverManager.getConnection("jdbc:db2:mySystem");
```

```

// Get the database meta data from the connection.
DatabaseMetaData dbMeta = c.getMetaData();

// Get a list of tables matching this criteria.
String catalog = "myCatalog";
String schema = "mySchema";
String table = "myTable%"; // % indicates search pattern
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"};
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterate through the ResultSet to get the values.

// Close the connection.
c.close();

```

詳細については、IBM Developer Kit for Java 用の DatabaseMetaData インターフェースを参照してください。

#### 例: 複数の列を持ったメタデータ ResultSet を使用する:

以下は、複数の列があるメタデータ ResultSet を使用方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// SafeGetUDTs example. This program demonstrates one way to deal with
// metadata ResultSets that have more columns in JDK 1.4 than they
// had in previous releases.
//
// Command syntax:
//   java SafeGetUDTs
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import java.sql.*;

public class SafeGetUDTs {

    public static int jdbcLevel;

```

```

// Note: Static block runs before main begins.
// Therefore, there is access to jdbcLevel in
// main.
{
    try {
        Class.forName("java.sql.Blob");

        try {
            Class.forName("java.sql.ParameterMetaData");
            // Found a JDBC 3.0 interface. Must support JDBC 3.0.
            jdbcLevel = 3;
        } catch (ClassNotFoundException ez) {
            // Could not find the JDBC 3.0 ParameterMetaData class.
            // Must be running under a JVM with only JDBC 2.0
            // support.
            jdbcLevel = 2;
        }

    } catch (ClassNotFoundException ex) {
        // Could not find the JDBC 2.0 Blob class. Must be
        // running under a JVM with only JDBC 1.0 support.
        jdbcLevel = 1;
    }
}

// Program entry point.
public static void main(java.lang.String[] args)
{
    Connection c = null;

    try {
        // Get the driver registered.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        c = DriverManager.getConnection("jdbc:db2:*local");
        DatabaseMetaData dmd = c.getMetaData();

        if (jdbcLevel == 1) {
            System.out.println("No support is provided for getUDTs. Just return.");
            System.exit(1);
        }

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN%", null);
        while (rs.next()) {

            // Fetch all the columns that have been available since the
            // JDBC 2.0 release.
            System.out.println("TYPE_CAT is " + rs.getString("TYPE_CAT"));
            System.out.println("TYPE_SCHEM is " + rs.getString("TYPE_SCHEM"));
            System.out.println("TYPE_NAME is " + rs.getString("TYPE_NAME"));
            System.out.println("CLASS_NAME is " + rs.getString("CLASS_NAME"));
            System.out.println("DATA_TYPE is " + rs.getString("DATA_TYPE"));
            System.out.println("REMARKS is " + rs.getString("REMARKS"));

            // Fetch all the columns that were added in JDBC 3.0.
            if (jdbcLevel > 2) {
                System.out.println("BASE_TYPE is " + rs.getString("BASE_TYPE"));
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    } finally {
        if (c != null) {
            try {
                c.close();
            } catch (SQLException e) {

```

```
        // Ignoring shutdown exception.
    }
}
}
}
```

## 例外

Java 言語では、プログラムのエラー処理機能を提供するために、例外を使用します。例外は、プログラムを実行しているときに、命令の通常フローが中断されたときに発生するイベントです。

Java ランタイム・システムおよび Java パッケージの多くのクラスは、いくつかの状況で `throw` ステートメントを使って例外をスローしています。この同じメカニズムを使って、ユーザーの Java プログラムでも例外をスローすることができます。

### SQLException:

`SQLException` クラスとそのサブタイプは、データ・ソースへのアクセス時に発生したエラーや警告に関する情報を提供します。

インターフェースで定義されている大半の `JDBC` とは異なり、例外サポートはクラスによって提供されています。 `JDBC` アプリケーションが実行されているときに発生する例外のための基本クラスは、`SQLException` です。 `JDBC API` のすべてのメソッドは、`SQLException` がスローできるように宣言されています。 `SQLException` は `java.lang.Exception` の拡張で、データベース・コンテキスト内で発生した障害に関連した追加情報を提供します。具体的には、`SQLException` からの次のような情報が使用可能です。

- テキスト記述
- `SQLState`
- エラー・コード
- 共に発生した他の例外への参照

`ExceptionExample`。これは、(この場合は、予測された) `SQLException` のキャッチと、提供されたすべての情報をダンプする適切な処理を行うプログラムです。

注: `JDBC` では、複数の例外を合わせてチェーニングするメカニズムが提供されています。これにより、ドライバまたはデータベースが、一度の要求で複数のエラーをレポートすることができます。現在のところ、ネイティブ `JDBC` ドライバが実際にこれを行う実例はありません。この情報は参照用として提供されており、ドライバが今後もこれを行わないことを明示するものではありません。

前述のように、エラーが発生すると `SQLException` オブジェクトがスローされます。これは正しい動作ですが、完全な全体像ではありません。現実には、ネイティブ `JDBC` ドライバが実際にスローすることはまれです。その `SQLException` サブクラスのインスタンスがスローされます。これにより、以下にあるように、実際にどんな障害であったのかに関する詳しい情報を判別することができます。

### DB2Exception.java

`DB2Exception` オブジェクトが直接スローされることはありません。この基本クラスは、すべての `JDBC` 例外に共通の機能性を保持するために使用されます。このクラスには、`JDBC` がスローする標準例外となる 2 つのサブクラスがあります。これらのサブクラスとは、`DB2DBException.java` および `DB2JDBCException.java` です。 `DB2DBException` は、データベースから直接レポートされた例外です。 `DB2JDBCExceptions` は、`JDBC` ドライバが、ドライバ自身に問題を発見したときにスローされます。

この方法で例外クラスが階層に分割されていることで、2 つのタイプの例外を個別に処理することができます。

## DB2DBException.java

前述のように、DB2DBExceptions はデータベースから直接送信される例外です。これらは、JDBC ドライバーが CLI への呼び出しを行い、SQLERROR 戻りコードが返されたときに発生します。このケースでは、CLI 機能の SQLException は、メッセージ・テキスト、SQLState、およびベンダー・コードを取得するために呼び出されます。SQLMessage の置換テキストも取得され、戻されます。DatabaseException クラスは、データベースが認識し、例外オブジェクトを構築するために JDBC ドライバーにレポートするエラーを発生させます。

## DB2JDBCException.java

DB2JDBCException は JDBC ドライバー自身からのエラー状態によって生成されます。この例外クラスの機能には基本的な違いがあります。JDBC ドライバーそのものは例外のメッセージの言語翻訳を処理し、オペレーティング・システムおよびデータベースが処理するその他の問題の例外については、データベース内で生成されます。可能な限り、JDBC ドライバーはデータベースの SQLStates を順守します。JDBC ドライバーがスローする例外のベンダー・コードは、常に -99999 です。しばしば、CLI 層によって認識され、戻される DB2DBException のエラー・コードも -99999 です。JDBCException クラスは、JDBC ドライバーが自身の例外を認識し、構築するエラーを発生させます。リリースの開発中の実行時には、後続の出力が生成されます。スタックの最上部に DB2JDBCException が含まれていることに注意してください。これは、常にデータベースへの要求が行われる前にエラーが JDBC ドライバーからレポートされることを示しています。

### 例: SQLException:

以下に、SQLException をキャッチし、提供されたすべての情報をダンプする例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;

public class ExceptionExample {

    public static Connection connection = null;

    public static void main(java.lang.String[] args) {

        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            int count = s.executeUpdate("insert into cujofake.cujofake values(1, 2,3)");

            System.out.println("Did not expect that table to exist.");

        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:....." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("-----");
            e.printStackTrace();
        } catch (Exception ex) {
```



```

        System.out.println("An exception other than an SQLException was thrown: ");
        ex.printStackTrace();
    } finally {
        try {
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            System.out.println("Exception caught attempting to shutdown...");
        }
    }
}
}
}

```

## SQLWarning:

一部のインターフェースのメソッドでは、データベース・アクセス警告を出すときに `SQLWarning` オブジェクトが生成されます。

以下のインターフェースのメソッドが `SQLWarning` を生成できます。

- `Connection`
- `Statement` とそのサブタイプ `PreparedStatement` および `CallableStatement`
- `ResultSet`

メソッドが `SQLWarning` オブジェクトを生成しても、データ・アクセス警告が発生したことは呼び出し元には通知されません。 `SQLWarning` オブジェクトを取り出すには、適切なオブジェクトで `getWarnings` メソッドを呼び出す必要があります。ただし、ある状況では、`SQLWarning` の `DataTruncation` サブクラスがスローされることがあります。ネイティブ JDBC ドライバーは、効率の向上のため、データベースが生成した一部の警告を無視する場合があります。たとえば、`ResultSet` の終わりに達した後でユーザーが `ResultSet.next` メソッドを使用してデータを取り出そうとすると、システムは警告を生成します。このような場合、`next` メソッドは、`true` ではなく `false` を返してユーザーにエラーを通知するように定義されています。これを改めて記述するオブジェクトを作成する必要はないので、この警告はただ無視されます。

複数のデータ・アクセス警告が発生すると、それらは最初の警告に連鎖されます。これらは、`SQLWarning.getNextWarning` メソッドを呼び出すことによって検索できます。連鎖した警告の終わりに達すると、`getNextWarning` は `null` を返します。

以降の `SQLWarning` オブジェクトは、次のステートメントが処理されるまで、その連鎖に追加され続けていきます。 `ResultSet` オブジェクトの場合には、カーソルが再配置されて連鎖した `SQLWarning` オブジェクトがすべて除去されるまで、その連鎖に追加され続けていきます。これにより、結果としてチェーン内のすべての `SQLWarning` オブジェクトが除去されます。

`Connection`、`Statement`、および `ResultSet` オブジェクトの使用は、`SQLWarnings` の生成される原因となることがあります。 `SQLWarning` は、特定の操作が正常に完了される間に注意すべき他の情報が存在した可能性を示す、通知メッセージです。 `SQLWarning` は、`SQLException` クラスの拡張機能ですが、スローされるものではなく、代わりに、その生成の原因となったオブジェクトに付加されます。なお、`SQLWarning` が生成されても、警告が生成されたことをアプリケーションに通知するイベントは何も起こりません。アプリケーションは、自発的に警告の情報を要求する必要があります。

`SQLException` と同様、`SQLWarning` は、相互に連鎖させることができます。オブジェクトの警告を消去するには、その `Connection`、`Statement`、または `ResultSet` といった各オブジェクトで、`clearWarnings` メソッドを呼び出すことができます。



注: clearWarnings メソッドを呼び出しても、すべての警告が消去されるわけではありません。消去されるのは、その特定のオブジェクトに関連付けられている警告だけです。

SQLWarning オブジェクトが手動で消去されない場合は、JDBC ドライバーが、ある特定のタイミングでそれらを消去します。SQLWarning オブジェクトは、以下のアクションが行われたときに消去されます。

- Connection インターフェースの場合は、新規の Statement、PreparedStatement、または CallableStatement オブジェクトが作成されると、警告が消去されます。
- Statement インターフェースの場合は、次のステートメントが処理される (あるいは、PreparedStatement や CallableStatement のためにもう一度同じステートメントが処理される) と、警告が消去されます。
- ResultSet インターフェースの場合は、カーソルが再配置されると、警告が消去されます。

### DataTruncation およびサイレント・トランケーション:

DataTruncation は SQLWarning のサブクラスです。SQLWarning がスローされていないときに DataTruncation オブジェクトがスローされることがあり、他の SQLWarning オブジェクトのように付加されます。サイレント・トランケーションは、列のサイズが setMaxFieldSize ステートメント・メソッドで指定されたサイズを超える場合に起こります。しかし警告または例外は報告されません。

DataTruncation オブジェクトは SQLWarning が戻す情報よりも詳細な追加情報を提供します。そのような追加情報には、次のようなものがあります。

- 転送されたデータのバイト数。
- 切り捨てられた列、またはパラメーター索引。
- 索引がパラメーター用か、あるいは ResultSet 列用か。
- 切り捨てが、データベースから読み取り中に発生したか、書き込み中に発生したか。
- 実際に転送されたデータ量。

場合によっては、この情報を判読することはできますが、直感的にはわからないこともあります。たとえば、PreparedStatement の setFloat メソッドを使って、整数値を持つ列に値を挿入しようとした場合、その列が保持できる最大値よりも大きな値を挿入しようとして DataTruncation が返されることがあります。この場合は、切り捨てられたバイト数は意味がなく、ドライバーにとっては切り捨て情報が提供されることが重要です。

### set() および update() メソッドの報告

JDBC ドライバーの間にも若干の違いがあります。ネイティブのドライバーや IBM Toolbox for Java JDBC ドライバーなどの一部のドライバーでは、パラメーターで設定した時点でデータ切り捨てをキャッチし、報告します。これは、PreparedStatement の set メソッドか、ResultSet の update メソッドのどちらかで行われます。その他のドライバーでは、ステートメントを処理したときに問題が報告され、execute、executeQuery、または updateRow メソッドが完了したときに報告されます。

不正なデータを提供した際に問題を報告する代わりに、それ以降の処理が継続できないときに問題を報告することには、次の 2 つの利点があります。

- 処理時に問題に対処する代わりに、問題が発生したときにアプリケーション内の障害に対処することができる。
- パラメーターを設定するときに検査することにより、JDBC ドライバーは、ステートメントの処理時にデータベースに渡す値が正しいことを確認できる。これにより、データベースは作業を最適化して、処理を早く完了することができます。

## ResultSet.update() メソッドが DataTruncation 例外をスローする

以前のいくつかのリリースでは、ResultSet.update() メソッドは切り捨て条件が存在すると警告を通知しました。これは、データ値をデータベースに挿入しようとするときに発生します。仕様では、このような状況が発生した場合、JDBC ドライバーは例外をスローするように決められています。その結果、JDBC ドライバーはこの方法で動作します。

データ切り捨てエラーを受け取る ResultSet 更新機能进行处理することと、エラーを受け取る更新または挿入ステートメントの PreparedStatement パラメーター・セット进行处理することには、大きな違いはありません。どちらのケースでも、問題は同一です。ユーザーが望んでいたものに適合しないデータが提供されたことです。

NUMERIC および DECIMAL は、小数点の右側を黙って切り捨てます。JDBC for UDB NT の動作でも、iSeries サーバー上で対話式 SQL で動作させた場合でも、この切り捨てが行われます。

注: データ切り捨てが発生すると、値は丸められません。NUMERIC または DECIMAL 列に適合しない、パラメーターの端数部分は、警告なしに単に失われます。

以下の例では、PreparedStatement 上のパラメーターによって、VALUES 文節の値が実際にセットされていることを想定しています。

```
create table cujosql.test (col1 numeric(4,2))
a) insert into cujosql.test values(22.22) // works - inserts 22.22
b) insert into cujosql.test values(22.223) // works - inserts 22.22
c) insert into cujosql.test values(22.227) // works - inserts 22.22
d) insert into cujosql.test values(322.22) // fails - Conversion error on assignment to column COL1.
```

### データ切り捨て警告と、データ切り捨て例外との違い

仕様では、データベースに書き込まれる値のデータ切り捨ては、例外をスローするように決められています。データベースに書き込まれる値のデータ切り捨てが行われなかった場合は、警告が生成されます。これは、データ切り捨ての状況がどのポイントかを識別されることを意味しており、データの切り捨てが処理されたステートメントのタイプにも注意する必要があります。この要件に関連して、以下にいくつかの SQL ステートメントのタイプの動作を示します。

- SELECT ステートメントでは、照会パラメーターはデータベースの内容に損傷を与えることはありません。したがって、データ切り捨ては常に警告の通知として扱われます。
- VALUES INTO および SET<sup>TM</sup> ステートメントでは、入力値は出力値を生成するためだけに使用されません。その結果、警告が発行されます。
- CALL ステートメントでは、JDBC ドライバーはパラメーターが与えられたストアド・プロシージャを判別することができません。ストアド・プロシージャのパラメーターの切り捨てが行われると、常に例外がスローされます。
- その他のすべてのステートメント・タイプでは、警告が通知されるのではなく、例外がスローされます。

### Connection および DataSource のデータ切り捨てプロパティー

多くのリリースでデータ切り捨てプロパティーが使用可能になっています。このプロパティーのデフォルトは true で、データ切り捨ての事象はチェックされ、警告の通知または例外のスローが行われます。このプロパティーは、値がデータベースの列に適合するかどうかの問題にならない場合に、便宜とパフォーマンスのために提供されています。列に挿入できる形で値を挿入するため、ドライバーが使用されます。

## 文字およびバイナリー・ベースのデータ・タイプにのみ効果のあるデータ切り捨てプロパティ

2 つ前のリリースでは、データ切り捨て例外をスローするかどうかはデータ切り捨てプロパティによって判断されました。このデータ切り捨てプロパティは、JDBC アプリケーションにとって切り捨てが重要ではないときに、切り捨てられた値を無視できるように用意されています。アプリケーションが DECIMAL(2,0) に 100 を挿入しようとしたとき、データベースに 00 または 10 のどちらかを格納したいというケースがあります。そのため、JDBC データ切り捨てプロパティは、パラメーターが CHAR、VARCHAR、CHAR FOR BIT DATA、および VARCHAR FOR BIT DATA のような文字ベースのタイプの状況でのみ有効になるよう、変更されました。

### パラメーターにのみ適用されるデータ切り捨てプロパティ

データ切り捨てプロパティは、JDBC ドライバーの設定で、データベースの設定ではありません。そのため、ステートメント・リテラルには影響はありません。たとえば、データベース内の CHAR(8) 列に値を挿入する処理を行う以下のステートメントは、データ切り捨てフラグが false に設定されて失敗します (接続は java.sql.Connection オブジェクトとして別の場所で割り当てられていることを前提としています)。

```
Statement stmt = connection.createStatement();
Stmt.executeUpdate("create table cujosql.test (col1 char(8))");
Stmt.executeUpdate("insert into cujosql.test values('dettinger')");
// Fails as the value does not fit into database column.
```

### 問題とならないデータ切り捨てに対するネイティブ JDBC ドライバーの例外のスロー

ネイティブ JDBC ドライバーは、パラメーターとして提供されたデータを確認することは行いません。これは処理をスローダウンさせるだけです。しかし、値の切り捨てが問題にならない状況で、データ切り捨て接続プロパティを false に設定していない状態では、この状況が発生することがあります。

たとえば、CHAR(10) である 'dettinger' が渡されると、適合する値のすべてが重要なものであるとしても、例外がスローされます。これは、JDBC for UDB NT で発生する動作です。しかし、SQL ステートメント内でリテラルとして値を渡した場合は、このような振る舞いは得られません。このような場合、データベース・エンジンは追加のスペースを暗黙のうちにスローアウトします。

JDBC が例外をスローしない問題は、次のとおりです。

- 必要かどうかにかかわらず、すべての set メソッドでパフォーマンスのオーバーヘッドが拡大します。たいいていの場合、これは有益なことではなく、setString() のような関数で相当なパフォーマンスのオーバーヘッドがあります。
- 渡されたストリング値でトリム関数を呼び出すなど、次善策は小さなものです。
- データベースの列に関する考慮すべき問題があります。CCSID 37 でのスペースは、CCSID 65535 または 13488 でのスペースとは全く異なります。

### サイレント・トランザクション

setMaxFieldSize ステートメント・メソッドは、任意の列の最大フィールド・サイズを指定できます。最大フィールド・サイズ値を超えたためにデータの切り捨てが行われた場合、警告や例外は報告されません。このメソッドは、前述のデータ切り捨てプロパティと同じように、CHAR、VARCHAR、CHAR FOR BIT DATA、および VARCHAR FOR BIT DATA のような文字ベースのタイプでのみ有効です。

### トランザクション

トランザクションは作業の論理単位です。作業の論理単位を完了するには、データベースに対していくつかのアクションを実行しなければならない場合があります。

トランザクション・サポートは、アプリケーションが以下を行うことを可能にします。

- 作業論理単位を完了するためのすべてのステップに従う。
- 作業単位を完了するためのステップの 1 つが失敗した場合に、作業論理単位一部として実行されたすべての作業を元に戻し、データベースをトランザクションが開始される前の状態に戻す。

トランザクションは、同時アクセスの間、データの整合性、正しいアプリケーション・セマンティクス、および矛盾のないデータのビューを提供するために使用されます。トランザクションのサポートには、常に Java Database Connectivity (JDBC) に準拠したドライバーが使用されなければなりません。

**注:** このセクションは、ローカル・トランザクションと、トランザクションの標準的な JDBC 概念を説明しているに過ぎません。Java やネイティブ JDBC ドライバーは、Java Transaction API (JTA)、分散トランザクション、および 2 フェーズ・コミット・プロトコル (2PC) をサポートします。

すべてのトランザクション作業は、Connection オブジェクト・レベルで処理されます。トランザクションの作業が完了すると、作業は、commit メソッドを呼び出すことによって終了処理できます。アプリケーションがトランザクションを打ち切る場合は、rollback メソッドが呼び出されます。

接続の下にあるすべての Statement オブジェクトは、トランザクションの一部になります。つまり、アプリケーションが 3 つの Statement オブジェクトを作成して、データベースに変更を加えるためにそれらの各オブジェクトを使用する場合は、commit または rollback 呼び出しが行われると、その 3 つのステートメントすべての作業が永続的にコミットされたり、ロールバックして廃棄されたりします。

純粋に SQL を使用して作業する場合は、トランザクションの終了処理に commit および rollback SQL ステートメントを使用します。これらの SQL ステートメントは、動的には作成できません。また、JDBC アプリケーションでは、これらのステートメントを使用したトランザクションの終了は試さないでください。

#### 自動コミット・モード:

デフォルトでは、JDBC は自動コミットと呼ばれる操作モードを使用します。このモードでは、データベースに対するすべての更新が即時に永続的にコミットされます。

ただし、自動コミット・モードは、作業論理単位がデータベースに複数の更新を必要とする状況では、安全性に欠けます。自動コミット・モードを使用した場合、1 つの更新が行われてから他の更新が行われるまでの間にアプリケーションやシステムで何らかの問題が発生すると、最初の更新は元に戻せなくなります。

自動コミット・モードでは、変更がすぐに永続的にコミットされるため、アプリケーションでは commit メソッドや rollback メソッドを呼び出す必要がありません。このため、アプリケーションの作成は容易になります。

自動コミット・モードの使用可能化/使用不可能化は、接続が存在している間に動的に行うことができます。自動コミットは、次のようにして使用可能にされます (データ・ソースがすでに存在していると想定した場合)。

```
Connection connection = dataSource.getConnection();  
  
Connection.setAutoCommit(false); // Disables auto-commit.
```

トランザクションの途中で自動コミットの設定が変更されると、保留中の作業はすべて自動的にコミットされます。分散トランザクションの一部となっている接続に対して自動コミットが使用可能にされると、SQLException が生成されます。

#### トランザクション分離レベル:



トランザクション分離レベルは、トランザクション内でステートメントが認識できるデータを指定します。これらのレベルは、同じターゲット・データ・ソースに対するトランザクション間で可能な相互作用を定義することにより、同時に行われるアクセスのレベルに直接影響します。

## データベース異常

データベース異常とは、単一のトランザクションの視点から見ると誤っているように見え、すべてのトランザクションの視点から見ると正しく見える、生成された結果のことを言います。データベース異常の各タイプは、次のように説明できます。

- **ダーティー読み取り**は、次の場合に行われます。
  1. トランザクション A がテーブルに行を挿入する。
  2. トランザクション B が新しい行を読み取る。
  3. トランザクション A がロールバックする。

トランザクション B は、トランザクション A によって挿入された行に基づいてシステムに対する作業を完了できますが、この行は、永続的なデータベースの一部にはなりません。

- **繰り返し不可の読み取り**は、次の場合に行われます。
  1. トランザクション A が行を読み取る。
  2. トランザクション B が行を変更する。
  3. トランザクション A は、2 度目に同じ行を読み取るが、最初とは違う新しい結果を得る。
- **ファントム読み取り**は、次の場合に行われます。
  1. トランザクション A が SQL 照会で WHERE 文節を満たすすべての行を読み取る。
  2. トランザクション B が WHERE 文節を満たす別の行を挿入する。
  3. トランザクション A が WHERE 条件を再評価すると、追加された行が検出される。

注: DB2 for iSeries は、ロック・ストラテジーにより規定されているレベルでの許容データベース異常に対して、常にアプリケーションをさらしているわけではありません。

## JDBC トランザクション分離レベル

IBM Developer Kit for Java の JDBC API には、5 つのトランザクション分離レベルがあります。最小のものから最大のをリストすると、次のようになります。

### JDBC\_TRANSACTION\_NONE

これは、JDBC ドライバーがトランザクションをサポートしないことを示す特殊な定数です。

### JDBC\_TRANSACTION\_READ\_UNCOMMITTED

このレベルでは、トランザクションは、データに対するコミットされていない変更を認識できません。データベース異常が起こるのは、すべてこのレベルです。

### JDBC\_TRANSACTION\_READ\_COMMITTED

このレベルは、トランザクション内で行われる一切の変更が、トランザクションがコミットされるまで外から認識されないことを意味します。これにより、ダーティー読み取りが行われる可能性はなくなります。

### JDBC\_TRANSACTION\_REPEATABLE\_READ

このレベルは、読み取られる行がロックしたまま保持されることにより、トランザクションが完了するまで他のトランザクションが行を変更できなくなることを意味します。これにより、ダーティー読み取りや繰り返し不可の読み取りは行えなくなります。ファントム読み取りは可能です。

## JDBC\_TRANSACTION\_SERIALIZABLE

テーブルはトランザクション用にロックされ、テーブルに値を追加したりテーブルから値を除去したりする他のトランザクションによって **WHERE** 条件が変更されなくなります。これにより、すべてのタイプのデータベース異常が起こらなくなります。

接続のトランザクション分離レベルを変更するには、`setTransactionIsolation` メソッドを使用できます。

### 考慮事項

前述の 5 つのトランザクション・レベルについては、これが JDBC 仕様で定義されていると誤解される場合がよくあります。一般に、`TRANSACTION_NONE` の値は、コミットメント制御なしで実行する概念を表していると考えられています。JDBC 仕様は、これと同じ方法で `TRANSACTION_NONE` を定義するものではありません。`TRANSACTION_NONE` は、JDBC 仕様において、ドライバーがトランザクションをサポートしないレベルとして定義されており、JDBC 互換のドライバーであるわけではありません。

`getTransactionIsolation` メソッドが呼び出されたときに `NONE` というレベルが報告されることは決してありません。

問題が少し複雑になっているのは、JDBC ドライバーのデフォルト・トランザクション分離レベルがインプリメンテーションで定義されるためです。ネイティブ JDBC ドライバーのデフォルトのトランザクション分離レベルは、`NONE` です。このレベルでは、ジャーナルがないファイルでもドライバーで処理することができ、QGPL ライブラリーのファイルのような指定を一切作成する必要がありません。

ネイティブ JDBC ドライバーでは、`setTransactionIsolation` メソッドに `JDBC_TRANSACTION_NONE` を渡したり、接続のプロパティに `NONE` を指定することが可能です。とはいえ、`getTransactionIsolation` メソッドは、値が `NONE` であると常に `JDBC_TRANSACTION_READ_UNCOMMITTED` を報告します。アプリケーションで、どのレベルで実行しているかをトラッキングし続ける必要がある場合、そのトラッキングは、アプリケーションの責任で行われます。

過去のリリースでは、システムに本当の意味での自動コミット・モードの概念がなかったため、JDBC ドライバーは、自動コミットに真が指定されるとトランザクション分離レベルを `NONE` に変更することにこれを処理していました。これは、機能としては近いものでしたが、すべてのシナリオにおいて正確な結果をもたらす処理ではありませんでした。この処理は行われなくなり、データベースは、トランザクション分離レベルの概念と自動コミットの概念を切り離します。これにより、自動コミットを使用可能にしたまま `JDBC_TRANSACTION_SERIALIZABLE` レベルでシステムを稼働させることが、完全に有効になります。唯一有効でないシナリオは、自動コミット・モードを使用せずに `JDBC_TRANSACTION_NONE` レベルでシステムを実行するシナリオです。トランザクション分離レベルを指定せずにシステムが実行された場合、アプリケーションはコミットの境界を制御することができません。

### JDBC 仕様と iSeries プラットフォームの間のトランザクション分離レベル

iSeries プラットフォームで使用されるトランザクション分離レベルの共通名は、JDBC 仕様で指定されている名前と一致しません。次の表は、iSeries プラットフォームで使用される名前と付き合わせたものですが、これは JDBC 仕様で使用されるものに対応するものではありません。

| JDBC レベル*                                      | iSeries レベル   |
|--|---------------|
| <code>JDBC_TRANSACTION_NONE</code>             | *NONE または *NC |
| <code>JDBC_TRANSACTION_READ_UNCOMMITTED</code> | *CHG または *UR  |
| <code>JDBC_TRANSACTION_READ_COMMITTED</code>   | *CS           |
| <code>JDBC_TRANSACTION_REPEATABLE_READ</code>  | *ALL または *RS  |
| <code>JDBC_TRANSACTION_SERIALIZABLE</code>     | *RR           |

\* この表では、分かりやすくするために JDBC\_TRANSACTION\_NONE の値と iSeries レベル \*NONE および \*NC を並べてあります。これは、仕様と iSeries レベルが直接一致することを示すものではありません。

### 保管ポイント:

保管ポイントを使用して、トランザクション内に「ステージング・ポイント」を設定できます。保管ポイントは、アプリケーションがトランザクション全体を取り消さなくてもロールバックできるチェックポイントです。JDBC 3.0 では保管ポイントが新しくなりました。したがって、アプリケーションが保管ポイントを使用するには、そのアプリケーションを Java Development Kit (JDK) 1.4 または後続のリリース上で実行しなければなりません。さらに、Developer Kit for Java にとっては保管ポイントは新しい機能なので、旧リリースの Developer Kit for Java で JDK 1.4 または後続のリリースを使用していない場合は、これらの保管ポイントはサポートされません。

注: システムには保管ポイントを処理する SQL ステートメントが備えられています。JDBC アプリケーションの中で、これらのステートメントを直接使用しないようにすることをお勧めします。直接使用しても作動しますが、JDBC ドライバーの実行時に保管ポイントを追跡する機能は失われます。最低限、2 つのモデルを混合する (つまり、独自の保管ポイント SQL ステートメントと JDBC API を使用する) ことは避ける必要があります。

## 保管ポイントの設定とロールバック

トランザクションの作業全体のどこにでも、保管ポイントを設定できます。その後、誤りが発生した場合に、アプリケーションはこれらのいずれかの保管ポイントにロールバックし、そのポイントから処理を続行することができます。以下の例では、アプリケーションはデータベース・テーブル中に値 FIRST を挿入します。その後、保管ポイントが設定され、別の値 SECOND がデータベース中に挿入されます。保管ポイントへのロールバックが発行されると、SECOND の挿入作業は取り消されますが、FIRST は保留トランザクションの一部として残ります。最後に、値 THIRD が挿入され、トランザクションがコミットされます。データベース・テーブルには、値 FIRST と THIRD が含まれます。

### 例: 保管ポイントの設定とロールバック

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
Statement s = Connection.createStatement();
s.executeUpdate("insert into table1 values ('FIRST')");
Savepoint pt1 = connection.setSavepoint("FIRST SAVEPOINT");
s.executeUpdate("insert into table1 values ('SECOND')");
connection.rollback(pt1);          // Undoes most recent insert.
s.executeUpdate("insert into table1 values ('THIRD')");
connection.commit();
```

自動コミット・モードの場合に、保管ポイントの設定時に問題が起きることは考えられませんが、トランザクションの終了時に保管ポイントの存続期間が終了するとロールバックできません。

## 保管ポイントの解放

Connection オブジェクト上に releaseSavepoint メソッドを指定したアプリケーションにより、保管ポイントを解放できます。保管ポイントの解放後にロールバックを試行すると例外が生じます。トランザクションのコミット時やロールバック時に、保管ポイントはすべて解放されます。特定の保管ポイントがロールバックされた時点にも、後続の他の保管ポイントは解放されます。



## 分散トランザクション

通常、Java Database Connectivity (JDBC) のトランザクションはローカルです。これは、単一の接続がトランザクションのすべての作業を行い、その接続では一度に 1 つのトランザクションだけが動作できることを意味します。このトランザクションのすべての動作が完了するか、失敗すると、永続化するためにコミットまたはロールバックが呼び出され、新しいトランザクションが開始されます。ただしこれは、ローカル・トランザクション以上の機能を提供する Java で使用可能な、トランザクションの拡張サポートです。このサポートの完全な仕様は、「Java Transaction API」を参照してください。

Java Transaction API (JTA) は、複雑なトランザクションをサポートします。また、Connection オブジェクトからのトランザクションの分離もサポートします。JDBC は ODBC、および X/Open Call Level Interface (CLI) 仕様を、また JTA は X/Open Extended Architecture (XA) 仕様をモデルにしています。JTA と JDBC は共に動作して、Connection オブジェクトからトランザクションを分離します。Connection オブジェクトからトランザクションを分離することにより、同時に複数のトランザクション上で単一の接続を動作させることができるようになります。逆に、単一のトランザクションで複数の接続を機能させることもできます。

Java Transaction API (JTA) 1.0.1 仕様 (英語)

注: JTA を使用する計画の場合は、JDBC の入門にある、拡張クラスパス内に必要な JAR ファイルに関する情報を参照してください。JDBC 2.0 のオプション・パッケージと JTA JAR ファイルの両方が必要です (JDK 1.4 またはそれ以降のバージョンを実行している場合は、これらのファイルが JDK によって自動的に検索されます)。デフォルトでは検索されません。

## JTA を使ったトランザクション

JTA と JDBC を同時に使用するときは、これらの間に、トランザクションの作業を完遂するための一連のステップがあります。XA のサポートは、XADataSource クラスを通して提供されます。このクラスは、ConnectionPoolDataSource スーパークラスとまったく同じ方法による接続プーリングの設定のためのサポートが含まれています。

XADataSource インスタンスを使うと、XAConnection オブジェクトを取得できます。XAConnection オブジェクトは、JDBC 接続と XAResource オブジェクトの両方のためのコンテナを提供します。XAResource オブジェクトは、XA トランザクション・サポートを処理するために設計されました。XAResource は、オブジェクトを経由したトランザクションを、トランザクション ID (XID) によって処理します。

XID は、必ずインプリメントする必要のあるインターフェースです。これは、X/Open トランザクション ID の XID 構造の Java マッピングに相当します。このオブジェクトには、次の 3 つの部分が含まれます。

- グローバル・トランザクションのフォーマット ID
- グローバル・トランザクション ID
- ブランチ修飾子

このインターフェースの完全な詳細については、JTA 仕様を参照してください。

例: JTA を使ってトランザクションを処理するでは、アプリケーション内で JTA を使ってトランザクションを処理する方法が示されています。

## プーリングおよび分散トランザクション用の UDBXADataSource サポートを使用する

Java Transaction API サポートは、接続プーリングの直接サポートを提供しています。UDBXADataSource は ConnectionPoolDataSource の拡張で、プールされた XAConnection オブジェクトにアプリケーションがアクセスすることを可能にします。UDBXADataSource は ConnectionPoolDataSource なので、UDBXADataSource の構成および使用方法は、オブジェクト・プーリング用の DataSource サポートの使用で説明されている方法と同一です。

### XADataSource プロパティ

ConnectionPoolDataSource によって提供されているプロパティに加えて、XADataSource インターフェースは次のようなプロパティを提供しています。

| Set メソッド (データ・タイプ)          | 値           | 説明  |
|-----------------------------|-------------|---|
| setLockTimeout (int)        | 0 または任意の正の値 | トランザクション・レベルの有効なロック・タイムアウト (秒) で、任意の正の値です。<br><br>ロック・タイムアウトを 0 にするのは、たとえ他のレベル (ジョブまたはテーブル) で強制される値がある場合でも、トランザクション・レベルで強制されるロック・タイムアウトが無いことを意味します。<br><br>デフォルト値は 0 です。  |
| setTransactionTimeout (int) | 0 または任意の正の値 | 有効なトランザクション・タイムアウト (秒) で、任意の正の値です。<br><br>トランザクション・タイムアウトが 0 に指定されるのは、強制されるトランザクション・タイムアウト値がないことを意味します。トランザクションがタイムアウト値よりも長時間、活動状態になった場合は、ロールバックのみとしてマークされ、このトランザクション下の以降の操作の試行では例外が発生します。<br><br>デフォルト値は 0 です。 |

### ResultSets およびトランザクション

前述の例でも示されていたように、トランザクションの開始と終了は区別されているため、トランザクションをしばらく中断し、後で再開することができます。これにより、トランザクションの間に作成された ResultSet リソースのたくさんのシナリオが提供されます。

#### 単純なトランザクションの終了

トランザクションを終了すると、トランザクションの中で作成され、開かれているすべての ResultSet は自動的にクローズされます。最大の並列処理を行うため、ResultSet は使用が完了したときに明示的にクロー

ズすることをお勧めします。しかし、トランザクション中で開かれている `ResultSet` に `XAResource.end` 呼び出しを行った後にアクセスしようとする、結果として例外が発生します。

この動作については、例: トランザクションを終了するを参照してください。

## 中断および再開

トランザクションが中断されている間、トランザクションが活動状態にあるときに作成された `ResultSet` にアクセスすることはできず、例外になります。しかし、トランザクションを再開すると、再び `ResultSet` は使用可能になり、トランザクションが中断される前と同じ状態に戻ります。

この動作については、例: トランザクションの中断と再開を参照してください。

## ResultSet の中断の影響

トランザクションを中断している間は、`ResultSet` にアクセスできません。しかし、動作を実行するため、他のトランザクションの下で `Statement` オブジェクトを再処理することができます。 `JDBC Statement` オブジェクトは同時に 1 つの `ResultSet` しか持つことができず (`JDBC 3.0` の、ストアード・プロシージャ呼び出しからの複数の同時 `ResultSet` のサポートは除きます)、新しいトランザクションからの要求を満たすために、中断されたトランザクションの `ResultSet` はクローズしなければなりません。このことが発生します。

この動作については、例: `ResultSet` の中断を参照してください。

**注:** `JDBC 3.0` では、`Statement` がストアード・プロシージャ呼び出しで同時に複数の `ResultSet` を開くことができますが、これは 1 つの単位として見なされ、`Statement` が新しいトランザクションで再処理されると、すべてクローズされます。単一のステートメントで、同時に活動状態にある 2 つのトランザクションで `ResultSet` を持つことはできません。

## 多重化

`JTA API` は、`JDBC` 接続からトランザクションの分離のために設計されています。この `API` によって、単一のトランザクション上で複数の接続が動作させることも、同時に複数のトランザクション上で単一の接続を動作させることも可能です。これは**多重化**と呼ばれ、`JDBC` 単体では実行できない数多くの複雑なタスクを実行できます。

この例では、単一のトランザクション上で複数の接続を動作させる方法を示しています。

この例では、複数のトランザクションで単一の接続を同時に使用する方法が示されています。

`JTA` を使用するためのさらに詳細な情報は、`JTA` 仕様を参照してください。 `JDBC 3.0` の仕様でも、これらの 2 つのテクノロジーが共に分散トランザクションをサポートする方法についての情報が含まれています。

## 2 フェーズ・コミットおよびトランザクション・ログ

`JTA API` は、分散 2 フェーズ・コミット・プロトコルの役割をアプリケーションに対して完全に外部化します。これまでの例で示されているとおり、`JTA` トランザクション下で `JTA` および `JDBC` を使ってデータベースにアクセスするとき、アプリケーションは変更をコミットするために `XAResource.prepare()` および `XAResource.commit()` メソッド、または単に `XAResource.commit()` メソッドを使用します。

さらに、単一のトランザクションを使用して複数の個別のデータベースにアクセスするとき、これらのデータベースにあるトランザクションの非分離性に必要とされる、2 フェーズ・コミット・プロトコル、および

すべての関連するロギングを保証するのは、アプリケーションの責任です。一般的には、複数のデータベースにまたがる 2 フェーズ・コミットの処理 (つまり、XAResource)、およびロギングは、アプリケーション・サーバーまたはトランザクション・モニターの制御下で実行されます。これは、アプリケーションそのものが、実際にこれらの問題に関係しないためです。

たとえば、アプリケーションがいくつかの commit() メソッドを呼び出し、エラーなしで処理が戻されたとします。その後、基礎となるアプリケーション・サーバーまたはトランザクション・モニターは、単一の分散トランザクションに加わっている各データベース (XAResource) の処理を開始します。

アプリケーション・サーバーは 2 フェーズ・コミット処理の間は、広範囲のロギングを使用します。XAResource.prepare() メソッドがそれぞれの参加データベース (XAResource) で順番に呼び出され、続いて XAResource.commit() メソッドがそれぞれの参加データベース (XAResource) で呼び出されます。

この処理中に障害が発生した場合、アプリケーション・サーバーのトランザクション・モニターのログにより、アプリケーション・サーバーそのものが後で分散トランザクションを回復するために JTA API を使用することができるようにします。アプリケーション・サーバーまたはトランザクション・モニターの制御下で行われる回復は、アプリケーション・サーバーがそれぞれの参加データベース (XAResource) の既知の状態へのトランザクションを取得できるようにします。これで、すべての参加データベースをまたいだ分散トランザクション全体の既知の状態を確実に得られます。

#### 例: トランザクションを処理するために JTA を使用する:

以下は、アプリケーション内でトランザクションを処理するための Java Transaction API (JTA) の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTACommit {

    public static void main(java.lang.String[] args) {
        JTACommit test = new JTACommit();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
```

```

        // Ignore... does not exist
    }

    s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
    s.close();
} finally {
    if (c != null) {
        c.close();
    }
}
}

/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADatasource.
        UDBXADatasource ds = (UDBXADatasource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with the
        // JDBC driver. See Transactions with JTA for a description of
        // this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Standard JDBC work is performed.
        int count =
            stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is pretty fun.')");

        // When the transaction work has completed, the XA resource must
        // again be notified.
        xaRes.end(xid, XAResource.TMSUCCESS);

        // The transaction represented by the transaction ID is prepared
        // to be committed.
        int rc = xaRes.prepare(xid);

        // The transaction is committed through the XAResource.
        // The JDBC Connection object is not used to commit
        // the transaction when using JTA.
        xaRes.commit(xid, false);

    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {

```

```

        try {
            if (c != null)
                c.close();
        } catch (SQLException e) {
            System.out.println("Note: Cleanup exception.");
            e.printStackTrace();
        }
    }
}
}

```

#### 例: 単一トランザクション上で動作する複数の接続:

以下は、単一トランザクション上で動作する複数の接続の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;
public class JTAMultiConn {
    public static void main(java.lang.String[] args) {
        JTAMultiConn test = new JTAMultiConn();
        test.setup();
        test.run();
    }
}
/**
 * Handle the previous cleanup run so that this test can recommence.
 */
public void setup() {
    Connection c = null;
    Statement s = null;
    try {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        c = DriverManager.getConnection("jdbc:db2:*local");
        s = c.createStatement();
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
        }
        catch (SQLException e) {
            // Ignore... does not exist
        }
        s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
            (50))");
        s.close();
    }
    finally {
        if (c != null) {
            c.close();
        }
    }
}
/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c1 = null;
    Connection c2 = null;
    Connection c3 = null;
    try {
        Context ctx = new InitialContext();
        // Assume the data source is backed by a UDBXDataSource.

```

```

    UDBXADataSource ds = (UDBXADataSource)
        ctx.lookup("XADataSource");
    // From the DataSource, obtain some XAConnection objects that
    // contain an XAResource and a Connection object.
    XAConnection xaConn1 = ds.getXAConnection();
    XAConnection xaConn2 = ds.getXAConnection();
    XAConnection xaConn3 = ds.getXAConnection();
    XAResource xaRes1 = xaConn1.getXAResource();
    XAResource xaRes2 = xaConn2.getXAResource();
    XAResource xaRes3 = xaConn3.getXAResource();
    c1 = xaConn1.getConnection();
    c2 = xaConn2.getConnection();
    c3 = xaConn3.getConnection();
    Statement stmt1 = c1.createStatement();
    Statement stmt2 = c2.createStatement();
    Statement stmt3 = c3.createStatement();
    // For XA transactions, a transaction identifier is required.
    // Support for creating XIDs is again left to the application
    // program.
    Xid xid = JDXATest.xidFactory();
    // Perform some transactional work under each of the three
    // connections that have been created.
    xaRes1.start(xid, XAResource.TMNOFLAGS);
    int count1 = stmt1.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-A')");
    xaRes1.end(xid, XAResource.TMNOFLAGS);

    xaRes2.start(xid, XAResource.TMJOIN);
    int count2 = stmt2.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-B')");
    xaRes2.end(xid, XAResource.TMNOFLAGS);

    xaRes3.start(xid, XAResource.TMJOIN);
    int count3 = stmt3.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-C')");
    xaRes3.end(xid, XAResource.TMSUCCESS);
    // When completed, commit the transaction as a single unit.
    // A prepare() and commit() or 1 phase commit() is required for
    // each separate database (XAResource) that participated in the
    // transaction. Since the resources accessed (xaRes1, xaRes2, and xaRes3)
    // all refer to the same database, only one prepare or commit is required.
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);
}
catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
}
finally {
    try {
        if (c1 != null) {
            c1.close();
        }
    }
    catch (SQLException e) {
        System.out.println("Note: Cleanup exception " +
            e.getMessage());
    }
    try {
        if (c2 != null) {
            c2.close();
        }
    }
    catch (SQLException e) {
        System.out.println("Note: Cleanup exception " +
            e.getMessage());
    }
    try {
        if (c3 != null) {
            c3.close();
        }
    }
}

```



```

    }
  }
  catch (SQLException e) {
    System.out.println("Note: Cleanup exception " +
      e.getMessage());
  }
}
}
}

```

**例: 複数のトランザクションで単一の接続を使用する:**

以下は、複数のトランザクションでの単一接続の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTAMultiTx {

    public static void main(java.lang.String[] args) {
        JTAMultiTx test = new JTAMultiTx();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... does not exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }

    /**
     * This test uses JTA support to handle transactions.
     */

```

```

public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();
        Statement stmt = c.createStatement();

        // For XA transactions, a transaction identifier is required.
        // This is not meant to imply that all the XIDs are the same.
        // Each XID must be unique to distinguish the various transactions
        // that occur.
        // Support for creating XIDs is again left to the application
        // program.
        Xid xid1 = JDXTTest.xidFactory();
        Xid xid2 = JDXTTest.xidFactory();
        Xid xid3 = JDXTTest.xidFactory();

        // Do work under three transactions for this connection.
        xaRes.start(xid1, XAResource.TMNOFLAGS);
        int count1 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-A')");
        xaRes.end(xid1, XAResource.TMNOFLAGS);

        xaRes.start(xid2, XAResource.TMNOFLAGS);
        int count2 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-B')");
        xaRes.end(xid2, XAResource.TMNOFLAGS);

        xaRes.start(xid3, XAResource.TMNOFLAGS);
        int count3 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-C')");
        xaRes.end(xid3, XAResource.TMNOFLAGS);

        // Prepare all the transactions
        int rc1 = xaRes.prepare(xid1);
        int rc2 = xaRes.prepare(xid2);
        int rc3 = xaRes.prepare(xid3);

        // Two of the transactions commit and one rolls back.
        // The attempt to insert the second value into the table is
        // not committed.
        xaRes.commit(xid1, false);
        xaRes.rollback(xid2);
        xaRes.commit(xid3, false);

    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {
        try {
            if (c != null)
                c.close();
        } catch (SQLException e) {
            System.out.println("Note: Cleanup exception.");
            e.printStackTrace();
        }
    }
}
}

```

## 例: 中断状態の ResultSets:

以下は、作業を実行するために Statement オブジェクトを別のトランザクションで再処理する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEffect {

    public static void main(java.lang.String[] args) {
        JTATxEffect test = new JTATxEffect();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... does not exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.')");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }

    /**
     * This test uses JTA support to handle transactions.
     */
    public void run() {
        Connection c = null;

        try {
            Context ctx = new InitialContext();

            // Assume the data source is backed by a UDBXADataSource.
```

```

    UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADataSource");

    // From the DataSource, obtain an XAConnection object that
    // contains an XAResource and a Connection object.
    XAConnection xaConn = ds.getXAConnection();
    XAResource xaRes = xaConn.getXAResource();
    Connection c = xaConn.getConnection();

    // For XA transactions, a transaction identifier is required.
    // An implementation of the XID interface is not included with
    // the JDBC driver. See Transactions with JTA
    // for a description of this interface to build a
    // class for it.
    Xid xid = new XidImpl();

    // The connection from the XAResource can be used as any other
    // JDBC connection.
    Statement stmt = c.createStatement();

    // The XA resource must be notified before starting any
    // transactional work.
    xaRes.start(xid, XAResource.TMNOFLAGS);

    // Create a ResultSet during JDBC processing and fetch a row.
    ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
    rs.next();

    // The end method is called with the suspend option. The
    // ResultSets associated with the current transaction are 'on hold'.
    // They are neither gone nor accessible in this state.
    xaRes.end(xid, XAResource.TMSUSPEND);

    // In the meantime, other work can be done outside the transaction.
    // The ResultSets under the transaction can be closed if the
    // Statement object used to create them is reused.
    ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
    while (nonXARS.next()) {
        // Process here...
    }

    // Attempt to go back to the suspended transaction. The suspended
    // transaction's ResultSet has disappeared because the statement
    // has been processed again.
    xaRes.start(newXid, XAResource.TMRESUME);
    try {
        rs.next();
    } catch (SQLException ex) {
        System.out.println("This exception is expected. " +
            "The ResultSet closed due to another process.");
    }

    // When the transaction had completed, end it
    // and commit any work under it.
    xaRes.end(xid, XAResource.TMNOFLAGS);
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);

} catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
} finally {
    try {
        if (c != null)

```

```

        c.close();
    } catch (SQLException e) {
        System.out.println("Note: Cleaup exception.");
        e.printStackTrace();
    }
}
}
}
}
}

```

### 例: トランザクションを終了する:

以下は、アプリケーション内でトランザクションを終了する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEnd {

    public static void main(java.lang.String[] args) {
        JTATxEnd test = new JTATxEnd();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... does not exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.')");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }

    /**
     * This test use JTA support to handle transactions.

```

```

*/
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADDataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();

        // For XA transactions, transaction identifier is required.
        // An implementation of the XID interface is not included
        // with the JDBC driver. See Transactions with JTA for a
        // description of this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Create a ResultSet during JDBC processing and fetch a row.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // When the end method is called, all ResultSet cursors close.
        // Accessing the ResultSet after this point results in an
        // exception being thrown.
        xaRes.end(xid, XAResource.TMNOFLAGS);

        try {
            String value = rs.getString(1);
            System.out.println("Something failed if you receive this message.");
        } catch (SQLException e) {
            System.out.println("The expected exception was thrown.");
        }

        // Commit the transaction to ensure that all locks are
        // released.
        int rc = xaRes.prepare(xid);
        xaRes.commit(xid, false);

    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {
        try {
            try {
                if (c != null)
                    c.close();
            } catch (SQLException e) {
                System.out.println("Note: Cleanup exception.");
                e.printStackTrace();
            }
        }
    }
}
}

```

## リンク集

コードのサンプルに関する特記事項

### JTA を使ったトランザクション

通常、Java Database Connectivity (JDBC) のトランザクションはローカルです。これは、単一の接続がトランザクションのすべての作業を行い、その接続では一度に 1 つのトランザクションだけが動作できることを意味します。このトランザクションのすべての動作が完了するか、失敗すると、永続化するためにコミットまたはロールバックが呼び出され、新しいトランザクションが開始されます。ただしこれは、ローカル・トランザクション以上の機能を提供する Java で使用可能な、トランザクションの拡張サポートです。このサポートの完全な仕様は、「Java Transaction API」を参照してください。

### 例: トランザクションを中断して再開する:

以下は、中断され、その後再開されるトランザクションの例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxSuspend {

    public static void main(java.lang.String[] args) {
        JTATxSuspend test = new JTATxSuspend();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... doesn't exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.')");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }
}
```



```

}

/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADDataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with
        // the JDBC driver. Transactions with JTA for a
        // description of this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Create a ResultSet during JDBC processing and fetch a row.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // The end method is called with the suspend option. The
        // ResultSets associated with the current transaction are 'on hold'.
        // They are neither gone nor accessible in this state.
        xaRes.end(xid, XAResource.TMSUSPEND);

        // Other work can be performed with the transaction.
        // As an example, you can create a statement and process a query.
        // This work and any other transactional work that the transaction may
        // perform is separate from the work done previously under the XID.
        Statement nonXASstmt = conn.createStatement();
        ResultSet nonXARS = nonXASstmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Process here...
        }
        nonXARS.close();
        nonXASstmt.close();

        // If an attempt is made to use any suspended transactions
        // resources, an exception results.
        try {
            rs.getString(1);
            System.out.println("Value of the first row is " + rs.getString(1));
        } catch (SQLException e) {
            System.out.println("This was an expected exception - " +
                "suspended ResultSet was used.");
        }
    }
}

```

```

// Resume the suspended transaction and complete the work on it.
// The ResultSet is exactly as it was before the suspension.
xaRes.start(newXid, XAResource.TMRESUME);
rs.next();
System.out.println("Value of the second row is " + rs.getString(1));

// When the transaction has completed, end it
// and commit any work under it.
xaRes.end(xid, XAResource.TMNOFLAGS);
int rc = xaRes.prepare(xid);
xaRes.commit(xid, false);

} catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
} finally {
    try {
        if (c != null)
            c.close();
    } catch (SQLException e) {
        System.out.println("Note: Cleanup exception.");
        e.printStackTrace();
    }
}
}
}
}

```

## ステートメントのタイプ

**Statement** インターフェースとその **PreparedStatement** および **CallableStatement** サブクラスは、データベースに対する構造化照会言語 (SQL) コマンドの処理に使用されます。SQL ステートメントが処理されると、**ResultSet** オブジェクトが生成されます。

**Statement** インターフェースのサブクラスは、**Connection** インターフェース上のいくつかのメソッドを使用して作成されます。1 つの **Connection** オブジェクトには、そのオブジェクトの下で同時に作成された多数の **Statement** オブジェクトを含めることができます。過去のリリースでは、作成できる **Statement** オブジェクトの正確な数を示すことができましたが、このリリースではそれはできません。というのは、様々なタイプの **Statement** オブジェクトがあり、そのタイプによって、データベース・エンジンに持ち込む「ハンドル」の数が異なるからです。したがって、1 回の接続でアクティブにできるステートメントの数は、使用する **Statement** オブジェクトのタイプによって異なります。

アプリケーションは、**Statement.close** メソッドを呼び出して、ステートメントの処理が終了したことを示します。すべての **Statement** オブジェクトは、作成元の接続がクローズされたときにクローズされます。ただし、**Statement** オブジェクトをクローズするときに、この動作に 100% の信頼を置かないようにしてください。たとえば、アプリケーションが変更され、接続を明示的にクローズする代わりに接続プールが使用される場合は、接続がクローズされないため、アプリケーションからステートメント・ハンドルが「リーク」します。必要なくなった時点で **Statement** オブジェクトをすぐにクローズすれば、そのステートメントが使用している外部データベース・リソースをすぐに解放できます。

ネイティブの JDBC ドライバーは、リークしているステートメントを検出し、それを代わりに処理しようとします。ただし、このサポートに頼ると、パフォーマンスは低くなります。

各インターフェースは、**CallableStatement** が **PreparedStatement** を拡張し、**PreparedStatement** が **Statement** を拡張する、というように階層状の継承関係にあるため、各インターフェースの機能は、そのインターフェースを拡張するクラスで使用できます。たとえば、**Statement** クラスの機能は、**PreparedStatement** クラスや

CallableStatement クラスでもサポートされます。ただし、これに対する主な例外として、Statement クラスの executeQuery、executeUpdate、および execute メソッドがあります。これらのメソッドは、動的な処理を行うための SQL ステートメントを扱うので、PreparedStatement オブジェクトや CallableStatement オブジェクトでこれらのメソッドを使用すると、例外が発生します。

### Statements:

Statement オブジェクトは、静的 SQL ステートメントの処理と、それによって生成される結果の取得に使用されます。一度にオープンできるのは、各 Statement オブジェクトにつき 1 つの ResultSet だけです。SQL ステートメントを処理するすべてのステートメント・メソッドは、すでにオープンされている ResultSet があると、暗黙的にステートメントの現行の ResultSet をクローズします。

## ステートメントの作成

Statement オブジェクトは、createStatement メソッドを使用して Connection オブジェクトから作成されます。たとえば、conn という Connection オブジェクトがすでに存在していた場合、データベースに SQL ステートメントを渡すための Statement オブジェクトは、次のようなコードの行で作成されます。

```
Statement stmt = conn.createStatement();
```

## ResultSet 特性の指定

ResultSet の特性は、最終的にその ResultSet を作成するステートメントに関連付けられています。これらの ResultSet の特性は、Connection.createStatement メソッドで指定できます。以下は、createStatement メソッドに対する有効な呼び出しの例を示しています。

例: createStatement メソッド

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// The following is new in JDBC 2.0

Statement stmt2 = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATEABLE);

// The following is new in JDBC 3.0

Statement stmt3 = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY, ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

これらの特性に関する詳細は、ResultSetを参照してください。

## ステートメントの処理

Statement オブジェクトを使用した SQL ステートメントの処理は、executeQuery()、executeUpdate()、および execute() メソッドで行われます。

## SQL 照会からの戻り結果

ResultSet を戻す SQL 照会ステートメントを処理する場合は、executeQuery() メソッドを使用します。Statement オブジェクトの executeQuery メソッドを使用して ResultSet を取得する サンプル・プログラムを参照してください。

注: executeQuery で処理される SQL ステートメントが ResultSet を戻さない場合は、SQLException がスローされます。

## SQL ステートメントの更新カウンターの戻り

SQL が、更新カウンターを戻すデータ定義言語 (DDL) ステートメントまたはデータ操作言語 (DML) ステートメントであると分かっている場合は、`executeUpdate()` メソッドを使用します。 `StatementExample` プログラムは、`Statement` オブジェクトの `executeUpdate` メソッドを使用します。

## 何が戻されるか分からない SQL ステートメントの処理

SQL ステートメントのタイプが不明な場合、`execute` メソッドを使用します。このメソッドが一度処理されると、JDBC ドライバーはアプリケーションに、API 呼び出しを通して SQL ステートメントが生成した結果のタイプを通知することができます。 `execute` メソッドは、結果が少なくとも 1 つの `ResultSet` である場合は `true` を、戻り値が更新カウンターである場合は `false` を戻します。この情報を得た後、アプリケーションは `statement` メソッドの `getUpdateCount` または `getResultSet` を使用して、SQL ステートメントの処理から戻り値を取り出すことができます。 `StatementExecute` プログラムは、`Statement` オブジェクトで `execute` メソッドを使用します。このプログラムは、パラメーターとして SQL ステートメントが渡されることを期待します。プログラムは、渡された SQL のテキストを確認しなくても、ステートメントを処理することによって、何についての情報が処理されたのかを判別します。

注: 結果が `ResultSet` の場合に `getUpdateCount` メソッドを呼び出すと、-1 が戻されます。結果が更新カウンターの場合に `getResultSet` メソッドを呼び出すと、ヌルが戻されます。

## cancel メソッド

ネイティブの JDBC ドライバーのメソッドは、同じオブジェクトに対して 2 つのスレッドが実行されてオブジェクトが壊れないよう、同期されます。ただし、`cancel` メソッドは例外です。 `cancel` メソッドは、同じオブジェクトの別のスレッドで長時間実行されている SQL ステートメントを停止させるのに使用できます。ネイティブの JDBC ドライバーでは、実行していたタスクすべてを停止するように要求することしかできず、強制的にスレッドに作業を停止させることはできません。この理由で、JDBC ドライバーでは、キャンセルされたステートメントを停止させるのにも時間がかかります。 `cancel` メソッドは、システム上のランナウェイ SQL 照会を停止させるのに使用できます。

### 例: `Statement` オブジェクトの `executeUpdate` メソッドを使用する:

次に、`Statement` オブジェクトの `executeUpdate` メソッドを使用する方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import java.util.Properties;

public class StatementExample {

    public static void main(java.lang.String[] args)
    {

        // Suggestion: Load these from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Register the native JDBC driver. If the driver cannot be
        // registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
        }
    }
}
```

```

    System.exit(1);
}

Connection c = null;
Statement s = null;

try {
    // Create the connection properties.
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Connect to the local iSeries database.
    c = DriverManager.getConnection(URL, properties);

    // Create a Statement object.
    s = c.createStatement();
    // Delete the test table if it exists. Note: This
    // example assumes that the collection MYLIBRARY
    // exists on the system.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Just continue... the table probably does not exist.
    }

    // Run an SQL statement that creates a table in the database.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

    // Run some SQL statements that insert records into the table.
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('RICH', 123)");
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('FRED', 456)");
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('MARK', 789)");

    // Run an SQL query on the table.
    ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

    // Display all the data in the table.
    while (rs.next()) {
        System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
    }

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (s != null) {
            s.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }

    try {
        if (c != null) {
            c.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Connection.");
    }
}
}
}

```

## PreparedStatement:

PreparedStatement は Statement インターフェースを拡張し、SQL ステートメントへのパラメーターの追加をサポートします。

データベースに渡される SQL ステートメントは、結果を戻すまでに 2 段階のプロセスを通ります。これらはまず準備され、次いで処理されます。Statement オブジェクトの場合、これらの 2 つのフェーズは、アプリケーションには 1 つのフェーズとして映ります。PreparedStatement では、この 2 つのステップは分離が可能です。準備ステップは、オブジェクトが作成される時に発生し、処理ステップは PreparedStatement オブジェクトに対して executeQuery、executeUpdate、または execute メソッドが呼び出される時に発生します。

SQL 処理を個々のフェーズに分割できても、パラメーター・マーカーが追加されなければ、それは無意味です。アプリケーションにパラメーター・マーカーが置かれることによって、アプリケーションは、準備時には特定の値を持たないこと、しかし、処理の前には値を指定していることをデータベースに知らせることができます。パラメーター・マーカーは SQL ステートメントでは疑問符で表示されます。

パラメーター・マーカーを使用すれば、汎用 SQL ステートメントを作成し、それを特定の要求に合わせて使用することができます。例として、以下の SQL 照会ステートメントについて取り上げてみます。

```
SELECT * FROM EMPLOYEE_TABLE WHERE LASTNAME = 'DETTINGER'
```

これは、ただ 1 つの値、つまり Dettinger という名前の従業員についての情報を戻す特定の SQL ステートメントです。このステートメントは、以下のようなパラメーター・マーカーを追加することによって、より柔軟なものにすることができます。

```
SELECT * FROM EMPLOYEE_TABLE WHERE LASTNAME = ?
```

単純に値にパラメーター・マーカーを設定することによって、テーブル内のどの従業員についての情報でも取得することができます。

上の Statement の例は、準備フェーズを一度だけ通過すれば、パラメーターに異なる値を指定して繰り返し処理できるので、PreparedStatement によって Statement のパフォーマンスは大幅に向上しています。

注: PreparedStatement の使用は、ネイティブ JDBC ドライバーの Statement プーリングをサポートするための要件です。

PreparedStatement の作成、ResultSet 特性の指定、自動生成キーの処理、およびパラメーター・マーカーの設定を含め、PreparedStatement については、以下のページを参照してください。

## PreparedStatement の作成と使用:

新しい PreparedStatement オブジェクトを作成するには、prepareStatement メソッドを使用します。createStatement メソッドとは違って、SQL ステートメントは PreparedStatement オブジェクトの作成時に指定する必要があります。その時点で SQL ステートメントは使用のためにプリコンパイルされます。

たとえば、conn という名前の Connection オブジェクトがすでに存在しているとするなら、以下の例では PreparedStatement オブジェクトが作成され、データベース内の処理のための SQL ステートメントが準備されます。

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM EMPLOYEE_TABLE  
WHERE LASTNAME = ?");
```



## ResultSet 特性の指定および自動生成キー・サポート

createStatement メソッドと同様に、prepareStatement メソッドは、ResultSet の特性の指定をサポートするよう多重定義されています。さらに prepareStatement メソッドには、自動生成キーを処理するためのバリエーションがあります。以下に、prepareStatement メソッドの有効な呼び出し例をいくつか示します。

例: prepareStatement メソッド

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// New in JDBC 2.0

PreparedStatement ps2 = conn.prepareStatement("SELECT * FROM
EMPLOYEE_TABLE WHERE LASTNAME = ?",

ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATEABLE);

// New in JDBC 3.0

PreparedStatement ps3 = conn.prepareStatement("SELECT * FROM
EMPLOYEE_TABLE WHERE LASTNAME = ?",
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
ResultSet.HOLD_CURSOR_OVER_COMMIT);

PreparedStatement ps4 = conn.prepareStatement("SELECT * FROM
EMPLOYEE_TABLE WHERE LASTNAME = ?", Statement.RETURN_GENERATED_KEYS);
```

## パラメーターの処理

PreparedStatement オブジェクトを処理する前に、各パラメーター・マーカーに何らかの値を設定する必要があります。PreparedStatement オブジェクトには、パラメーターを設定するためのいくつかのメソッドが備わっています。どのメソッドも、set<Type> (<Type> は Java データ・タイプ) という形式です。これらのメソッドの一部の例には、setInt、setLong、setString、setTimestamp、setNull、および setBlob が含まれています。ほとんどすべてのメソッドは 2 つのパラメーターをとります。

- 最初のパラメーターは、ステートメント内のパラメーターの指標です。パラメーター・マーカーには番号が付けられます。番号は 1 から始まります。
- 2 番目のパラメーターは、パラメーターに設定する値です。setBinaryStream の length パラメーターなど、追加パラメーターを持つ set<Type> メソッドもいくつかあります。

詳しくは、java.sql パッケージの Javadoc を参照してください。ps オブジェクト用の上記の例の準備済み SQL ステートメントの場合、処理前のパラメーター値の指定方法は、以下のコードのようになります。

```
ps.setString(1, 'Dettinger');
```

設定されていないパラメーター・マーカーを用いて PreparedStatement を処理しようとすると、SQLException が出されます。

注: パラメーター・マーカーは一度設定されると、以下の状況が発生しない限り、処理の間で同じ値を保持します。

- 別の set メソッドの呼び出しによって値が変更された。
- clearParameters メソッドが呼び出されたときに値が除去された。

clearParameters メソッドは、すべてのパラメーターに設定解除済みのフラグを立てます。clearParameters の呼び出しが行われた後、次の処理の前にすべてのパラメーターに対してもう一度 set メソッドを呼び出す必要があります。



## ParameterMetaData サポート

新しい ParameterMetaData インターフェースでは、パラメーターについての情報を検索することができません。このサポートは、ResultSetMetaData に対する補足で、これと類似しています。精度、位取り、データ・タイプ、および、パラメーターがヌル値を許可するかどうか、などの情報がすべて供給されます。

アプリケーション・プログラムでこの新しいサポートを利用する方法については、例: ParameterMetaData を参照してください。

### 例: ParameterMetaData:

これは、ParameterMetaData インターフェースを使用して、パラメーターについての情報を検索するときの一例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
//
// ParameterMetaData example. This program demonstrates
// the new support of JDBC 3.0 for learning information
// about parameters to a PreparedStatement.
//
// Command syntax:
//   java PMD
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////

import java.sql.*;

public class PMD {

    // Program entry point.
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Obtain setup.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.MYTABLE VALUES(?, ?, ?)");
    }
}
```

```

ParameterMetaData pmd = ps.getParameterMetaData();

for (int i = 1; i < pmd.getParameterCount(); i++) {
    System.out.println("Parameter number " + i);
    System.out.println("  Class name is " + pmd.getParameterClassName(i));
    // Note: Mode relates to input, output or inout
    System.out.println("  Mode is " + pmd.getParameterClassName(i));
    System.out.println("  Type is " + pmd.getParameterType(i));
    System.out.println("  Type name is " + pmd.getParameterTypeName(i));
    System.out.println("  Precision is " + pmd.getPrecision(i));
    System.out.println("  Scale is " + pmd.getScale(i));
    System.out.println("  Nullable? is " + pmd.isNullable(i));
    System.out.println("  Signed? is " + pmd.isSigned(i));
}
}
}

```

### ***PreparedStatement の処理:***

PreparedStatement オブジェクトを含む SQL ステートメントの処理は、Statement オブジェクトの処理と同様に executeQuery、executeUpdate、および execute メソッドによって行われます。しかし、この SQL ステートメントはオブジェクトの作成時に既に指定されているものなので、Statement 用のメソッドとは違って、これらのメソッドにパラメーターは渡されません。PreparedStatement は Statement を拡張するものなので、アプリケーションは SQL ステートメントを取る種類の executeQuery、executeUpdate、および execute メソッドを呼び出そうとする可能性もあります。そうすると、SQLException が出されます。

### **SQL 照会からの戻り結果**

ResultSet オブジェクトを戻す SQL 照会ステートメントを処理する場合は、executeQuery メソッドを使用します。PreparedStatementExample プログラムは、PreparedStatement オブジェクトの executeQuery メソッドを使用して ResultSet を取得します。

注: executeQuery メソッドによって処理された SQL ステートメントが ResultSet を戻さない場合、SQLException が出されます。

### **SQL ステートメントの更新カウン트의戻り**

SQL が、更新カウン트를戻すデータ定義言語 (DDL) ステートメントまたはデータ操作言語 (DML) ステートメントであると分かっている場合は、executeUpdate() メソッドを使用します。PreparedStatementExample サンプル・プログラムは、PreparedStatement オブジェクトの executeUpdate メソッドを使用します。

### **何が戻されるか分からない SQL ステートメントの処理**

SQL ステートメントのタイプが不明な場合、execute メソッドを使用します。一度このメソッドが処理されると、JDBC ドライバーは、SQL ステートメントが API 呼び出しを通して生成した結果のタイプをアプリケーションに通知することができるようになります。execute メソッドは、結果が少なくとも 1 つの ResultSet である場合は true を、戻り値が更新カウントである場合は false を戻します。この情報を得た後、アプリケーションは getUpdateCount または getResultSet ステートメント・メソッドを使用して、SQL ステートメントの処理から戻り値を取り出すことができます。

注: 結果が ResultSet の場合に getUpdateCount メソッドを呼び出すと、-1 が戻されます。結果が更新カウントの場合に getResultSet メソッドを呼び出すと、ヌルが戻されます。

**例: ResultSet を取得するために PreparedStatement を使用する:**

これは、PreparedStatement オブジェクトの executeQuery メソッドを使用して、ResultSet を入手するときの一例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import java.util.Properties;

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
        // Load the following from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL     = "jdbc:db2://*local";

        // Register the native JDBC driver. If the driver cannot
        // be registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        // This program creates a table that is
        // used by prepared statements later.
        try {
            // Create the connection properties.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database.
            c = DriverManager.getConnection(URL, properties);

            // Create a Statement object.
            s = c.createStatement();
            // Delete the test table if it exists. Note that
            // this example assumes throughout that the collection
            // MYLIBRARY exists on the system.
            try {
                s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
            } catch (SQLException e) {
                // Just continue... the table probably did not exist.
            }

            // Run an SQL statement that creates a table in the database.
            s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

        } catch (SQLException sqle) {
            System.out.println("Database processing has failed.");
            System.out.println("Reason: " + sqle.getMessage());
        } finally {
            // Close database resources
            try {
                try {
                    if (s != null) {
                        s.close();
                    }
                } catch (SQLException e) {
                    System.out.println("Cleanup failed to close Statement.");
                }
            }
        }
    }
}
```

```

    }
}

// This program then uses a prepared statement to insert many
// rows into the database.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Create a PreparedStatement object that is used to insert data into the
    // table.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

    for (int i = 0; i < nameArray.length; i++) {
        ps.setString(1, nameArray[i]);    // Set the Name from our array.
        ps.setInt(2, i+1);                // Set the ID.
        ps.executeUpdate();
    }
} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}

// Use a prepared statement to query the database
// table that has been created and return data from it. In
// this example, the parameter used is arbitrarily set to
// 5, meaning return all rows where the ID field is less than
// or equal to 5.
try {
    ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
        "WHERE ID <= ?");

    ps.setInt(1, 5);

    // Run an SQL query on the table.
    ResultSet rs = ps.executeQuery();
    // Display all the data in the table.
    while (rs.next()) {
        System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
    }
} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}

try {

```

```

        if (c != null) {
            c.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Connection.");
    }
}
}
}

```

### CallableStatement:

CallableStatement インターフェースは PreparedStatement を拡張し、パラメーターの出力および入出力のサポートを提供します。CallableStatement インターフェースは、PreparedStatement インターフェースによって提供される入力パラメーターもサポートします。

CallableStatement インターフェースは、SQL ステートメントを使ってストアド・プロシージャを呼び出せるようにします。ストアド・プロシージャは、データベース・インターフェースを持ったプログラムです。このプログラムは、次のような機能を持っています。

- 入力および出力パラメーター、または入出力両方のパラメーターを持つことができます。
- 戻り値を持つことができます。
- 複数の ResultSet を戻すことができます。

JDBC では概念的に、ストアド・プロシージャ呼び出しはデータベースに対する単一の呼び出しですが、ストアド・プロシージャに関連したプログラムは多数のデータベース要求を処理することがあります。さらにストアド・プロシージャ・プログラムは、通常は SQL ステートメントでは実行されないような、プログラマチックな他の多くのタスクを実行するのに使用されることがあります。

CallableStatements は、準備および処理フェーズを分離した PreparedStatement モデルに従っているため、再利用するために最適化することが可能です (詳しくは、PreparedStatement を参照してください)。ストアド・プロシージャの SQL ステートメントがプログラムに結合されると、それらのステートメントは静的 SQL として処理され、さらにパフォーマンスの向上も期待できます。多くのデータベース処理を単一で、再利用可能なデータベース呼び出しにカプセル化することは、ストアド・プロシージャの良い使用例です。この呼び出しは他のシステムまでネットワーク上を通過しますが、多くの作業の要求はリモート・システム上で完了します。

### CallableStatements を作成する

新規 CallableStatement オブジェクトを作成するには、prepareCall メソッドを使用します。

CallableStatement オブジェクトが作成される際には、prepareStatement メソッドの場合と同様、SQL ステートメントが提供される必要があります。SQL ステートメントのプリコンパイルはその時点で行われます。たとえば、conn という名前の Connection オブジェクトが既に存在していると想定した場合に、CallableStatement オブジェクトを作成し、SQL ステートメントを取得する準備フェーズを完了して、データベース内で処理可能な状態にするには、次のようにします。

```
PreparedStatement ps = conn.prepareStatement("? = CALL ADDEMPLOYEE(?, ?, ?);");
```

ADDEMPLOYEE ストアド・プロシージャは入力パラメーターとして、新しい従業員の名前と社会保障番号、およびその従業員のマネージャーのユーザー ID を受け取ります。この情報によって、会社のデータベースの複数のテーブルがその従業員の勤務開始日、部署などの情報によって更新されます。さらに、ストアド・プロシージャは、その従業員の標準ユーザー ID と E メール・アドレスを生成することもで

きるプログラムです。ストアード・プロシージャが初期ユーザー名とパスワード付きの E メールを雇用管理者あてに送信することもできます。その後、雇用管理者はその従業員に対して、それらの情報を知らせることができます。

ADDEMPLOYEE ストアード・プロシージャは戻り値を持つようにセットアップされます。呼び出し側プログラムが、障害が発生した際に利用できるよう、戻りコードとして成功または失敗コードを戻すことができます。戻り値は新しい従業員の会社 ID 番号として定義することもできます。最後に、ストアード・プロシージャ・プログラムは内部で処理される照会を持つことがあり、それらの照会によって開かれた ResultSet を呼び出し側プログラムが利用できるようにしておきます。すべての新しい従業員の情報を照会し、戻された ResultSet を通して呼び出し側プログラムがそれらを利用できるようにするのは妥当なことです。

これらのタスクのそれぞれのタイプを完了する方法については、以下のセクションで扱われています。

## ResultSet 特性の指定および自動生成キー・サポート

CreateStatement および PreparedStatement では、ResultSet 特性を指定するためのサポートを提供する PreparedStatement の複数のバージョンがあります。PreparedStatement とは異なり、PreparedStatement メソッドは CallableStatement からの自動生成キーを処理するバリエーションは提供されていません (JDBC 3.0 では、この概念はサポートされていません)。以下に、PreparedStatement メソッドの正しい呼び出し方法のいくつかの例を示します。

### 例: PreparedStatement メソッド

```
// The following is new in JDBC 2.0
```

```
CallableStatement cs2 = conn.prepareStatement("CALL ADDEMPLOYEE(?, ?, ?)",  
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE);
```

```
// New in JDBC 3.0
```

```
CallableStatement cs3 = conn.prepareStatement("CALL ADDEMPLOYEE(?, ?, ?)",  
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,  
    ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

## パラメーターの処理

前述のとおり、CallableStatement オブジェクトは 3 タイプのパラメーターを取ることができます。

- **IN**

IN パラメーターは PreparedStatement と同じ方法でハンドルされます。PreparedStatement クラスから継承されたさまざまな set メソッドを使って、パラメーターを設定することができます。

- **OUT**

OUT パラメーターは、registerOutParameter メソッドによってハンドルされます。最も一般的な形式の registerOutParameter では、最初のパラメーターとして索引パラメーターが、2 番目のパラメーターとして SQL タイプが取られます。これにより、JDBC ドライバーは、ステートメントが処理されるときにパラメーターがどのようなデータであるかが分かります。registerOutParameter メソッドの他の 2 つのバリエーションは、java.sql パッケージ Javadoc で見つけることができます。

- **INOUT**

INOUT パラメーターは IN パラメーターおよび OUT パラメーターの両方を使って処理が行われる場合に必要です。INOUT パラメーターごとに、ステートメントが処理される前に set メソッドおよび



registerOutParameter メソッドを呼び出す必要があります。なんらかのパラメーターの設定または登録に失敗すると、ステートメントが処理されるときに `SQLException` がスローされます。

詳しくは、例: 入出力パラメーターを持つプロシージャを作成するを参照してください。

`PreparedStatement` と同様に、`CallableStatement` パラメーター値は `set` メソッドを再び呼び出さなくても、処理の間は値が保持されます。出力として登録されたパラメーターには、`clearParameters` メソッドの効果はありません。`clearParameters` を呼び出した後、すべての `IN` パラメーターは再び値を設定する必要がありますが、すべての `OUT` パラメーターは再登録の必要がありません。

注: このパラメーターの概念を、パラメーター・マーカースの概念と混同しないでください。ストアード・プロシージャ呼び出しは、信頼できるパラメーター数が渡されることを要求します。`SQL` ステートメントの中には、実行時に指定される値を表す文字 "?" (パラメーター・マーカース) が含まれるものがあります。次の例を考慮して、これら 2 つの概念の違いを確認するようにしてください。

```
CallableStatement cs = con.prepareCall("CALL PROC(?, \"SECOND\", ?)");

cs.setString(1, "First");    //Parameter marker 1, Stored procedure parm 1

cs.setString(2, "Third");    //Parameter marker 2, Stored procedure parm 3
```

## ストアード・プロシージャ・パラメーターに名前アクセスする

ストアード・プロシージャのパラメーターは、次のストアード・プロシージャ宣言の例のように、関連付けられた名前を持っています。

例: ストアード・プロシージャ・パラメーター

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
CREATE
PROCEDURE MYLIBRARY.APROC
  (IN PARM1 INTEGER)
LANGUAGE SQL SPECIFIC MYLIBRARY.APROC
BODY: BEGIN
  <Perform a task here...>
END BODY
```

ここでは、`PARM1` という名前が付けられた 1 つの整数パラメーターがあります。`JDBC 3.0` では、索引だけでなく、名前によるストアード・プロシージャ・パラメーターの指定がサポートされています。このプロシージャの `CallableStatement` を設定するコードは、次のようになります。

```
CallableStatement cs = con.prepareCall("CALL APROC(?)");

cs.setString("PARM1", 6);    //Sets input parameter at index 1 (PARM1) to 6.
```

### *CallableStatement* の処理:

`CallableStatement` オブジェクトでの `SQL` ストアード・プロシージャ呼び出しの処理は、`PreparedStatement` オブジェクトで使用されるものと同じメソッドによって行われます。

## ストアード・プロシージャの結果を戻す

ストアード・プロシージャ内で 1 つの `SQL` 照会ステートメントが処理されると、そのストアード・プロシージャを呼び出したプログラムが、照会された結果を利用できます。ストアード・プロシージャ内で複数の照会を呼び出し、呼び出し側プログラムがすべての使用可能な `ResultSet` を処理することもできます。



詳しくは、例: 複数の ResultSet を持つプロシージャーを作成するを参照してください。

注: ストアード・プロシージャーが executeQuery で処理され、ResultSet が戻されない場合は、SQLException がスローされます。

## ResultSet に並行してアクセスする

『ストアード・プロシージャーの結果を戻す』では、ResultSet およびストアード・プロシージャーについてを扱っており、すべての Java Development Kit (JDK) リリースで動作する例が提供されています。この例では、ResultSet はストアード・プロシージャーが最初に開いた ResultSet から、最後に開いた ResultSet までが順番に処理されます。次の ResultSet を使う前に、ResultSet はクローズされます。

JDK 1.4 および後続のバージョンでは、ストアード・プロシージャーから ResultSet を並行して処理する機能がサポートされています。

注: この機能は、コマンド入力行インターフェース (CLI) の V5R2 による、基礎となるシステム・サポートに追加されたものです。結果として、JDK 1.4 または後続のバージョンを V5R2 より前のバージョンのシステムで動作させた場合は、このサポートを利用できません。

## ストアード・プロシージャーの更新数を戻す

ストアード・プロシージャーから更新数を戻す機能については JDBC 仕様で扱われていますが、現在のところ、iSeries プラットフォームではサポートされていません。ストアード・プロシージャー呼び出しから複数の更新数を戻す方法はありません。ストアード・プロシージャー内の準備済み SQL ステートメントからの更新数が必要な場合は、この値を戻すための 2 つの方法があります。

- 出力パラメーターとして値を戻す。
- パラメーターからの戻り値として値を戻す。これは出力パラメーターの特殊なケースです。詳しくは、『戻り値を持つストアード・プロシージャーを処理する』を参照してください。

## 戻り値が不明なストアード・プロシージャーを処理する

ストアード・プロシージャー呼び出しの結果が不明な場合は、実行メソッドを使用します。このメソッドが一度処理されると、JDBC ドライバーはアプリケーションに、API 呼び出しを通してストアード・プロシージャーが生成する結果のタイプを通知することができます。実行メソッドは、結果が 1 つか複数の ResultSet であった場合、True を戻します。ストアード・プロシージャー呼び出しから更新数は返されません。

## 戻り値を持つストアード・プロシージャーを処理する

iSeries プラットフォームは、関数の戻り値に似た、戻り値を持つストアード・プロシージャーをサポートしています。ストアード・プロシージャーからの戻り値は他のパラメーター・マークのようにラベル付けされており、ストアード・プロシージャー呼び出しによって割り当てられます。以下に、その例を示します。

```
? = CALL MYPROC(?, ?, ?)
```

ストアード・プロシージャー呼び出しからの戻り値は常に整数タイプで、他の出力パラメーターのように登録されている必要があります。

詳しくは、例: 戻り値を持つプロシージャーを作成するを参照してください。

**例: 複数の ResultSet を持つプロシージャーを作成する:**

この例では、データベースにアクセスして、複数の ResultSet を持つプロシージャを作成する方法を示しています。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;
import java.util.Properties;

public class CallableStatementExample1 {

    public static void main(java.lang.String[] args) {

        // Register the Native JDBC driver. If we cannot
        // register the driver, the test cannot continue.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Create the connection properties
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Create a procedure with multiple ResultSets.
            String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX1 " +
                "RESULT SET 2 LANGUAGE SQL READS SQL DATA SPECIFIC MYLIBRARY.SQLSPEX1 " +
                "EX1: BEGIN " +
                "    DECLARE C1 CURSOR FOR SELECT * FROM QSYS2.SYSPROCS " +
                "        WHERE SPECIFIC_SCHEMA = 'MYLIBRARY'; " +
                "    DECLARE C2 CURSOR FOR SELECT * FROM QSYS2.SYSPARMS " +
                "        WHERE SPECIFIC_SCHEMA = 'MYLIBRARY'; " +
                "    OPEN C1; " +
                "    OPEN C2; " +
                "    SET RESULT SETS CURSOR C1, CURSOR C2; " +
                "END EX1 ";

            try {
                s.executeUpdate(sql);
            } catch (SQLException e) {
                // NOTE: We are ignoring the error here. We are making
                // the assumption that the only reason this fails
                // is because the procedure already exists. Other
                // reasons that it could fail are because the C compiler
                // is not found to compile the procedure or because
                // collection MYLIBRARY does not exist on the system.
            }
            s.close();

            // Now use JDBC to run the procedure and get the results back. In
            // this case we are going to get information about 'MYLIBRARY's stored
            // procedures (which is also where we created this procedure, thereby
            // ensuring that there is something to get.
            CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX1");

            ResultSet rs = cs.executeQuery();

            // We now have the first ResultSet object that the stored procedure
            // left open. Use it.
            int i = 1;
            while (rs.next()) {
                System.out.println("MYLIBRARY stored procedure
                    " + i + " is " + rs.getString(1) + "." +
                    rs.getString(2));
            }
        }
    }
}
```

```

        i++;
    }
    System.out.println("");

    // Now get the next ResultSet object from the system - the previous
    // one is automatically closed.
    if (!cs.getMoreResults()) {
        System.out.println("Something went wrong. There should have
            been another ResultSet, exiting.");
        System.exit(0);
    }
    rs = cs.getResultSet();

    // We now have the second ResultSet object that the stored procedure
    // left open. Use that one.
    i = 1;
    while (rs.next()) {
        System.out.println("MYLIBRARY procedure " + rs.getString(1)
            + "." + rs.getString(2) +
            " parameter: " + rs.getInt(3) + " direction:
            " + rs.getString(4) +
            " data type: " + rs.getString(5));

        i++;
    }

    if (i == 1) {
        System.out.println("None of the stored procedures have any parameters.");
    }

    if (cs.getMoreResults()) {
        System.out.println("Something went wrong,
            there should not be another ResultSet.");
        System.exit(0);
    }

    cs.close(); // close the CallableStatement object
    c.close(); // close the Connection object.

} catch (Exception e) {
    System.out.println("Something failed..");
    System.out.println("Reason: " + e.getMessage());
    e.printStackTrace();
}
}
}

```

### 例: 入出力パラメーターを持つプロシージャを作成する:

この例では、データベースにアクセスして、入出力パラメーターを持つプロシージャを作成する方法を示しています。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

import java.sql.*;
import java.util.Properties;

public class CallableStatementExample2 {

    public static void main(java.lang.String[] args) {

        // Register the Native JDBC driver. If we cannot
        // register the driver, the test cannot continue.

```

```

try {
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

    // Create the connection properties
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Connect to the local iSeries database
    Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

    Statement s = c.createStatement();

    // Create a procedure with in, out, and in/out parameters.
    String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX2 " +
        "(IN P1 INTEGER, OUT P2 INTEGER, INOUT P3 INTEGER) " +
        "LANGUAGE SQL SPECIFIC MYLIBRARY.SQLSPEX2 " +
        "EX2: BEGIN " +
        "    SET P2 = P1 + 1; " +
        "    SET P3 = P3 + 1; " +
        "END EX2 ";

    try {
        s.executeUpdate(sql);
    } catch (SQLException e) {
        // NOTE: We are ignoring the error here. We are making
        // the assumption that the only reason this fails
        // is because the procedure already exists. Other
        // reasons that it could fail are because the C compiler
        // is not found to compile the procedure or because
        // collection MYLIBRARY does not exist on the system.
    }
    s.close();

    // Prepare a callable statement used to run the procedure.
    CallableStatement cs = c.prepareCall("CALL MYLIBRARY.SQLSPEX2(?, ?, ?)");

    // All input parameters must be set and all output parameters must
    // be registered. Notice that this means we have two calls to make
    // for an input output parameter.
    cs.setInt(1, 5);
    cs.setInt(3, 10);
    cs.registerOutParameter(2, Types.INTEGER);
    cs.registerOutParameter(3, Types.INTEGER);

    // Run the procedure
    cs.executeUpdate();

    // Verify the output parameters have the desired values.
    System.out.println("The value of P2 should be P1 (5) + 1 = 6. --> " + cs.getInt(2));
    System.out.println("The value of P3 should be P3 (10) + 1 = 11. --> " + cs.getInt(3));

    cs.close(); // close the CallableStatement object
    c.close(); // close the Connection object.

} catch (Exception e) {
    System.out.println("Something failed..");
    System.out.println("Reason: " + e.getMessage());
    e.printStackTrace();
}
}
}

```

### 例: 戻り値を持つプロシージャを作成する:

この例では、データベースにアクセスして、戻り値を持つプロシージャを作成する方法を示しています。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;
import java.util.Properties;

public class CallableStatementExample3 {

    public static void main(java.lang.String[] args) {

        // Register the native JDBC driver. If the driver cannot
        // be registered, the test cannot continue.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            // Create the connection properties
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

            // Connect to the local iSeries database
            Connection c = DriverManager.getConnection("jdbc:db2://*local", properties);

            Statement s = c.createStatement();

            // Create a procedure with a return value.
            String sql = "CREATE PROCEDURE MYLIBRARY.SQLSPEX3 " +
                " LANGUAGE SQL SPECIFIC MYLIBRARY.SQLSPEX3 " +
                " EX3: BEGIN " +
                "     RETURN 1976; " +
                " END EX3 ";

            try {
                s.executeUpdate(sql);
            } catch (SQLException e) {
                // NOTE: The error is ignored here. The assumption is
                //       made that the only reason this fails is
                //       because the procedure already exists. Other
                //       reasons that it could fail are because the C compiler
                //       is not found to compile the procedure or because
                //       collection MYLIBRARY does not exist on the system.
            }
            s.close();

            // Prepare a callable statement used to run the procedure.
            CallableStatement cs = c.prepareCall("? = CALL MYLIBRARY.SQLSPEX3");

            // You still need to register the output parameter.
            cs.registerOutParameter(1, Types.INTEGER);

            // Run the procedure.
            cs.executeUpdate();

            // Show that the correct value is returned.
            System.out.println("The return value
                should always be 1976 for this example:
                --> " + cs.getInt(1));

            cs.close(); // close the CallableStatement object
            c.close(); // close the Connection object.

        } catch (Exception e) {
            System.out.println("Something failed..");
            System.out.println("Reason: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

## ResultSet

ResultSet インターフェースは、照会の実行によって生成された結果へのアクセスを提供します。概念上、ResultSet のデータは、特定数の列と特定数の行を含むテーブルとして考えることができます。デフォルトでは、テーブル行は順番に検索されます。検索の対象が 1 行であれば、列の値は、任意の順序でアクセスできます。

### ResultSet の特性:

このトピックでは、ResultSet の特性を説明しています。これには ResultSet のタイプ、並行性、接続オブジェクトをコミットすることで ResultSet をクローズする機能、および ResultSet 特性の仕様などがあります。

デフォルトで、作成されるすべての ResultSet は、「順方向のみ」のタイプと「読み取り専用」の並行性を持ち、カーソルはコミット境界を超えて保持されます。例外として、WebSphere では現在カーソルの保持可能性のデフォルトが変更されていて、カーソルはコミット時に暗黙的にクローズするようになっています。これらの特性は、Statement、PreparedStatement、および CallableStatement オブジェクトでアクセス可能なメソッドを通して構成できます。

## ResultSet のタイプ

ResultSet タイプは、ResultSet に関して以下の事柄を指定します。

- ResultSet はスクロール可能かどうか。
- ResultSet インターフェースの定数によって定義されている JDBC (Java Database Connectivity) ResultSet のタイプ。

これらの ResultSet タイプの定義は以下のとおりです。

### TYPE\_FORWARD\_ONLY

ResultSet の先頭から ResultSet の末尾に向かう処理だけが行えるカーソル。これはデフォルトのタイプです。

### TYPE\_SCROLL\_INSENSITIVE

ResultSet の中を各方向にスクロールすることができるカーソル。このタイプのカーソルは、オープンしているときにデータベースに加えられる変更を感知しません。これには、照会が処理されたときやデータが取り出されるときに、照会の条件を満たす行が含まれます。

### TYPE\_SCROLL\_SENSITIVE

ResultSet の中を各方向にスクロールすることができるカーソル。このタイプのカーソルは、オープンしているときにデータベースに加えられる変更を感知します。データベースへの変更は、ResultSet データに直接影響します。

JDBC 1.0 ResultSet は常に「順方向のみ」です。スクロール可能なカーソルは JDBC 2.0 で追加されました。

注: blocking enabled および block size 接続プロパティは、TYPE\_SCROLL\_SENSITIVE カーソルの感度に影響します。ブロック化を行うと、データが JDBC ドライバー層そのものにキャッシングされ、パフォーマンスが向上します。

テーブルに行が挿入されるとき sensitive ResultSet と insensitive ResultSet の相違を示す例: 感知および非感知の ResultSet を参照してください。

変更が ResultSet の感度に基づいて SQL ステートメントの where 文節にどのように影響する可能性があるのかを示す例: ResultSet の感度を参照してください。

## 並行性

並行性は、ResultSet を更新できるかどうかを決定します。このタイプも、ResultSet インターフェースの定数によって定義されています。使用可能な並行性の設定値は以下のとおりです。

### CONCUR\_READ\_ONLY

データベースからのデータの読み取りにだけ使用される ResultSet。これはデフォルトの設定です。

### CONCUR\_UPDATEABLE

変更を加えることのできる ResultSet。これらの変更は、基になるデータベースに加えることができます。詳しくは、ResultSet の変更を参照してください。

JDBC 1.0 ResultSet は常に「順方向のみ」です。更新可能な ResultSet は JDBC 2.0 で追加されました。

注: JDBC 仕様によれば、JDBC ドライバーは、値と一緒に使用することができない場合、ResultSet の並行性設定の ResultSet タイプを変更することができます。その場合、JDBC ドライバーは Connection オブジェクトに対する警告を出します。

アプリケーションが TYPE\_SCROLL\_INSENSITIVE, CONCUR\_UPDATEABLE ResultSet を指定するある状況があります。データのコピーを作成することによって、Insensitivity (不感知) がデータベース・エンジンにインプリメントされています。このとき、このコピーを通して基になるデータベースを更新することはできません。この組み合わせが指定されている場合は、ドライバーが感度を TYPE\_SCROLL\_SENSITIVE に変更し、要求が変更されたことを示す警告を作成します。

## 保持可能性

保持可能性の特性は、Connection オブジェクトでコミットを呼び出すと ResultSet がクローズされるかどうかを決定します。保持可能性特性の処理のための JDBC API は、バージョン 3.0 での新機能です。ただし、ネイティブ JDBC ドライバーは、いくつかのリリースで接続プロパティを提供しており、この接続プロパティでは、接続の下で作成されたすべての ResultSet にそのデフォルトを指定することができます(接続プロパティと DataSource プロパティを参照)。API サポートは、接続プロパティの設定をオーバーライドします。保持可能性特性の値は、ResultSet 定数によって定義されており、それは以下のとおりです。

### HOLD\_CURSOR\_OVER\_COMMIT

オープンしているカーソルはすべて、commit 文節が呼び出されてもオープンしたままです。これはネイティブ JDBC のデフォルト値です。

### CLOSE\_CURSORS\_ON\_COMMIT

オープンしているカーソルは、commit 文節が呼び出されるとクローズされます。

注: いつでも接続でロールバックが呼び出されると、オープンしているカーソルはすべてクローズされます。この事実はあまりよく知られていませんが、データベースがカーソルを扱うときの共通の方法です。

JDBC 仕様によれば、カーソルの保持可能性のデフォルトは、インプリメンテーション定義です。一部のプラットフォームは CLOSE\_CURSORS\_ON\_COMMIT をデフォルトとして使用します。大半のアプリケーションでは普通これは問題になりませんが、コミット境界を超えてカーソルを操作する場合は、使用しているドライバーの動作に注意する必要があります。IBM Toolbox for Java JDBC ドライバーは、HOLD\_CURSORS\_ON\_COMMIT のデフォルトも使用しますが、UDB for Windows NT® 用の JDBC ドラ



イバーのデフォルトは、CLOSE\_CURSORS\_ON\_COMMIT です。

## ResultSet 特性の指定

ResultSet の特性は、その ResultSet オブジェクトがいったん作成されてしまうと、変化しません。したがって、特性はオブジェクトを作成する前に指定する必要があります。これらの特性は、多重定義されている createStatement、prepareStatement、および prepareCall メソッドのバリエーションを通して指定できます。

ResultSet 特性を指定するには、以下のトピックを参照してください。

- ResultSet 特性の指定 (Statement の場合)
- ResultSet 特性と自動生成キー・サポートの指定 (PreparedStatement の場合)
- ResultSet 特性と自動生成キー・サポートの指定 (CallableStatement の場合)

注: ResultSet タイプと ResultSet の並行性を取得するための ResultSet メソッドはありますが、ResultSet の保持可能性を取得するためのメソッドはありません。

### 例: 感知および非感知の ResultSet:

以下の例は、テーブルに行が挿入される際の、感知 ResultSet と非感知 ResultSet との違いを示しています。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;

public class Sensitive {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive test = new Sensitive();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }

    public void setup() {
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("drop table cujosql.sensitive");
            } catch (SQLException e) {
                // Ignored.
            }

            s.executeUpdate("create table cujosql.sensitive(col1 int)");
            s.executeUpdate("insert into cujosql.sensitive values(1)");
        }
    }
}
```

```

        s.executeUpdate("insert into cujosql.sensitive values(2)");
        s.executeUpdate("insert into cujosql.sensitive values(3)");
        s.executeUpdate("insert into cujosql.sensitive values(4)");
        s.executeUpdate("insert into cujosql.sensitive values(5)");
        s.close();
    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another: " + another.getMessage());
        }
    }
}

public void run(String sensitivity) {
    try {
        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

        // Fetch the five values that are there.
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        System.out.println("fetched the five rows...");

        // Note: If you fetch the last row, the ResultSet looks
        //        closed and subsequent new rows that are added
        //        are not be recognized.

        // Allow another statement to insert a new value.
        Statement s2 = connection.createStatement();
        s2.executeUpdate("insert into cujosql.sensitive values(6)");
        s2.close();

        // Whether a row is recognized is based on the sensitivity setting.
        if (rs.next()) {
            System.out.println("There is a row now: " + rs.getInt(1));
        } else {
            System.out.println("No more rows.");
        }
    }

    } catch (SQLException e) {
        System.out.println("SQLException exception: ");
    }
}

```

```

        System.out.println("Message:....." + e.getMessage());
        System.out.println("SQLState:...." + e.getSQLState());
        System.out.println("Vendor Code:." + e.getErrorCode());
        System.out.println("-----");
        e.printStackTrace();
    }
    catch (Exception ex) {
        System.out.println("An exception other than an SQLException was thrown: ");
        ex.printStackTrace();
    }
}

```

```

public void cleanup() {
    try {
        connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

#### 例: *ResultSet* の感度:

以下の例は、*ResultSet* の感度に基づいた、変更が SQL ステートメントの *where* 文節に与える影響を示しています。

この例にはフォーマット設定の正しくない箇所があるかもしれません。これは、この例を印刷ページに収めるためです。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;

public class Sensitive2 {

    public Connection connection = null;

    public static void main(java.lang.String[] args) {
        Sensitive2 test = new Sensitive2();

        test.setup();
        test.run("sensitive");
        test.cleanup();

        test.setup();
        test.run("insensitive");
        test.cleanup();
    }

    public void setup() {

        try {
            System.out.println("Native JDBC used");
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

```

```

Statement s = connection.createStatement();
try {
    s.executeUpdate("drop table cujosql.sensitive");
} catch (SQLException e) {
    // Ignored.
}

s.executeUpdate("create table cujosql.sensitive(col1 int)");
s.executeUpdate("insert into cujosql.sensitive values(1)");
s.executeUpdate("insert into cujosql.sensitive values(2)");
s.executeUpdate("insert into cujosql.sensitive values(3)");
s.executeUpdate("insert into cujosql.sensitive values(4)");
s.executeUpdate("insert into cujosql.sensitive values(5)");

try {
    s.executeUpdate("drop table cujosql.sensitive2");
} catch (SQLException e) {
    // Ignored.
}

s.executeUpdate("create table cujosql.sensitive2(col2 int)");
s.executeUpdate("insert into cujosql.sensitive2 values(1)");
s.executeUpdate("insert into cujosql.sensitive2 values(2)");
s.executeUpdate("insert into cujosql.sensitive2 values(3)");
s.executeUpdate("insert into cujosql.sensitive2 values(4)");
s.executeUpdate("insert into cujosql.sensitive2 values(5)");

s.close();

} catch (Exception e) {
    System.out.println("Caught exception: " + e.getMessage());
    if (e instanceof SQLException) {
        SQLException another = ((SQLException) e).getNextException();
        System.out.println("Another: " + another.getMessage());
    }
}

}

public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select col1, col2 From cujosql.sensitive,
            cujosql.sensitive2 where col1 = col2");

        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
    }
}

```

```

rs.next();
System.out.println("value is " + rs.getInt(1));

System.out.println("fetched the four rows...");

// Another statement creates a value that does not fit the where clause.
Statement s2 =
    connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATEABLE);
ResultSet rs2 = s2.executeQuery("select *
from cujosql.sensitive where col1 = 5 FOR UPDATE");
rs2.next();
rs2.updateInt(1, -1);
rs2.updateRow();
s2.close();

if (rs.next()) {
    System.out.println("There is still a row: " + rs.getInt(1));
} else {
    System.out.println("No more rows.");
}

} catch (SQLException e) {
    System.out.println("SQLException exception: ");
    System.out.println("Message:....." + e.getMessage());
    System.out.println("SQLState:...." + e.getSQLState());
    System.out.println("Vendor Code:." + e.getErrorCode());
    System.out.println("-----");
    e.printStackTrace();
}
catch (Exception ex) {
    System.out.println("An exception other
than an SQLException was thrown: ");
    ex.printStackTrace();
}
}

public void cleanup() {
    try {
        connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

### カーソル移動:

iSeries Java Database Connectivity (JDBC) ドライバーは、スクロール可能な `ResultSet` をサポートします。スクロール可能な `ResultSet` では、いくつかのカーソル配置メソッドを使用して、データの行をどんな順序でも処理できます。

`ResultSet.next` メソッドを使用すると、`ResultSet` 内を 1 行ずつ移動します。Java Database Connectivity (JDBC) 2.0 では、iSeries JDBC ドライバーはスクロール可能な `ResultSet` をサポートします。スクロール可能な `ResultSet` では、`previous`、`absolute`、`relative`、`first`、および `last` メソッドを使用することにより、データの行をどのような順序でも処理できます。

デフォルトでは、JDBC `ResultSet` は常に「順方向のみ」です。これは、唯一の有効なカーソル配置メソッドが `next()` であることを意味します。スクロール可能な `ResultSet` は明示的に要求する必要があります。詳しくは、`ResultSet` のタイプを参照してください。

スクロール可能な `ResultSet` では、以下のカーソル配置メソッドが使用できます。

| メソッド                             | 説明  |
|----------------------------------|---|
| <code>Next</code>                | このメソッドは、 <code>ResultSet</code> 内でカーソルを順方向に 1 行移動します。<br><br>このメソッドは、カーソルが有効な行に配置されれば <code>true</code> を、そうでなければ <code>false</code> を返します。   |
| <code>Previous</code>            | このメソッドは、 <code>ResultSet</code> 内でカーソルを逆方向に 1 行移動します。<br><br>このメソッドは、カーソルが有効な行に配置されれば <code>true</code> を、そうでなければ <code>false</code> を返します。   |
| <code>First</code>               | このメソッドは、カーソルを <code>ResultSet</code> 内の最初の行に移動します。<br><br>このメソッドは、カーソルが最初の行に配置されれば <code>true</code> を、 <code>ResultSet</code> が空なら <code>false</code> を返します。   |
| <code>Last</code>                | このメソッドは、カーソルを <code>ResultSet</code> 内の最後の行に移動します。<br><br>このメソッドは、カーソルが最後の行に配置されれば <code>true</code> を、 <code>ResultSet</code> が空なら <code>false</code> を返します。   |
| <code>BeforeFirst</code>         | このメソッドは、カーソルを <code>ResultSet</code> 内の最初の行のすぐ前に移動します。<br><br>空の <code>ResultSet</code> には、このメソッドは無効です。このメソッドからの戻り値はありません。  |
| <code>AfterLast</code>           | このメソッドは、カーソルを <code>ResultSet</code> 内の最後の行のすぐ後ろに移動します。<br><br>空の <code>ResultSet</code> には、このメソッドは無効です。このメソッドからの戻り値はありません。   |
| <code>Relative (int rows)</code> | このメソッドは、カーソルをその現行位置に相対する位置へ移動します。 <ul style="list-style-type: none"> <li>行が 0 の場合、このメソッドは無効です。</li> <li>行が正の場合、カーソルは順方向にその行数だけ移動されます。現行行から <code>ResultSet</code> の末尾までの行数が、入力パラメーターで指定された行数より少ない場合、このメソッドは <code>afterLast</code> と同様に操作します。</li> <li>行が負の場合、カーソルは逆方向にその行数だけ移動されます。現行行から <code>ResultSet</code> の先頭までの行数が、入力パラメーターで指定された行数より少ない場合、このメソッドは <code>beforeFirst</code> と同様に操作します。</li> </ul><br>このメソッドは、カーソルが有効な行に配置されれば <code>true</code> を、そうでなければ <code>false</code> を返します。 |

| メソッド               | 説明  |
|--------------------|---|
| Absolute (int row) | <p>このメソッドは、カーソルを行値で指定された行に移動します。</p> <p>行値が正の場合、カーソルは <code>ResultSet</code> の先頭から数えてその行数番目に配置されます。最初の行の番号は 1、2 番目は 2 です。 <code>ResultSet</code> 内の行数が行値で指定された行数より少ない場合、このメソッドは <code>afterLast</code> と同じ方法で操作します。</p> <p>行値が負の場合、カーソルは <code>ResultSet</code> の終了から数えてその行数番目に配置されます。最終行の番号は -1、最後から 2 番目は -2 です。 <code>ResultSet</code> 内の行数が行値で指定された行数より少ない場合、このメソッドは <code>beforeFirst</code> と同じ方法で操作します。</p> <p>行値が 0 の場合、このメソッドは <code>beforeFirst</code> と同じ方法で操作します。</p> <p>このメソッドは、カーソルが有効な行に配置されれば <code>true</code> を、そうでなければ <code>false</code> を返します。</p> |

### ResultSet データの検索:

`ResultSet` オブジェクトは、行の列データを取得するためのいくつかのメソッドが備わっています。どのメソッドも、`get<Type>` (`<Type>` は Java データ・タイプ) という形式です。これらのメソッドには、たとえば、`getInt`、`getLong`、`getString`、`getTimestamp`、`getBlob` などがあります。これらメソッドのほとんどすべては単一のパラメーターをとり、そのパラメーターは `ResultSet` 内の列索引か、列名のいずれかです。

`ResultSet` の列には番号が付けられます。番号は 1 から始まります。列名が使用され、`ResultSet` 内に同じ名前を持つ列が複数ある場合は、最初のものが戻されます。複数のパラメーターをとる `get<Type>` メソッドもあります。オプションの `Calendar` オブジェクトはその一例です。このオブジェクトは `getTime`、`getDate`、および `getTimestamp` に渡すことができます。詳しくは、`java.sql` パッケージの Javadoc を参照してください。

オブジェクトを戻す `get` メソッドの場合、`ResultSet` の列がヌルのときは、戻り値はヌルになります。プリミティブ・タイプの場合は、ヌルは戻せません。その場合、値は 0 か `false` です。アプリケーションがヌルと 0 または `false` を区別する必要がある場合は、呼び出しの直後に `wasNull` メソッドを使用します。このメソッドは、値が実際の 0 または `false` 値かどうか、あるいは、その値は `ResultSet` 値が本当にヌルであったために戻されたのかどうかを判別します。

`ResultSet` インターフェースの使用法の例については、例: IBM Developer Kit for Java の `ResultSet` インターフェースを参照してください。

### ResultSetMetaData サポート

`getMetaData` メソッドは、`ResultSet` オブジェクトに対して呼び出されると、`ResultSet` オブジェクトの列を記述する `ResultSetMetaData` オブジェクトを戻します。実行されている SQL ステートメントが実行時まで認識されない場合、`ResultSetMetaData` を使用すれば、データの検索に使用すべき `get` メソッドを判別できません。以下のコード例では、`ResultSetMetaData` を使用して `ResultSet` 内の各列タイプを判別しています。

**例:** `ResultSetMetaData` を使用して `ResultSet` 内の各列タイプを判別する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
ResultSet rs = stmt.executeQuery(sqlString);
ResultSetMetaData rsmd = rs.getMetaData();
int colType [] = new int[rsmd.getColumnCount()];
for (int idx = 0, int col = 1; idx < colType.length; idx++, col++)
colType[idx] = rsmd.getColumnType(col);
```



ResultSetMetaData インターフェースの使用法の例については、『例: IBM Developer Kit for Java 用の ResultSetMetaData インターフェース』を参照してください。

### 例: IBM Developer Kit for Java 用の ResultSetMetaData インターフェース:

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;

/**
ResultSetMetaDataExample.java

This program demonstrates using a ResultSetMetaData and
a ResultSet to display all the metadata about a ResultSet
created querying a table. The user passes the value for the
table and library in.
**/
public class ResultSetMetaDataExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage: java ResultSetMetaDataExample <library> <table>");
            System.out.println("where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Get a database connection and prepare a statement.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
            int rowCount = 0;
            for (int i = 1; i <= colCount; i++) {
                System.out.println("Information about column " + i);
                System.out.println("  Name.....: " + rsmd.getColumnName(i));
                System.out.println("  Data Type.....: " + rsmd.getColumnType(i) +
                    " ( " + rsmd.getColumnTypeName(i) + " )");
                System.out.println("  Precision.....: " + rsmd.getPrecision(i));
                System.out.println("  Scale.....: " + rsmd.getScale(i));
                System.out.print ("  Allows Nulls..: ");
                if (rsmd.isNullable(i)==0)
                    System.out.println("false");
                else
                    System.out.println("true");
            }
        } catch (Exception e) {
            // Handle any errors.
            System.out.println("Oops... we have an error... ");
            e.printStackTrace();
        } finally {
            // Ensure we always clean up. If the connection gets closed, the
            // statement under it closes as well.
            if (con != null) {
```

```

        try {
            con.close();
        } catch (SQLException e) {
            System.out.println("Critical error - cannot close connection object");
        }
    }
}
}
}

```

## リンク集

コードのサンプルに関する特記事項

### ResultSet の変更:

iSeries JDBC ドライバーでは、以下のタスクを実行することによって、ResultSet を変更できます。

ResultSet のデフォルト設定は、「読み取り専用」です。しかし、Java Database Connectivity (JDBC) 2.0 では、iSeries JDBC ドライバーは更新可能 ResultSet を完全にサポートしています。

ResultSet の更新方法については、ResultSet の並行性を参照することができます。

### 行の更新

行は ResultSet インターフェースを通してデータベース・テーブル内で更新できます。このプロセスに関連したステップは、以下のとおりです。

1. 種々の update<Type> メソッド (<Type> は Java データ・タイプ) を使用して、特定の行の値を変更する。 update<Type> メソッドは、値の検索に使用することができる get<Type> メソッドと対応しています。
2. 行を基になるデータベースに適用する。

データベースそのものは、2 番目のステップを実行するまでは更新されません。 updateRow メソッドを呼び出さずに ResultSet の列を更新しても、データベースに変更は加えられません。

計画した行に対する更新は、cancelUpdates メソッドで破棄することができます。いったん updateRow メソッドを呼び出せば、データベースに対する変更は確定され、元に戻すことはできません。

**注:** データベースが更新済みの行を指し示す手段を持っていない場合、rowUpdated メソッドは常に false を戻します。これに対応して updatesAreDetected メソッドも false を戻します。

### 行の削除

行は ResultSet インターフェースを通してデータベース・テーブル内で削除できます。 deleteRow メソッドを指定すると、現行の行が削除されます。

### 行の挿入

行は ResultSet インターフェースを通してデータベース・テーブルに挿入できます。このプロセスでは「挿入行」を使用します。アプリケーションはこの「挿入行」に実際にカーソルを移動し、データベースに挿入する値を設定します。このプロセスに関連したステップは、以下のとおりです。

1. 挿入行にカーソルを置く。
2. 新しい行の各列の値を設定する。
3. データベースに行を挿入し、任意でカーソルを ResultSet 内の現行の行に戻す。

注: 新規行は、カーソルが置かれているテーブルには挿入されません。これらは通常、テーブル・データ・スペースの末尾に追加されます。リレーショナル・データベースは、デフォルトでは位置依存ではありません。たとえば、3 番目の行にカーソルを移動して何かを挿入しても、後のユーザーがデータを取り出すときに 4 番目の行の前にそれが表示されることはありません。

## 定位置更新のサポート

ResultSet を通してデータベースを更新するためのメソッド以外に、定位置更新を発行するための SQL ステートメントを使用することができます。このサポートは、名前付きカーソルを使用することを前提としています。JDBC は、これらの値へのアクセスを可能にする Statement からの setCursorName メソッドと、ResultSet からの getCursorName メソッドを提供しています。

supportsPositionedUpdated と supportsPositionedDelete の 2 つの DatabaseMetaData メソッドはどちらも、ネイティブ JDBC ドライバーでこの機能がサポートされていれば true を返します。

詳しくは、例: 他のステートメントのカーソルを介してステートメントで値を変更するを参照してください。

詳しくは、例: 他のステートメントのカーソルを介してテーブルから値を除去するを参照してください。

### 例: 他のステートメントのカーソルを介してテーブルから値を除去する:

以下は、他のステートメントのカーソルを介してテーブルから値を除去する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;

public class UsingPositionedDelete {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {
        UsingPositionedDelete test = new UsingPositionedDelete();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }

    /**
    Handle all the required setup work.
    */
    public void setup() {
        try {
            // Register the JDBC driver.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignore problems here.
            }
        }
    }
}
```

```

        s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
            "COL_IND INT, COL_VALUE CHAR(20)) ");

        for (int i = 1; i <= 10; i++) {
            s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
        }

        s.close();
    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        e.printStackTrace();
    }
}

```

/\*\*

In this section, all the code to perform the testing should be added. If only one connection to the database is needed, the global variable 'connection' can be used.

\*\*/

```

    public void run() {
        try {
            Statement stmt1 = connection.createStatement();

            // Update each value using next().
            stmt1.setCursorName("CUJ0");
            ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                "FOR UPDATE OF COL_VALUE");

            System.out.println("Cursor name is " + rs.getCursorName());

            PreparedStatement stmt2 = connection.prepareStatement
                ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
                    rs.getCursorName ());

            // Loop through the ResultSet and update every other entry.
            while (rs.next ()) {
                if (rs.next())
                    stmt2.execute ();
            }

            // Clean up the resources after they have been used.
            rs.close ();
            stmt2.close ();

        } catch (Exception e) {
            System.out.println("Caught exception: ");
            e.printStackTrace();
        }
    }
}

```

/\*\*

In this section, put all clean-up work for testing.

\*\*/

```

    public void cleanup() {
        try {
            // Close the global connection opened in setup().
            connection.close();
        } catch (Exception e) {

```

```

        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}

/**
Display the contents of the table.
**/
public void displayTable()
{
    try {
        Statement s = connection.createStatement();
        ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

        while (rs.next ()) {
            System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
        }

        rs.close ();
        s.close();
        System.out.println("-----");
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## リンク集

コードのサンプルに関する特記事項

### 例: 他のステートメントのカーソルを介してステートメントで値を変更する:

以下は、他のステートメントのカーソルを介したステートメントによる値の変更方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;

public class UsingPositionedUpdate {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {

        UsingPositionedUpdate test = new UsingPositionedUpdate();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }

    /**
Handle all the required setup work.
**/
    public void setup() {
        try {
            // Register the JDBC driver.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

```

```

connection = DriverManager.getConnection("jdbc:db2:*local");

Statement s = connection.createStatement();
try {
    s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
} catch (SQLException e) {
    // Ignore problems here.
}

s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
    "COL_IND INT, COL_VALUE CHAR(20)) ");

for (int i = 1; i <= 10; i++) {
    s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
}

s.close();

} catch (Exception e) {
    System.out.println("Caught exception: " + e.getMessage());
    e.printStackTrace();
}
}

```

/\*\*

In this section, all the code to perform the testing should be added. If only one connection to the database is required, the global variable 'connection' can be used.

\*\*/

```

public void run() {
    try {
        Statement stmt1 = connection.createStatement();

        // Update each value using next().
        stmt1.setCursorName("CUJO");
        ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
            "FOR UPDATE OF COL_VALUE");

        System.out.println("Cursor name is " + rs.getCursorName());

        PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
            + " CUJOSQL.WHERECUREX
            SET COL_VALUE = 'CHANGED'
            WHERE CURRENT OF "
            + rs.getCursorName ());

        // Loop through the ResultSet and update every other entry.
        while (rs.next ()) {
            if (rs.next())
                stmt2.execute ();
        }

        // Clean up the resources after they have been used.
        rs.close ();
        stmt2.close ();

    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}

```

```

/**
In this section, put all clean-up work for testing.
**/
public void cleanup() {
    try {
        // Close the global connection opened in setup().
        connection.close();

    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}

/**
Display the contents of the table.
**/
public void displayTable()
{
    try {
        Statement s = connection.createStatement();
        ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

        while (rs.next ()) {
            System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
        }

        rs.close ();
        s.close();
        System.out.println("-----");
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## リンク集

コードのサンプルに関する特記事項

### ResultSet の作成:

ResultSet オブジェクトを作成するには、executeQuery メソッド、または他のメソッドを使用できます。このトピックでは、ResultSet の作成のためのオプションについて説明します。

これらのメソッドは、Statement、PreparedStatement、または CallableStatement インターフェースから実行します。ただし、方法は他にもあります。たとえば、getColumnNames、getTableNames、getUDTs、getPrimaryKeys などの DatabaseMetaData メソッドは、ResultSet を戻します。単一の SQL ステートメントの処理で、複数の ResultSet を戻すこともできます。さらに、Statement、PreparedStatement、または CallableStatement インターフェースで提供されている execute メソッドを呼び出した後に、getResultSet メソッドを使用して ResultSet オブジェクトを検索することができます。

詳しくは、例: 複数の ResultSet を持つプロシージャを作成するを参照してください。



## ResultSet のクローズ

ResultSet オブジェクトは、関連する Statement オブジェクトがクローズすると自動的にクローズされますが、ResultSet オブジェクトは使用しなくなったらクローズすることをお勧めします。そうすれば、即時に内部データベース・リソースが解放され、それによってアプリケーションのスループットが増大する可能性があります。

DatabaseMetaData 呼び出しによって生成された ResultSet をクローズすることも大切です。これらの ResultSet の作成に使用された Statement オブジェクトに直接アクセスすることはできないので、Statement オブジェクトで直接 close を呼び出すことはしません。これらのオブジェクトは互いにリンクされ、外部の ResultSet オブジェクトをクローズすると、JDBC ドライバーが内部の Statement オブジェクトをクローズするようになっています。これらのオブジェクトを手動でクローズしない場合、システムは引き続き作動しますが、必要以上のリソースを使用することになります。

注: ResultSet の保持可能性特性でも ResultSet を自動的にクローズすることができます。close は ResultSet オブジェクトで何回でも呼び出すことができます。

### 例: IBM Developer Kit for Java 用の ResultSet インターフェース:

以下は、ResultSet インターフェースの使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;

/**
ResultSetExample.java

This program demonstrates using a ResultSetMetaData and
a ResultSet to display all the data in a table even though
the program that gets the data does not know what the table
is going to look like (the user passes in the values for the
table and library).
**/
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage: java ResultSetExample <library> <table>");
            System.out.println(" where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Get a database connection and prepare a statement.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

            int colCount = rsmd.getColumnCount();
```



トで表される物理接続への一時ハンドルとして機能します。アプリケーションにとっては、Connection オブジェクトが戻されれば、それら 2 つの間には大きな違いはありません。Connection オブジェクトでクローズ・メソッドを呼び出すときに、わずかな違いが出てくるだけです。この呼び出しは、論理接続を無効にして、別のアプリケーションが物理接続を使用できるプールに物理接続を戻します。この技法を使用すると、多くの論理接続オブジェクトで、1 つの物理接続を再利用できるようになります。

## 接続プーリングの設定

接続プーリングは、ConnectionPoolDataSource オブジェクトを参照する DataSource オブジェクトを作成することで実現します。ConnectionPoolDataSource オブジェクトには、プール保守のさまざまな要素を処理するために設定できるプロパティがあります。

UDBDataSource および UDBConnectionPoolDataSource を使用して接続プーリングをセットアップする方法の詳細の例を参照してください。この例で JNDI が担当する役割の詳細については、Java Naming and Directory Interface (JNDI) も参照できます。

例では、2 つの DataSource オブジェクトを 1 つにバインドするリンクは、dataSourceName です。このリンクは、プーリングを自動的に管理する ConnectionPoolDataSource オブジェクトへの接続の確立を延期するよう、DataSource オブジェクトに通知します。

## プーリングおよび非プーリング・アプリケーション

Connection プーリングを使用するアプリケーションと、それを使用しないアプリケーションとの間には、違いはありません。したがって、プーリング・サポートは、アプリケーション・コードの完了後に追加できます。その際に、アプリケーション・コードに変更を加える必要はありません。

詳細は、例: 接続プーリングのパフォーマンスをテストするを参照してください。

次に示すのは、開発時に前述のプログラムをローカルに実行するときの出力です。

非プーリング DataSource バージョンのタイミングを開始 (Start timing the non-pooling DataSource version...) 経過時間: 6410 (Time spent: 6410)

プーリング・バージョンのタイミングを開始... (Start timing the pooling version...) 経過時間: 282 (Time spent: 282)

Java プログラムが完了しました (Java program completed)

デフォルトでは、UDBConnectionPoolDataSource は 1 つの接続をプーリングします。アプリケーションが接続を複数回必要としていて、一度に 1 つの接続だけを必要とする場合、UDBConnectionPoolDataSource を使用することは、完全な解決策になります。多数の接続を同時に必要とする場合は、ConnectionPoolDataSource を構成し、必要とリソースを満たす必要があります。

### 例: UDBDataSource および UDBConnectionPoolDataSource で接続プーリングをセットアップする:

以下に、UDBDataSource および UDBConnectionPoolDataSource で接続プーリングを使用する方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
```

```

import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class ConnectionPoolingSetup
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        // Create a ConnectionPoolDataSource implementation
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object");

        // Establish a JNDI context and bind the connection pool data source
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Create a standard data source that references it.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting pooling");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
    }
}

```

**例: 接続プーリングのパフォーマンスをテストする:**

以下に、プーリングされたときのパフォーマンスとプーリングされていないときのパフォーマンスを対比してテストする方法を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;

public class ConnectionPoolingTest
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        Context ctx = new InitialContext();
        // Do the work without a pool:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("\nStart timing the non-pooling DataSource version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Do the work with pooling:
        ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
    }
}

```

```

        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}

```

### ConnectionPoolDataSource のプロパティ:

ConnectionPoolDataSource インターフェースは、用意されている一連のプロパティを使用することによって構成できます。

次の表には、このようなプロパティの説明が載せられています。

| プロパティ           | 説明  |
|-----------------|---|
| initialPoolSize | <p>プールを初めてインスタンス化する場合、このプロパティにより、プールに配置する接続数が決定されます。この値が minPoolSize と maxPoolSize の範囲外で指定される場合、作成する初期接続の数として minPoolSize または maxPoolSize が使用されます。</p>  |
| maxPoolSize     | <p>プールが使用される場合、プールに含まれる数よりも多くの接続を要求できます。このプロパティでは、プールに作成できる最大接続数を指定します。</p> <p>プールが最大サイズになっていて、すべての接続が使用中である場合には、アプリケーションは、プールに戻される接続を「ブロック」せずに待機します。代わりに、JDBC ドライバーは、DataSource プロパティに基づいて新しい接続を構成し、接続を戻します。</p> <p>maxPoolSize として 0 が指定される場合、渡すことのできるリソースがシステムにある限り、プールを無限に拡大することができます。</p>                                  |
| minPoolSize     | <p>プールを使用中のスパイクは、それに含まれる接続の数を増やすことができます。アクティビティ・レベルが、プールからいくつかの Connection が引き出されない点まで下がると、特に理由はなくてもリソースが取られます。</p> <p>そのようなケースでは、JDBC ドライバーに、累積したいくつかの接続を解放する機能があります。このプロパティを使用すると、JDBC に接続を解放するよう通知し、使用できる特定の接続数を常に保つようにすることができます。</p> <p>minPoolSize として 0 が指定される場合、プールがすべての接続を解放し、アプリケーションが接続要求ごとに接続時間を実際に処理できるようになります。</p> |

| プロパティ         | 説明  |
|---------------|---|
| maxIdleTime   | <p>接続は、使用されずに放置されている期間を追跡します。このプロパティは、接続を解放する前に、アプリケーションが接続を未使用にしておける期間を指定します (つまり、必要な数よりもさらに多くの接続が存在するということです)。</p> <p>このプロパティは、秒単位の時間であり、実際のクローズが行われる時刻を指定するものではありません。ここでは、接続を解放するための十分な時間がいつ経過するかを指定します。</p> |
| propertyCycle | このプロパティは、これらの規則の実行と実行の間で、経過することを認められている秒数を表します。   |

**注:** maxIdleTime または propertyCycle のいずれかの時間を 0 に設定する場合、JDBC ドライバーは、それ自体ではプールから除去される接続を検査しません。initial、min、および max サイズに指定される規則はまだ有効です。

maxIdleTime および propertyCycle が 0 でない場合、プールを監視するために管理スレッドが使用されます。このスレッドは、propertyCycle 秒ごとにウェイクし、プール内のすべての接続を検査して、maxIdleTime 秒以上使用されていない接続を確認します。この基準に当てはまる接続は、minPoolSize に達するまでプールから除去されます。

#### DataSource ベースのステートメント・プーリング:

UDBConnectionPoolDataSource インターフェース上で使用できる maxStatements プロパティを使用すると、接続プール内でのステートメント・プーリングが可能になります。ステートメント・プーリングだけが、PreparedStatements および CallableStatements に影響します。ステートメント・オブジェクトは、プールされません。

ステートメント・プーリングの実装は、接続プーリングの実装と似ています。アプリケーションが Connection.prepareStatement("select \* from tablex") を呼び出すと、プーリング・モジュールが、接続下で Statement オブジェクトが準備されているかどうかを確認します。準備されている場合、物理オブジェクトではなく、論理 PreparedStatement オブジェクトが渡されます。クローズを呼び出すと、Connection オブジェクトがプールに戻され、論理 Connection オブジェクトが出され、そして Statement オブジェクトが再利用できるようになります。

maxStatements プロパティを使用すると、DataSource は、接続下でプールできるステートメントの数を指定できます。値が 0 の場合、ステートメント・プーリングを使用しないことを示します。ステートメント・プールがいっぱいの場合、使用された一番古いアルゴリズムが適用され、出されるステートメントが判別されます。

例: 2 つの DataSource のパフォーマンスのテストでは、接続プーリングだけを使用する 1 つの DataSource と、ステートメントおよび接続プーリングを使用するもう 1 つの DataSource をテストします。

次の例は、開発時にこのプログラムをローカルに実行するときの出力です。



ステートメント・プーリングのデータ・ソースを展開しています (Deploying statement pooling data source)  
接続プーリング専用バージョンのタイミングを開始... (Start timing the connection pooling only version...)  
経過時間: 26312 (Time spent: 26312)

ステートメント・プーリング・バージョンのタイミングを開始... (Starting timing the statement pooling version...)  
経過時間: 2292 (Time spent: 2292) Java プログラムが完了しました (Java program completed)

### 例: 2 つの *DataSource* のパフォーマンスをテストする:

次に、接続プーリングだけを使用する 1 つの *DataSource* と、ステートメントと接続プーリングを使用する別の *DataSource* をテストする例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class StatementPoolingTest
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();

        System.out.println("deploying statement pooling data source");
        deployStatementPoolDataSource();

        // Do the work with connection pooling only.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("\nStart timing the connection pooling only version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Do the work with statement pooling added.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("\nStart timing the statement pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}
```



```

private static void deployStatementPoolDataSource()
throws Exception
{
    // Create a ConnectionPoolDataSource implementation
    UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
    cpds.setDescription("Connection Pooling DataSource object with Statement pooling");
    cpds.setMaxStatements(10);

    // Establish a JNDI context and bind the connection pool data source
    Context ctx = new InitialContext();
    ctx.rebind("StatementSupport", cpds);

    // Create a standard datasource that references it.
    UDBDataSource ds = new UDBDataSource();
    ds.setDescription("DataSource supporting statement pooling");
    ds.setDataSourceName("StatementSupport");
    ctx.rebind("StatementPoolingDataSource", ds);
}
}

```

### 独自の接続プーリングの構築:

DataSourcees のサポートを必要としない、または別の製品に依存しない、独自の接続およびステートメント・プーリングを開発できます。

プーリング技法は、小さな Java アプリケーションで実演されますが、このことは、サーブレットや大規模な n 層アプリケーションにも同じように当てはまります。この例は、パフォーマンス問題を実演するときに使用されます。

デモンストレーション・アプリケーションには、以下の 2 つの機能があります。

- 新しい索引と名前をデータベース・テーブルに挿入すること。
- 指定した索引の名前をテーブルから読み取ること。

接続プーリング・アプリケーションの完全なコードは、以下からダウンロードできます。

### JDBC tips and trick

このアプリケーション例は、速く動作しません。このコードを使用して `getValue` メソッドに対して 100 の呼び出しを実行し、`putValue` メソッドに対して 100 の呼び出しを実行すると、標準のワークステーションで平均 31.86 秒かかりました。

問題は、要求ごとに膨大なデータベース作業が存在することです。つまり、接続を確立し、ステートメントを入手し、ステートメントを処理し、ステートメントをクローズし、そして接続をクローズするわけです。行ったことすべてをそれぞれの要求後に破棄してしまうのではなく、このプロセスの部分を再利用するための方法があるはずで、**接続プーリング**は、接続の作成コードを、プールから接続を入手するコードに置き換え、接続のクローズ・コードを、使用する接続をプールに戻すコードに置き換えます。

接続プーリングのコンストラクターは、接続を作成してプールに置きます。プール・クラスには、使用する接続を探し、接続を使用した処理が終了したときに、その接続をプールに戻すための、`take` および `put` メソッドがあります。プール・オブジェクトは共用リソースであるため、これらのメソッドは同期化されますが、プーリングされたリソースを複数のスレッドで同時に操作するわけではありません。

`getValue` メソッドでは、呼び出しコードに変更が加えられています。 `putValue` メソッドは示されていませんが、確実に変更が加えられていて、IBM の Developer Kit for Java JDBC Web ページから利用できま

す。接続プール・オブジェクトのインスタンス化についても示されていません。コンストラクターを呼び出して、たくさんある必要な接続オブジェクトをプールに渡すことができます。このステップは、アプリケーションを開始するときに行うようにします。

変更が加えられた以前のアプリケーションを実行する (つまり、100 `getValue` メソッドと 100 `putValue` メソッドを要求する) 場合、適切な接続プーリング・コードを使用して平均 13.43 秒かかりました。接続プーリングを使用しないと、ワークロードの処理時間は、元の処理時間の半分未満に縮まります。

## 独自のステートメント・プーリングの構築

接続プーリングを使用する場合、各ステートメントを処理するときには、ステートメントの作成とクローズに時間がかかります。これは、再利用できるオブジェクトを無駄にしているもう一つの例といえます。

オブジェクトを再利用するために、準備したステートメント・クラスを使用できます。ほとんどのアプリケーションで、小さな変更が加えられた同じ SQL ステートメントが再利用されます。たとえば、アプリケーションで 1 回反復がある場合、次の照会を生成できます。

```
SELECT * from employee where salary > 100000
```

次の反復では、次の照会を生成できます。

```
SELECT * from employee where salary > 50000
```

これは同じ照会ですが、別のパラメーターを使用しています。両方の照会を、次の照会で実現できます。

```
SELECT * from employee where salary > ?
```

その後、最初の照会の処理時にパラメーター・マーカー (疑問符で示される) を 100000 に設定し、2 番目の照会の処理時に 50000 に設定します。このようにすると、接続プールで実現できる機能以外の 3 つの理由で、パフォーマンスが拡張されます。

- 作成されるオブジェクトがより少なく済む。要求のたびに `Statement` オブジェクトが作成されるのではなく、`PreparedStatement` オブジェクトが作成されて再利用されます。したがって、実行するコンストラクターが少なく済みます。
- SQL ステートメントを設定するデータベース作業 (準備という) を再利用できる。SQL ステートメントの準備は、SQL ステートメント・テキストの内容と、要求されたタスクをシステムで実現する方法を識別することが関係するため、それなりに高く付きます。
- 別のオブジェクト作成を除去するときに、あまり考慮されない利点がある。作成されなかったものを破棄する必要はありません。このモデルは、Java ガーベッジ・コレクター上ではより使い勝手が良く、ユーザーが多くて時間がかかっているパフォーマンスの点でも有利です。

デモンストレーション・プログラムは、`Connection` ではなく、`PreparedStatement` オブジェクトをプールするよう変更できます。プログラムを変更すると、さらに多くのオブジェクトを再利用し、パフォーマンスを改善できます。プールされるオブジェクトを含むクラスを作成することから始められます。このクラスは、使用するさまざまなリソースをカプセル化しなければなりません。接続プールの例では、`Connection` が唯一のプーリングされたリソースであるため、クラスをカプセル化する必要はありませんでした。プールされた各オブジェクトには、`Connection` と 2 つの `PreparedStatement` が含まれていなければなりません。その後、接続の代わりにデータベース・アクセス・オブジェクトを含むプール・クラスを作成できます。

最後に、アプリケーションを変更して、データベース・アクセス・オブジェクトを入手し、使用するオブジェクトからリソースを指定する必要があります。特定のリソースを指定すること以外は、アプリケーションは同じままです。

この変更を加えた上で実行される同じテストは平均 0.83 秒かかります。この時間は、元のバージョンのプログラムの場合よりも、約 38 分の 1 の速さです。

## 考慮事項

パフォーマンスは、複製を行うことによって改善されます。特定の項目を再利用しない場合、その項目をプールするためにリソースを無駄にしています。

ほとんどのアプリケーションには、コードのクリティカル・セクションが含まれています。一般には、アプリケーションは、コードの 10 から 20 % だけに対して、処理時間の 80 から 90 % を費やします。アプリケーションで 10,000 個の SQL ステートメントが使用される可能性がある場合、そのすべてがプールされるわけではありません。その目的は、アプリケーションのコードのクリティカル・セクションで使用される SQL ステートメントを識別してプールすることです。

Java インプリメンテーションでオブジェクトを作成すると、コストが非常に高く付く可能性があります。この点で、プーリング・ソリューションを使用することには利点があります。プロセスで使用されるオブジェクトは、開始時に作成されますが、これは他のユーザーがシステムを使用しようとする前です。これらのオブジェクトは、必要なときに再利用されます。パフォーマンスは優秀ですし、アプリケーションをきめ細かく徐々に調整するので、大勢のユーザーが使用できるようになります。結果として、さらに多くのオブジェクトがプールされます。さらに、アプリケーションのデータベース・アクセスをより効率的にマルチスレッド化することにより、より良いスループットを得ることができます。

Java (JDBC を使用した) は、動的 SQL をベースにしているため、遅くなる傾向があります。プーリングすることにより、この問題を最小限にすることができます。開始時にステートメントを準備することにより、データベースへのアクセスを静的に実現できます。ステートメントを準備した後は、動的 SQL と静的 SQL との間には、パフォーマンスの点でほとんど差はありません。

Java でのデータベース・アクセスのパフォーマンスは効率的になりますが、このことは、オブジェクト指向設計やコードの保守容易性を犠牲にすることなく実現できます。ステートメントおよび接続プーリングを構築するためにコードを作成することは難しくありません。さらに、コードを変更して拡張することで、複数のアプリケーションやアプリケーション・タイプ (Web ベース、クライアント/サーバー) などをサポートできるようになります。

## バッチ更新

バッチ更新サポートを使用することにより、データベースに対する任意の数の更新を、ユーザー・プログラムとデータベースの間の単一トランザクションとして渡すことができます。このプロシージャは、一度に多くの更新を実行しなければならないときに、パフォーマンスをかなり向上させることができます。

たとえば、ある大規模な会社で、新しい社員たちが月曜日から業務を開始しなければならない場合、これは社員データベースに対して、一度に多くの更新 (この場合は、挿入) を行うことが必要になります。更新するためのバッチを作成し、データベースにこれを 1 つの単位としてサブミットすれば、処理時間を節約することができます。

バッチ更新には、次の 2 つのタイプがあります。

- Statement オブジェクトを使用したバッチ更新。
- PreparedStatement オブジェクトを使用したバッチ更新。

### Statement バッチ更新:

Statement バッチ更新を実行するためには、自動コミットをオフにする必要があります。Java Database Connectivity (JDBC) では、デフォルトで自動コミットがオンになっています。自動コミットは、データベ

ースに対する更新のたびに、それぞれの SQL ステートメントの処理の後にコミットされることです。データベースへの処理のための複数のステートメントのグループを、機能的に 1 つのグループとしてまとめて扱いたい場合は、各ステートメントごとに個別にデータベースにコミットすることは望ましくありません。自動コミットをオフにせずにバッチの途中で失敗してしまった場合は、バッチ全体をロールバックすることができず、ステートメント全体を完了するためにバッチ処理を再度実行する必要があります。さらに、バッチ内の各ステートメントでコミットするという追加作業は、多大なオーバーヘッドを生み出します。

詳細については、トランザクションを参照してください。

自動コミットをオフにすると、標準の Statement オブジェクトが作成できます。executeUpdate のようなメソッドを使ってステートメントを処理する代わりに、そのステートメントを addBatch メソッドを使ってバッチに追加します。バッチに追加したいすべてのステートメントを追加したなら、executeBatch メソッドですべてのステートメントを処理することができます。clearBatch メソッドを使用すれば、いつでもバッチを空にすることができます。

以下に、これらのメソッドを使い方を示します。

#### 例: Statement バッチ更新

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
connection.setAutoCommit(false);
Statement statement = connection.createStatement();
statement.addBatch("INSERT INTO TABLEX VALUES(1, 'Cujo')");
statement.addBatch("INSERT INTO TABLEX VALUES(2, 'Fred')");
statement.addBatch("INSERT INTO TABLEX VALUES(3, 'Mark')");
int [] counts = statement.executeBatch();
connection.commit();
```

この例では、executeBatch メソッドから整数の配列が戻されます。この配列は、バッチ内で処理されたステートメントごとに 1 つの整数値を持っています。データベースへ値を挿入した場合は、その各ステートメントのこの値は 1 になります (これは、処理が成功したことを想定しています)。しかし、更新ステートメントのようないくつかのステートメントでは、影響が複数の行にわたることがあります。バッチ内に INSERT、UPDATE、または DELETE 以外のステートメントを加えた場合は、例外が発生します。

#### PreparedStatement バッチ更新:

PreparedStatement バッチは Statement バッチと似ていますが、PreparedStatement バッチは同じ準備済みステートメントを常に取り除くので、そのステートメントに対するパラメーターを変更するだけで済みます。

以下に、PreparedStatement バッチを使用した例を示します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
connection.setAutoCommit(false);
PreparedStatement statement =
    connection.prepareStatement("INSERT INTO TABLEX VALUES(?, ?)");
statement.setInt(1, 1);
statement.setString(2, "Cujo");
statement.addBatch();
statement.setInt(1, 2);
statement.setString(2, "Fred");
statement.addBatch();
statement.setInt(1, 3);
statement.setString(2, "Mark");
statement.addBatch();
int [] counts = statement.executeBatch();
connection.commit();
```



## BatchUpdateException:

バッチ更新の重要な考慮事項は、executeBatch メソッドの呼び出しが失敗したときに、どのようなアクションが取られるかということです。この場合、新しいタイプの例外である BatchUpdateException がスローされます。BatchUpdateException は SQLException のサブクラスで、普段、メッセージや SQLState、ベンダー・コードを受信するために呼び出している同様のメソッドのすべてを呼び出すことができます。

BatchUpdateException は、整数配列を戻す getUpdateCounts メソッドも提供しています。この整数配列には、バッチ内で障害が発生する時点までのすべてのステートメントによって処理された更新数が含まれています。配列の長さは、バッチのどのステートメントが失敗したのかを示します。たとえば、例外によって返された配列の長さが 3 であった場合は、バッチ内の 4 番目のステートメントが失敗したことを示しています。そのため、戻された単一の BatchUpdateException オブジェクトから、成功したすべてのステートメントの更新数、どのステートメントが失敗したのか、およびその障害に関するすべての情報を判別することができます。

バッチ更新の処理の標準的なパフォーマンスは、各ステートメントを別々に処理したときのパフォーマンスと同等です。バッチ更新の最適化サポートについて詳しくは、ブロック挿入サポートを参照してください。コードを記述する際は、将来のパフォーマンスの最適化の利点を得るため、現在でもこの新しいモデルを使用すべきです。

注: JDBC 2.1 仕様では、バッチ更新の例外条件を処理する方法として別のオプションが提供されています。JDBC 2.1 では、バッチ項目が失敗した後もバッチの処理を継続するモデルが導入されています。特殊更新数は、失敗した各項目から戻された整数の更新数の配列の中に格納されています。これにより、大規模なバッチの一部の項目が失敗したとしても、処理を継続することができます。この操作の 2 つのモードについて詳しくは、JDBC 2.1 または JDBC 3.0 の仕様を参照してください。デフォルトでは、ネイティブ JDBC ドライバーは JDBC 2.0 定義を使用します。ドライバーは、接続を確立するために DriverManager を使用するときに使われる Connection プロパティを提供しています。また、ドライバーは接続を確立するために DataSource を使用するときに使われる DataSource プロパティも提供しています。これらのプロパティを使うと、バッチ操作中に発生した障害にどのように処理するか、アプリケーションが選択することができます。

## ブロック挿入サポート:

iSeries 操作であるブロック挿入を使用して、データベース・テーブルに一度に複数の行を挿入することができます。

ブロック挿入は、iSeries サーバーの操作の特殊なタイプで、データベースのテーブルに一度に複数の行を挿入する、高度に最適化された方法を提供します。ブロック挿入は、バッチ更新のサブセットと考えることができます。バッチ更新は任意の形式で更新要求ができますが、ブロック挿入は指定された形式です。しかし、バッチ更新のブロック挿入タイプは共通です。ネイティブ JDBC ドライバーはこの機能の利点を得るために変更が加えられました。

ブロック挿入サポートを使用するときには生じるシステム制約事項により、ネイティブ JDBC ドライバーのデフォルト設定では、ブロック挿入は使用不可になっています。これは、Connection プロパティまたは DataSource プロパティを通して使用可能にすることができます。ブロック挿入を使用するときは、ユーザーの利益のためにそれらの制約事項の多くをチェックおよびハンドルすることができますが、いくつかの制約事項ではそれができません。これが、デフォルト設定ではブロック挿入サポートがオフになっている理由です。制約事項のリストは次のとおりです。

- SQL ステートメントでは、INSERT ステートメントは SUBSELECT と共にではなく、VALUES 文節と共に使用しなければなりません。JDBC ドライバーはこの制約を認識し、適切な処理方針を取ります。
- PreparedStatement を必ず使用しなければならず、これによって Statement オブジェクトの最適化サポートはなくなります。JDBC ドライバーはこの制約を認識し、適切な処理方針を取ります。
- SQL ステートメントは、テーブル内のすべての列に対するパラメーター・マーカーを指定しなければなりません。これにより、列に定数値を使用するか、データベースが挿入時に任意の列にデフォルト値を挿入できるようにするか、どちらかが使用できません。JDBC ドライバーは、SQL ステートメント内のパラメーター・マーカーの指定をテストするためのメカニズムを持っていません。最適化されたブロック挿入を実行するためにプロパティを設定し、SQL ステートメント内でデフォルト値や定数値の使用を差し控えなかった場合は、最終的なデータベース・テーブル内の値は不正なものになります。
- 接続はローカル・システムに対するものでなければなりません。ブロック挿入操作では DRDA がサポートされていないため、リモート・システムへアクセスするために DRDA を使った接続は使用できません。JDBC ドライバーは、ローカル・システムへの接続をテストするためのメカニズムを持っていません。最適化されたブロック挿入を実行するためのプロパティを設定し、リモート・システムへの接続を行おうとすると、バッチ更新の処理は失敗します。

以下のコード例は、ブロック挿入処理サポートを使用可能にする方法を示しています。このコードと、ブロック挿入サポートを使用していないバージョンとの違いは、接続 URL に use block insert=true が追加されているかどうかだけです。

#### 例: ブロック挿入処理

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Create a database connection
Connection c = DriverManager.getConnection("jdbc:db2:*local;use block insert=true");
BigDecimal bd = new BigDecimal("123456");

// Create a PreparedStatement to insert into a table with 4 columns
PreparedStatement ps =
    c.prepareStatement("insert into cujosql.xxx values(?, ?, ?, ?)");

// Start timing...
for (int i = 1; i <= 10000; i++) {
    ps.setInt(1, i);                // Set all the parameters for a row
    ps.setBigDecimal(2, bd);
    ps.setBigDecimal(3, bd);
    ps.setBigDecimal(4, bd);
    ps.addBatch();                //Add the parameters to the batch
}

// Process the batch
int[] counts = ps.executeBatch();

// End timing...
```

同様のテスト・ケースにおいては、ブロック挿入を使用しなかった場合に同様の処理をする場合より、ブロック挿入処理を行った場合のほうが数倍処理が速くなります。たとえば、前述のコードでのテストでは、ブロック挿入を使うと 9 倍の速度になりました。オブジェクトの代わりにプリミティブ・タイプのみを使ったケースでは、最大で 16 倍まで速くなりました。相当数の処理が行われているアプリケーションでは、期待できる効果もそれ相応のものになると考えられます。

## 拡張データ・タイプ

拡張 SQL3 データ・タイプでは、非常に幅広い柔軟性が提供されています。これは、シリアル化された Java オブジェクト、XML (Extensible Markup Language) 文書、および音楽、製品の画像、従業員の写真やムービー・クリップといったマルチメディア・データを格納するのに理想的です。Java Database Connectivity (JDBC) 2.0 およびそれ以降では、これらのデータ・タイプを SQL99 標準の一部として処理するためのサポートが提供されています。

## 特殊タイプ

特殊タイプは、標準データベース・タイプに基づくユーザー定義タイプです。たとえば、内部的に CHAR(9) の社会保障番号タイプ、SSN を定義できます。次の SQL ステートメントは、特殊タイプを作成します。

```
CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)
```

特殊タイプは常に、組み込みデータ・タイプにマップされます。SQL を使用する際、どのように、またいつ特殊タイプを使用するかについては、「SQL のリファレンス・マニュアル」を参照してください。

JDBC で特殊タイプを使用するには、その基となったタイプにアクセスする方法と同じ方法でアクセスします。新しいメソッドである `getUDTs` メソッドを使って、システム上でどの特殊タイプが使用可能かを調べることができます。163 ページの『例: 特殊タイプ』プログラムは、以下の点を示しています。

- 特殊タイプの作成。
- 特殊タイプを使ったテーブルの作成。
- 特殊タイプ・パラメーターを設定するための `PreparedStatement` の使用。
- 特殊タイプを戻すための `ResultSet` の使用。
- 特殊タイプについて調べるために `getUDTs` を呼び出すためのメタデータ・アプリケーション・プログラミング・インターフェース (API) の使用。

詳しくは、特殊タイプを使用して実行できるさまざまな共通タスクを示す以下の例を参照してください。

163 ページの『例: 特殊タイプ』

## ラージ・オブジェクト

ラージ・オブジェクト (LOB) には、3 つのタイプがあります。

- バイナリー・ラージ・オブジェクト (BLOB)
- 文字ラージ・オブジェクト (CLOB)
- 2 バイト文字ラージ・オブジェクト (DBCLOB)

DBCLOB は、文字データの内部記憶域表現ということを除けば、CLOB と同じです。Java および JDBC はすべての文字データを Unicode として外部化しますが、これは JDBC では CLOB でのみサポートされています。DBCLOB の動作は、JDBC の観点からすると、CLOB サポートと交換可能です。

## バイナリー・ラージ・オブジェクト

多くの場合、バイナリー・ラージ・オブジェクト (BLOB) 列は、大きなデータを格納できる CHAR FOR BIT DATA 列と同等です。これらの列は、変換されていないバイト・データのストリームと見なされ、どんなデータでも保管することができます。BLOB 列はしばしば、シリアル化された Java オブジェクト、ピクチャー、音楽、および他のバイナリー・データを保管するために使用されます。



BLOB は他の標準データベース・タイプと同じ方法で使用することができます。ストアード・プロシージャに渡したり、PreparedStatement 内で使用したり、ResultSet 内で更新することができます。

PreparedStatement クラスには BLOB をデータベースに渡すための setBlob メソッドがあり、ResultSet クラスは BLOB をデータベースから取得するための getBlob クラスが追加されています。BLOB は Java プログラムの中では、JDBC インターフェースの BLOB オブジェクトとして扱われます。

BLOB の使い方について詳しくは、BLOB を使ったコードを記述するを参照してください。

## 文字ラージ・オブジェクト

文字ラージ・オブジェクト (CLOB) は、BLOB に文字データを補足するものです。変換なしでデータベースにデータを保管するのではなく、データをテキストとしてデータベースに保管し、CHAR 列と同様の方法で処理されます。BLOB と同様に、JDBC 2.0 には CLOB と直接やり取りするための機能が提供されています。PreparedStatement インターフェースには setClob メソッドが含まれており、ResultSet インターフェースには getClob メソッドが含まれています。

CLOB の使い方について詳しくは、CLOB を使ったコードを記述するを参照してください。

BLOB および CLOB 列は CHAR FOR BIT DATA および CHAR 列と似た動作をしますが、これは外部からのユーザーの視点でどのように動作するかを概念的に示したものです。内部的には、これらは別物です。巨大なサイズになることもあり得るラージ・オブジェクト (LOB) 列では、データは一般的に間接的に処理されます。たとえば、データベースから行ブロックをフェッチしたときは、LOB のブロックを ResultSet に移動することはありません。その代わりに、LOB ロケーターと呼ばれるポインター (これは、4 バイトの整数) を ResultSet に移動します。しかし、JDBC 内で LOB を処理する際には、ロケーターについて知っておく必要はありません。

## データ・リンク

データ・リンクは、データベースからデータベース外に保管されたファイルへの論理参照を含んだ、カプセル化された値です。データ・リンクは、JDBC 2.0 かそれ以前を使用しているか、JDBC 3.0 かそれ以降を使用しているかによって、JDBC から 2 つの異なる方法で扱われ、使用されます。

データ・リンクの使い方について詳しくは、データ・リンクを使ったコードを記述するを参照してください。

## サポートされていない SQL3 データ・タイプ

この他にも、既に定義され、JDBC API によってサポートが提供されている SQL3 データ・タイプがあります。ARRAY、REF、および STRUCT です。現在のところ、iSeries サーバーでこれらのタイプはサポートされていません。そのため、JDBC ドライバーはそれらタイプに対するいかなる形式のサポートも提供していません。

### BLOB を使ったコードを記述する:

Java Database Connectivity (JDBC) アプリケーション・プログラミング・インターフェース (API) を介し、データベースのバイナリー・ラージ・オブジェクト (BLOB) 列を使って達成できる、数多くのタスクがあります。以下のトピックでは、これらのタスクについて簡単に説明し、その使い方の例を示します。

## データベースからの BLOB の読み取り、およびデータベースへの BLOB の挿入

JDBC API では、データベースからの BLOB の取り出し、およびデータベースへの BLOB の書き込みにはいくつかの方法があります。しかし、BLOB オブジェクトを作成するための標準化された方法はありま

せん。これはデータベースが BLOB を完全に利用できる場合は問題ではありませんが、JDBC 経由で最初から BLOB を処理したい場合に問題を引き起こす可能性があります。JDBC API の BLOB および CLOB インターフェイス用のコンストラクターを定義する代わりに、他のタイプとしてデータベースに BLOB を直接格納したり、BLOB をデータベースから直接取り出すサポートが提供されています。たとえば、setBinaryStream メソッドを使うと、データベース内の BLOB タイプの列を処理できます。『例: BLOB』では、データベースに BLOB を書き込んだり、データベースから BLOB を取り出したりするための一般的な方法のいくつかが示されています。

## BLOB オブジェクト API の処理

BLOB は JDBC の中で、数多くのドライバーによってインプリメンテーションが提供されたインターフェースとして定義されています。このインターフェースには、BLOB オブジェクトと対話するために使用できる一連のメソッドがあります。157 ページの『例: BLOB の使用』では、この API を使って実行できる一般的なタスクのいくつかが示されています。BLOB オブジェクトで使用できるすべてのメソッドのリストは、JDBC Javadoc を調べてください。

## BLOB の更新のために JDBC 3.0 サポートを使用する

JDBC 3.0 では、LOB オブジェクトへ変更を加える機能がサポートされています。これらの変更は、データベース内の BLOB 列に保管することができます。156 ページの『例: BLOB の更新』では、JDBC 3.0 の BLOB サポートを使って実行できる一般的なタスクのいくつかが示されています。

### 例: BLOB:

以下は、BLOB をデータベースに書き込んだり、データベースから検索したりする方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
// PutGetBlobs is an example application
// that shows how to work with the JDBC
// API to obtain and put BLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two BLOB values
// in a new table. Both are identical
// and contain 500k of random byte
// data.
////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any previous run of this application.
```

```

try {
    s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
} catch (SQLException e) {
    // Ignore it - assume the table did not exist.
}

// Create a table with a BLOB column. The default BLOB column
// size is 1 MB.
s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

// Create a PreparedStatement object that allows you to put
// a new Blob object into the database.
PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

// Create a big BLOB value...
Random random = new Random ();
byte [] inByteArray = new byte[500000];
random.nextBytes (inByteArray);

// Set the PreparedStatement parameter. Note: This is not
// portable to all JDBC drivers. JDBC drivers do not have
// support when using setBytes for BLOB columns. This is used to
// allow you to generate new BLOBs. It also allows JDBC 1.0
// drivers to work with columns containing BLOB data.
ps.setBytes(1, inByteArray);

// Process the statement, inserting the BLOB into the database.
ps.executeUpdate();

// Process a query and obtain the BLOB that was just inserted out
// of the database as a Blob object.
ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
rs.next();
Blob blob = rs.getBlob(1);

// Put that Blob back into the database through
// the PreparedStatement.
ps.setBlob(1, blob);
ps.execute();

c.close(); // Connection close also closes stmt and rs.
}
}

```

### 例: BLOB の更新:

以下は、アプリケーション中で BLOB を更新する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UpdateBlobs is an example application
// that shows some of the APIs providing
// support for changing Blob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetBlobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
    public static void main(String[] args)

```

```

throws SQLException
{
    // Register the native JDBC driver.
    try {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    } catch (Exception e) {
        System.exit(1); // Setup error.
    }

    Connection c = DriverManager.getConnection("jdbc:db2:*local");
    Statement s = c.createStatement();

    ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

    rs.next();
    Blob blob1 = rs.getBlob(1);
    rs.next();
    Blob blob2 = rs.getBlob(1);

    // Truncate a BLOB.
    blob1.truncate((long) 150000);
    System.out.println("Blob1's new length is " + blob1.length());

    // Update part of the BLOB with a new byte array.
    // The following code obtains the bytes that are at
    // positions 4000-4500 and set them to positions 500-1000.

    // Obtain part of the BLOB as a byte array.
    byte[] bytes = blob1.getBytes(4000L, 4500);

    int bytesWritten = blob2.setBytes(500L, bytes);

    System.out.println("Bytes written is " + bytesWritten);

    // The bytes are now found at position 500 in blob2
    long startInBlob2 = blob2.position(bytes, 1);

    System.out.println("pattern found starting at position " + startInBlob2);

    c.close(); // Connection close also closes stmt and rs.
}
}

```

### 例: BLOB の使用:

以下は、アプリケーション中で BLOB を使用する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UseBlobs is an example application
// that shows some of the APIs associated
// with Blob objects.
//
// This program must be run after
// the PutGetBlobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UseBlobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.

```

```

    try {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    } catch (Exception e) {
        System.exit(1); // Setup error.
    }

    Connection c = DriverManager.getConnection("jdbc:db2:*local");
    Statement s = c.createStatement();

    ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

    rs.next();
    Blob blob1 = rs.getBlob(1);
    rs.next();
    Blob blob2 = rs.getBlob(1);

    // Determine the length of a LOB.
    long end = blob1.length();
    System.out.println("Blob1 length is " + blob1.length());

    // When working with LOBs, all indexing that is related to them
    // is 1-based, and is not 0-based like strings and arrays.
    long startingPoint = 450;
    long endingPoint = 500;

    // Obtain part of the BLOB as a byte array.
    byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

    // Find where a sub-BLOB or byte array is first found within a
    // BLOB. The setup for this program placed two identical copies of
    // a random BLOB into the database. Thus, the start position of the
    // byte array extracted from blob1 can be found in the starting
    // position in blob2. The exception would be if there were 50
    // identical random bytes in the LOBs previously.
    long startInBlob2 = blob2.position(outByteArray, 1);

    System.out.println("pattern found starting at position " + startInBlob2);

    c.close(); // Connection close closes stmt and rs too.
}
}

```

### CLOB を使ったコードを記述する:

Java Database Connectivity (JDBC) アプリケーション・プログラミング・インターフェース (API) を介し、データベースの CLOB および DBCLOB 列を使って達成できる、数多くのタスクがあります。以下のトピックでは、これらのタスクについて簡単に説明し、その使い方の例を示します。

## データベースからの CLOB の読み取り、およびデータベースへの CLOB の挿入

JDBC API では、データベースからの CLOB の取り出し、およびデータベースへの CLOB の書き込みにはいくつかの方法があります。しかし、CLOB オブジェクトを作成するための標準化された方法はありません。これはデータベースが CLOB を完全に利用できる場合は問題ではありませんが、JDBC 経由で最初から CLOB を処理したい場合に問題を引き起こす可能性があります。JDBC API の BLOB および CLOB インターフェース用のコンストラクターを定義する代わりに、他のタイプとしてデータベースに CLOB を直接格納したり、CLOB をデータベースから直接取り出すサポートが提供されています。たとえば、setCharacterStream メソッドを使うと、データベース内の CLOB タイプの列を処理できます。159 ページの『例: CLOB』では、データベースに CLOB を書き込んだり、データベースから CLOB を取り出したりするための一般的な方法のいくつかが示されています。

## CLOB オブジェクト API の処理

CLOB は JDBC の中で、数多くのドライバーによってインプリメンテーションが提供されたインターフェースとして定義されています。このインターフェースには、CLOB オブジェクトと対話するために使用できる一連のメソッドがあります。この例では、この API を使って実行できる一般的なタスクのいくつかが表示されています。CLOB オブジェクトで使用できるすべてのメソッドのリストは、JDBC Javadoc を調べてください。

## CLOB の更新のために JDBC 3.0 サポートを使用する

JDBC 3.0 では、LOB オブジェクトへ変更を加える機能がサポートされています。これらの変更は、データベース内の CLOB 列に保管することができます。この例では、JDBC 3.0 の CLOB サポートを使って実行できる一般的なタスクのいくつかが表示されています。

### 例: CLOB:

以下は、CLOB をデータベースに書き込んだり、データベースから検索したりする方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
// PutGetClobs is an example application
// that shows how to work with the JDBC
// API to obtain and put CLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two CLOB values
// in a new table. Both are identical
// and contain about 500k of repeating
// text data.
////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any previous run of this application.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
        } catch (SQLException e) {
            // Ignore it - assume the table did not exist.
        }

        // Create a table with a CLOB column. The default CLOB column
        // size is 1 MB.
        s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

        // Create a PreparedStatement object that allow you to put
```

```

// a new Clob object into the database.
PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

// Create a big CLOB value...
StringBuffer buffer = new StringBuffer(500000);
while (buffer.length() < 500000) {
    buffer.append("All work and no play makes Cujo a dull boy.");
}
String clobValue = buffer.toString();

// Set the PreparedStatement parameter. This is not
// portable to all JDBC drivers. JDBC drivers do not have
// to support setBytes for CLOB columns. This is done to
// allow you to generate new CLOBs. It also
// allows JDBC 1.0 drivers a way to work with columns containing
// Clob data.
ps.setString(1, clobValue);

// Process the statement, inserting the clob into the database.
ps.executeUpdate();

// Process a query and get the CLOB that was just inserted out of the
// database as a Clob object.
ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
rs.next();
Clob clob = rs.getClob(1);

// Put that Clob back into the database through
// the PreparedStatement.
ps.setClob(1, clob);
ps.execute();

c.close(); // Connection close also closes stmt and rs.
}
}

```

### 例: CLOB の更新:

以下は、アプリケーション中で CLOB を更新する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetClobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }
    }
}

```



```

Connection c = DriverManager.getConnection("jdbc:db2:*local");
Statement s = c.createStatement();

ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

rs.next();
Clob clob1 = rs.getClob(1);
rs.next();
Clob clob2 = rs.getClob(1);

// Truncate a CLOB.
clob1.truncate((long) 150000);
System.out.println("Clob1's new length is " + clob1.length());

// Update a portion of the CLOB with a new String value.
String value = "Some new data for once";
int charsWritten = clob2.setString(500L, value);

System.out.println("Characters written is " + charsWritten);

// The bytes can be found at position 500 in clob2
long startInClob2 = clob2.position(value, 1);

System.out.println("pattern found starting at position " + startInClob2);

c.close(); // Connection close also closes stmt and rs.
}
}

```

#### 例: CLOB の使用:

以下は、アプリケーション中で CLOB を使用方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetClobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UseClobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

```

```

rs.next();
Clob clob1 = rs.getClob(1);
rs.next();
Clob clob2 = rs.getClob(1);

// Determine the length of a LOB.
long end = clob1.length();
System.out.println("Clob1 length is " + clob1.length());

// When working with LOBs, all indexing that is related to them
// is 1-based, and not 0-based like strings and arrays.
long startingPoint = 450;
long endingPoint = 50;

// Obtain part of the CLOB as a byte array.
String outString = clob1.getSubString(startingPoint, (int)endingPoint);
System.out.println("Clob substring is " + outString);

// Find where a sub-CLOB or string is first found within a
// CLOB. The setup for this program placed two identical copies of
// a repeating CLOB into the database. Thus, the start position of the
// string extracted from clob1 can be found in the starting
// position in clob2 if the search begins close to the position where
// the string starts.
long startInClob2 = clob2.position(outString, 440);

System.out.println("pattern found starting at position " + startInClob2);

c.close(); // Connection close also closes stmt and rs.
}
}

```

#### データ・リンクを使ったコードを記述する:

データ・リンクをどのように使って処理するかどうかは、どのリリースを使用して処理するかに依存しています。JDBC 3.0 では、`getURL` および `putURL` メソッドを使って、データ・リンク列を直接処理する機能がサポートされています。

以前のバージョンの JDBC の場合は、ストリング列のようにデータ・リンク列を処理しなければなりません。現在のところ、データベースのデータ・リンクと文字データ・タイプの自動変換はサポートされていません。その結果として、SQL ステートメント内である型キャストを実行する必要があります。

#### 例: *Datalink*:

以下は、アプリケーションでの *Datalink* の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// PutGetDatalinks is an example application
// that shows how to use the JDBC
// API to handle datalink database columns.
////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
        throws SQLException
    {

```

```

// Register the native JDBC driver.
try {
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
} catch (Exception e) {
    System.exit(1); // Setup error.
}

// Establish a Connection and Statement with which to work.
Connection c = DriverManager.getConnection("jdbc:db2:*local");
Statement s = c.createStatement();

// Clean up any previous run of this application.
try {
    s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
} catch (SQLException e) {
    // Ignore it - assume the table did not exist.
}

// Create a table with a datalink column.
s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

// Create a PreparedStatement object that allows you to add
// a new datalink into the database. Since conversing
// to a datalink cannot be accomplished directly in the database, you
// can code the SQL statement to perform the explicit conversion.
PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

// Set the datalink. This URL points you to an article about
// the new features of JDBC 3.0.
ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

// Process the statement, inserting the CLOB into the database.
ps.executeUpdate();

// Process a query and obtain the CLOB that was just inserted out of the
// database as a Clob object.
ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
rs.next();
String datalink = rs.getString(1);

// Put that datalink value into the database through
// the PreparedStatement. Note: This function requires JDBC 3.0
// support.
/*
try {
    URL url = new URL(datalink);
    ps.setURL(1, url);
    ps.execute();
} catch (MalformedURLException mue) {
    // Handle this issue here.
}

rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
rs.next();
URL url = rs.getURL(1);
System.out.println("URL value is " + url);
*/

c.close(); // Connection close also closes stmt and rs.
}
}

```

例: 特殊タイプ:

以下に、特殊タイプの使用法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
// This example program shows examples of
// various common tasks that can be done
// with distinct types.
////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any old runs.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        // Create the type, create the table, and insert a value.
        s.executeUpdate("CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)");
        s.executeUpdate("CREATE TABLE CUJOSQL.SERIALNOS (COL1 CUJOSQL.SSN)");

        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.SERIALNOS VALUES(?)");
        ps.setString(1, "399924563");
        ps.executeUpdate();
        ps.close();

        // You can obtain details about the types available with new metadata in
        // JDBC 2.0
        DatabaseMetaData dmd = c.getMetaData();

        int types[] = new int[1];
        types[0] = java.sql.Types.DISTINCT;

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN", types);
        rs.next();
        System.out.println("Type name " + rs.getString(3) +
            " has type " + rs.getString(4));

        // Access the data you have inserted.
        rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
        rs.next();
        System.out.println("The SSN is " + rs.getString(1));
    }
}
```

```
        c.close(); // Connection close also closes stmt and rs.
    }
}
```

## RowSet

RowSet は、元は Java Database Connectivity (JDBC) 2.0 Optional Package に追加されていました。もっとよく知られたいくつかの JDBC 仕様のインターフェースとは異なり、RowSet 仕様は、実際のインプリメンテーションの仕様というよりは、フレームワークの仕様として設計されています。RowSet インターフェースは、すべての RowSets に含まれているコア機能のセットを定義します。RowSet インプリメンテーションのプロバイダーは、特定の問題スペースでのその必要を満たすために必要な機能をかなり自由に定義できます。

### RowSet の特性:

RowSet によって特定のプロパティの条件が満たされるよう要求できます。共通プロパティには、結果の RowSet によってサポートされるインターフェースのセットが含まれます。

## RowSet は ResultSet

RowSet インターフェースは ResultSet インターフェースを拡張するものです。このことは、RowSet には ResultSet が実行できるすべての機能を実行する能力があるということを意味します。たとえば、RowSet はスクロールと更新が可能です。

## RowSet はデータベースから切断可能

RowSet には、以下の 2 つのカテゴリがあります。

- **接続** 接続 RowSet は、データが読み込まれている間、基となるデータベースとの内部接続を常に開いており、ResultSet のインプリメンテーションのラッパーとして機能します。
- **切断** 切断 RowSet は、常にそのデータ・ソースへの接続を保持している必要はありません。切断 RowSet は、データベースから切り離してさまざまな用途に使用し、その後、加えられた変更を反映するためにデータベースに再接続することができます。

## RowSets は JavaBeans™ のコンポーネント

RowSet には、JavaBeans のイベント処理モデルに基づくイベント処理のサポートがあります。これらには、設定できるプロパティもあります。これらのプロパティは、RowSet が以下のことを実行するとき使用されます。

- データベースへの接続を確立する。
- SQL ステートメントを処理する。
- RowSet が表すデータのフィーチャーを判別し、RowSet オブジェクトの内部フィーチャーを処理する。

## RowSet は逐次化可能

RowSet は、ネットワーク接続上をフローできるように逐次化および並列化したり、フラット・ファイル (つまり、ワード・プロセッシングや他の構造文字をもたないテキスト文書) に書き込んだりすることができます。

### DB2CachedRowSet:

DB2CachedRowSet オブジェクトは切断された RowSet で、データベースに接続せずに使用できることを意味します。そのインプリメンテーションは、CachedRowSet の記述に厳密に従っています。

DB2CachedRowSet は ResultSet のデータ行のためのコンテナです。DB2CachedRowSet はそのデータを保持しており、明示的にデータをデータベースから読み込んだり、書き込んだりするときでなければ、データベースへの接続を保つ必要がありません。

### **DB2CachedRowSet を使用する:**

DB2CachedRowSet オブジェクトは切断および逐次化することができるため、全体の JDBC ドライバーを動作させることが必ずしも常に実際的でない環境 (たとえば、PDA および Java が使用できる携帯電話など) で有用です。

DB2CachedRowSet オブジェクトはメモリー内に格納され、そのデータは既に取得されているため、アプリケーションに対して、スクロール可能な ResultSet の高度に最適化された形式として提供できます。しかし、スクロール可能な DB2 ResultSet はしばしば、パフォーマンス面での弱点となります。それは、ランダムな移動が JDBC ドライバーのデータの行をキャッシュする機能と干渉してしまうためです。RowSet にはこの問題はありませぬ。

DB2CachedRowSet には、新しい RowSet を作成する 2 つのメソッドが提供されています。

- createCopy メソッドは、コピーされた同一の新しい RowSet を作成します。
- createShared メソッドは、オリジナルと同一の基礎データを共有する新しい RowSet を作成します。

クライアントに共通の ResultSet を配布するには、createCopy メソッドを使用できます。テーブル・データが変更されない場合、RowSet のコピーを作成して各クライアントに配布することは、毎回データベースに対して照会を実行するよりも効率的です。

createShared メソッドを使うと、同一のデータに複数のユーザーがアクセスできるようにしてデータベースのパフォーマンスを向上させることができます。たとえば、顧客が接続したとき、ホーム・ページ上で上位 20 位のベストセラーの商品を表示する Web サイトを想定してみましょ。メイン・ページ上の情報は定期的に更新する必要がありますが、顧客がメイン・ページを訪問するたびに良く売れている商品を取り出す照会を実行するのは実際的ではありません。createShared メソッドを使うと、何度も照会を処理したり、膨大な量の情報をメモリーに保管しておくことなく、事実上、各顧客用の「カーソル」を作成することができます。必要があれば、良く売れている商品を検索する照会を再度実行することもできます。共有カーソルを作成するために使用した RowSet に新しいデータを取り込み、サブレットがこれらを利用できます。

DB2CachedRowSets は遅延処理機能を提供しています。この機能を利用すると、複数の照会要求をグループ化し、データベースに対する 1 つの要求として処理することができます。DB2CachedRowSets を使用するには、これを利用しない場合に存在するデータベースの計算負荷の、いくらかを軽減する例が示されています。

RowSet は、データベースにデータを戻すため、また、加えた変更を元に戻したり、加えられたすべての変更を表示する機能をサポートするために、加えられた変更を注意深く追跡する必要があります。たとえば、RowSet に対して削除された行をフェッチするための showDeleted メソッドがあります。また、cancelRowInsert および cancelRowDelete メソッドは、ユーザーが行の挿入や削除を行った後、きちんと元に戻します。

DB2CachedRowSet オブジェクトは、イベント処理サポート、および RowSet またはその一部を Java コレクションに変換するための toCollection メソッドにより、他の Java API との良好な相互運用性を提供しています。

DB2CachedRowSet のイベント処理サポートは、グラフィカル・ユーザー・インターフェイス (GUI) アプリケーションの表示制御や、RowSet に加えられた変更の情報のロギング、または RowSet 以外のソースに加えられた変更に関する情報の検索などに使用できます。詳細については、例: DB2JdbcRowSet イベントを参照してください。

DB2CachedRowSet の個々の機能の詳細については、下記のトピックを参照してください。

- DB2CachedRowSet の作成とデータ取り込み
- DB2CachedRowSet データへのアクセスおよびカーソル操作
- DB2CachedRowSet データを変更し、データ・ソースに変更を反映する
- その他の DB2CachedRowSet の機能

イベント・モデルおよびイベント処理については、DB2JdbcRowSet を参照してください。このサポートは、どちらのタイプの RowSet でも同様に動作します。

### **DB2CachedRowSet の作成とデータ取り込み:**

DB2CachedRowSet にデータを入れるには、次のような方法があります。

### **populate メソッドを使用する**

DB2CachedRowSets には、DB2 ResultSet オブジェクトから RowSet にデータを書き込むための populate メソッドがあります。以下に、この方法の例を示します。

**例:** populate メソッドを使用する

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Establish a connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a statement and use it to perform a query.
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");

// Create and populate a DB2CachedRowSet from it.
DB2CachedRowSet crs = new DB2CachedRowSet();
crs.populate(rs);

// Note: Disconnect the ResultSet, Statement,
// and Connection used to create the RowSet.
rs.close();
stmt.close();
conn.close();

// Loop through the data in the RowSet.
while (crs.next()) {
    System.out.println("v1 is " + crs.getString(1));
}

crs.close();
```

### **DB2CachedRowSet プロパティと DataSources を使用する**

DB2CachedRowSets には、DB2CachedRowSet が SQL 照会と DataSource 名を受け取るためのプロパティがあります。SQL 照会と DataSource 名を使って、それ自身のデータを作成することができます。以下



に、この方法の例を示します。 `BaseDataSource` という名前の `DataSource` への参照が、事前に有効な `DataSource` としてセットアップされていることを想定しています。

**例:** `DB2CachedRowSet` プロパティと `DataSources` を使用する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for
// the RowSet to use a DataSource to populate itself.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the DataSource and SQL query
// specified to populate itself with data. Once
// the RowSet is populated, it disconnects from the database.
crs.execute();

// Loop through the data in the RowSet.
while (crs.next()) {
    System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

## **DB2CachedRowSet プロパティと JDBC URL を使用する**

`DB2CachedRowSet` には、`DB2CachedRowSet` が SQL 照会と JDBC URL を受け取るためのプロパティがあります。照会と JDBC URL を使って、それ自身のデータを作成することができます。以下に、この方法の例を示します。

**例:** `DB2CachedRowSet` プロパティおよび JDBC URL を使用する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for
// the RowSet to use a JDBC URL to populate itself.
crs.setUrl("jdbc:db2:*local");
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the DataSource and SQL query
// specified to populate itself with data. Once
// the RowSet is populated, it disconnects from the database.
crs.execute();

// Loop through the data in the RowSet.
while (crs.next()) {
    System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

## setConnection(Connection) メソッドを使って、既存のデータベース接続を使用する

JDBC Connection オブジェクトの再利用を促進するため、DB2CachedRowSet には、確立された Connection オブジェクトを、DB2CachedRowset (RowSet にデータを取り込むために使用される) に渡すメカニズムがあります。ユーザーが提供した Connection オブジェクトが渡されると、DB2CachedRowSet は取り込みが完了しても切断しません。

**例:** setConnection(Connection) メソッドを使って、既存のデータベース接続を使用する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Establish a JDBC connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for the
// RowSet to use an already connected connection
// to populate itself.
crs.setConnection(conn);
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the connection that it was provided
// with previously. Once the RowSet is populated, it does not
// close the user-supplied connection.
crs.execute();

// Loop through the data in the RowSet.
while (crs.next()) {
    System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

## execute(Connection) メソッドを使って、既存のデータベース接続を使用する

JDBC Connection オブジェクトの再利用を促進するため、DB2CachedRowSet には、確立された Connection オブジェクトを、メソッドが呼び出されたときに DB2CachedRowset に渡すメカニズムがあります。ユーザーが提供した Connection オブジェクトが渡されると、DB2CachedRowSet は取り込みが完了しても切断しません。

**例:** execute(Connection) メソッドを使って、既存のデータベース接続を使用する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Establish a JDBC connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the SQL statement that is to be used to
// populate the RowSet.
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method, passing in the connection
// that should be used. Once the Rowset is populated, it does not
// close the user-supplied connection.
crs.execute(conn);
```

```
// Loop through the data in the RowSet.
while (crs.next()) {
    System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

## execute(int) メソッドを使って、データベース要求をグループ化する

データベースのワークロードを軽減させるため、DB2CachedRowSet には、複数の SQL ステートメントをいくつかのスレッドでグループ化し、1 つの処理としてまとめてデータベースに要求するメカニズムがあります。

**例:** execute(int) メソッドを使って、データベース要求をグループ化する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Create a new DB2CachedRowSet
DB2CachedRowSet crs = new DB2CachedRowSet();

// Set the properties that are needed for
// the RowSet to use a DataSource to populate itself.
crs.setDataSourceName("BaseDataSource");
crs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the DataSource and SQL query
// specified to populate itself with data. Once
// the RowSet is populated, it disconnects from the database.
// This version of the execute method accepts the number of seconds
// that it is willing to wait for its results. By
// allowing a delay, the RowSet can group the requests
// of several users and only process the request against
// the underlying database once.
crs.execute(5);

// Loop through the data in the RowSet.
while (crs.next()) {
    System.out.println("v1 is " + crs.getString(1));
}

// Eventually, close the RowSet.
crs.close();
```

### **DB2CachedRowSet データへのアクセスおよびカーソル操作:**

このトピックでは、DB2CachedRowSet データへのアクセス、およびさまざまなカーソル操作機能についての情報を記載しています。

RowSet は ResultSet メソッドに依存しています。DB2CachedRowSet データ・アクセスやカーソルの移動などの多くの操作は、アプリケーション・レベルでは ResultSet の場合も RowSet の場合も違いはありません。

### **DB2CachedRowSet データへのアクセス**

RowSet と ResultSet は同じ方式でデータにアクセスします。以下の例ではプログラムで JDBC を使用し、テーブルを作成して、さまざまなデータ・タイプのデータを取り込みます。テーブルが準備されると、DB2CachedRowSet が作成され、テーブル内の情報が取り込まれます。この例では、RowSet クラスのさまざまなメソッドも使用されています。

## 例: DB2CachedRowSet データへのアクセス

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;
import java.math.*;

public class TestProgram
{
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                ex.getMessage());
            // No need to go any further.
            System.exit(1);
        }

        try {
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");

            Statement stmt = conn.createStatement();

            // Clean up previous runs
            try {
                stmt.execute("drop table cujosql.test_table");
            }
            catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create test table
            stmt.execute("Create table cujosql.test_table (col1 smallint, col2 int, " +
                "col3 bigint, col4 real, col5 float, col6 double, col7 numeric, " +
                "col8 decimal, col9 char(10), col10 varchar(10), col11 date, " +
                "col12 time, col13 timestamp)");
            System.out.println("Table created.");

            // Insert some test rows
            stmt.execute("insert into cujosql.test_table values (1, 1, 1, 1.5, 1.5, 1.5, 1.5, 1.5, 'one', 'one',
                {d '2001-01-01'}, {t '01:01:01'}, {ts '1998-05-26 11:41:12.123456'})");

            stmt.execute("insert into cujosql.test_table values (null, null, null, null, null, null, null, null,
                null, null, null, null, null)");
            System.out.println("Rows inserted");

            ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
            System.out.println("Query executed");

            // Create a new rowset and populate it...
            DB2CachedRowSet crs = new DB2CachedRowSet();
            crs.populate(rs);
            System.out.println("RowSet populated.");

            conn.close();
            System.out.println("RowSet is detached...");

            System.out.println("Test with getObject");
            int count = 0;
            while (crs.next()) {
```

```

        System.out.println("Row " + (++count));
        for (int i = 1; i <= 13; i++) {
            System.out.println(" Col " + i + " value " + crs.getObject(i));
        }
    }

    System.out.println("Test with getXXX... ");
    crs.first();
    System.out.println("Row 1");
    System.out.println(" Col 1 value " + crs.getShort(1));
    System.out.println(" Col 2 value " + crs.getInt(2));
    System.out.println(" Col 3 value " + crs.getLong(3));
    System.out.println(" Col 4 value " + crs.getFloat(4));
    System.out.println(" Col 5 value " + crs.getDouble(5));
    System.out.println(" Col 6 value " + crs.getDouble(6));
    System.out.println(" Col 7 value " + crs.getBigDecimal(7));
    System.out.println(" Col 8 value " + crs.getBigDecimal(8));
    System.out.println(" Col 9 value " + crs.getString(9));
    System.out.println(" Col 10 value " + crs.getString(10));
    System.out.println(" Col 11 value " + crs.getDate(11));
    System.out.println(" Col 12 value " + crs.getTime(12));
    System.out.println(" Col 13 value " + crs.getTimestamp(13));
    crs.next();
    System.out.println("Row 2");
    System.out.println(" Col 1 value " + crs.getShort(1));
    System.out.println(" Col 2 value " + crs.getInt(2));
    System.out.println(" Col 3 value " + crs.getLong(3));
    System.out.println(" Col 4 value " + crs.getFloat(4));
    System.out.println(" Col 5 value " + crs.getDouble(5));
    System.out.println(" Col 6 value " + crs.getDouble(6));
    System.out.println(" Col 7 value " + crs.getBigDecimal(7));
    System.out.println(" Col 8 value " + crs.getBigDecimal(8));
    System.out.println(" Col 9 value " + crs.getString(9));
    System.out.println(" Col 10 value " + crs.getString(10));
    System.out.println(" Col 11 value " + crs.getDate(11));
    System.out.println(" Col 12 value " + crs.getTime(12));
    System.out.println(" Col 13 value " + crs.getTimestamp(13));

    crs.close();
}
catch (Exception ex) {
    System.out.println("SQLException: " + ex.getMessage());
    ex.printStackTrace();
}
}
}
}

```

## カーソル操作

RowSet はスクロール可能で、その動作はスクロール可能な ResultSet と同じです。以下の例ではプログラムで JDBC を使用し、テーブルを作成して、データを取り込みます。テーブルが準備されると、DB2CachedRowSet オブジェクトが作成され、テーブル内の情報が取り込まれます。この例では、さまざまなカーソル操作機能も使用されています。

### 例: カーソル操作

```

import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample1
{
    public static void main(String args[])
    {
        // Register the driver.

```

```

try {
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
}
catch (ClassNotFoundException ex) {
    System.out.println("ClassNotFoundException: " +
        ex.getMessage());
    // No need to go any further.
    System.exit(1);
}

try {
    Connection conn = DriverManager.getConnection("jdbc:db2:*local");

    Statement stmt = conn.createStatement();

    // Clean up previous runs
    try {
        stmt.execute("drop table cujosql.test_table");
    }
    catch (SQLException ex) {
        System.out.println("Caught drop table: " + ex.getMessage());
    }

    // Create a test table
    stmt.execute("Create table cujosql.test_table (coll smallint)");
    System.out.println("Table created.");

    // Insert some test rows
    for (int i = 0; i < 10; i++) {
        stmt.execute("insert into cujosql.test_table values (" + i + ")");
    }
    System.out.println("Rows inserted");

    ResultSet rs = stmt.executeQuery("select coll from cujosql.test_table");
    System.out.println("Query executed");

    // Create a new rowset and populate it...
    DB2CachedRowSet crs = new DB2CachedRowSet();
    crs.populate(rs);
    System.out.println("RowSet populated.");

    conn.close();
    System.out.println("RowSet is detached...");

    System.out.println("Use next()");
    while (crs.next()) {
        System.out.println("v1 is " + crs.getShort(1));
    }

    System.out.println("Use previous()");
    while (crs.previous()) {
        System.out.println("value is " + crs.getShort(1));
    }

    System.out.println("Use relative()");
    crs.next();
    crs.relative(9);
    System.out.println("value is " + crs.getShort(1));

    crs.relative(-9);
    System.out.println("value is " + crs.getShort(1));

    System.out.println("Use absolute()");
    crs.absolute(10);
    System.out.println("value is " + crs.getShort(1));
    crs.absolute(1);
    System.out.println("value is " + crs.getShort(1));
}

```

```

    crs.absolute(-10);
    System.out.println("value is " + crs.getShort(1));
    crs.absolute(-1);
    System.out.println("value is " + crs.getShort(1));

    System.out.println("Test beforeFirst()");
    crs.beforeFirst();
    System.out.println("isBeforeFirst is " + crs.isBeforeFirst());
    crs.next();
    System.out.println("move one... isFirst is " + crs.isFirst());

    System.out.println("Test afterLast()");
    crs.afterLast();
    System.out.println("isAfterLast is " + crs.isAfterLast());
    crs.previous();
    System.out.println("move one... isLast is " + crs.isLast());

    System.out.println("Test getRow()");
    crs.absolute(7);
    System.out.println("row should be (7) and is " + crs.getRow() +
        " value should be (6) and is " + crs.getShort(1));

    crs.close();
}
catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
}
}
}

```

### **DB2CachedRowSet データを変更し、データ・ソースに変更を反映する:**

このトピックでは、DB2CachedRowSet 内の行に変更を行い、その後の基礎データベースの更新に関する情報を提供します。

DB2CachedRowSet では、RowSet オブジェクト内のデータを変更するための標準 ResultSet インターフェースと同じメソッドを使用します。アプリケーション・レベルでは、RowSet のデータを変更することと、ResultSet のデータを変更することには違いがありません。DB2CachedRowSet は acceptChanges メソッドを提供しており、このメソッドは RowSet に加えられた変更をデータ元のデータベースに反映するために使用します。

### **DB2CachedRowSet 内の行を削除、挿入、および更新する**

DB2CachedRowSet は更新可能です。以下の例ではプログラムで JDBC を使用し、テーブルを作成して、データを取り込みます。テーブルが準備されると、DB2CachedRowSet が作成され、テーブル内の情報が取り込まれます。この例では、RowSet を更新するために使用できる多くのメソッドを使っており、またアプリケーションが削除された後の行をフェッチできるようにする showDeleted プロパティの使い方を示しています。さらにこの例では、行の挿入および削除を元に戻すことのできる cancelRowInsert および cancelRowDelete メソッドの使い方も示されています。

**例:** DB2CachedRowSet 内の行を削除、挿入、および更新する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample2
{
    public static void main(String args[])

```



```

{
    // Register the driver.
    try {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
    }
    catch (ClassNotFoundException ex) {
        System.out.println("ClassNotFoundException: " +
            ex.getMessage());

        // No need to go any further.
        System.exit(1);
    }

    try {
        Connection conn = DriverManager.getConnection("jdbc:db2:*local");

        Statement stmt = conn.createStatement();

        // Clean up previous runs
        try {
            stmt.execute("drop table cujosql.test_table");
        }

        catch (SQLException ex) {
            System.out.println("Caught drop table: " + ex.getMessage());
        }

        // Create test table
        stmt.execute("Create table cujosql.test_table (col1 smallint)");
        System.out.println("Table created.");

        // Insert some test rows
        for (int i = 0; i < 10; i++) {
            stmt.execute("insert into cujosql.test_table values (" + i + ")");
        }
        System.out.println("Rows inserted");

        ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
        System.out.println("Query executed");

        // Create a new rowset and populate it...
        DB2CachedRowSet crs = new DB2CachedRowSet();
        crs.populate(rs);
        System.out.println("RowSet populated.");

        conn.close();
        System.out.println("RowSet is detached...");

        System.out.println("Delete the first three rows");
        crs.next();
        crs.deleteRow();
        crs.next();
        crs.deleteRow();
        crs.next();
        crs.deleteRow();

        crs.beforeFirst();
        System.out.println("Insert the value -10 into the RowSet");
        crs.moveToInsertRow();
        crs.updateShort(1, (short)-10);
        crs.insertRow();
        crs.moveToCurrentRow();

        System.out.println("Update the rows to be the negative of what they now are");
        crs.beforeFirst();
        while (crs.next())
            short value = crs.getShort(1);
    }
}

```

```

        value = (short)-value;
        crs.updateShort(1, value);
        crs.updateRow();
    }

    crs.setShowDeleted(true);

    System.out.println("RowSet is now (value - inserted - updated - deleted)");
    crs.beforeFirst();
    while (crs.next()) {
        System.out.println("value is " + crs.getShort(1) + " " +
            crs.rowInserted() + " " +
            crs.rowUpdated() + " " +
            crs.rowDeleted());
    }

    System.out.println("getShowDeleted is " + crs.getShowDeleted());

    System.out.println("Now undo the inserts and deletes");
    crs.beforeFirst();
    crs.next();
    crs.cancelRowDelete();
    crs.next();
    crs.cancelRowDelete();
    crs.next();
    crs.cancelRowDelete();
    while (!crs.isLast()) {
        crs.next();
    }

    crs.cancelRowInsert();

    crs.setShowDeleted(false);

    System.out.println("RowSet is now (value - inserted - updated - deleted)");
    crs.beforeFirst();
    while (crs.next()) {
        System.out.println("value is " + crs.getShort(1) + " " +
            crs.rowInserted() + " " +
            crs.rowUpdated() + " " +
            crs.rowDeleted());
    }

    System.out.println("finally show that calling cancelRowUpdates works");
    crs.first();
    crs.updateShort(1, (short) 1000);
    crs.cancelRowUpdates();
    crs.updateRow();
    System.out.println("value of row is " + crs.getShort(1));
    System.out.println("getShowDeleted is " + crs.getShowDeleted());

    crs.close();
}

catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
}
}
}

```

## DB2CachedRowSet に加えられた変更を、元のデータベースに反映する

DB2CachedRowSet への変更が加えられると、その変更は RowSet オブジェクトが存在している間のみ存在します。つまり、切断された RowSet に加えられた変更は、データベースには影響を与えません。RowSet に加えられた変更を元のデータベースに反映するには、acceptChanges メソッドを使用します。このメソッド

ドは、切断された RowSet がデータベースへの接続を再確立し、RowSet に加えられた変更を元のデータベースに戻すよう試行します。 RowSet が作成された後にデータベースに加えられた他の変更との競合により、データベースに安全に変更が加えられない場合は、例外がスローされ、トランザクションがロールバックします。

**例:** DB2CachedRowSet に加えられた変更を、元のデータベースに反映する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;

public class RowSetSample3
{
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                ex.getMessage());
            // No need to go any further.
            System.exit(1);
        }

        try {
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");

            Statement stmt = conn.createStatement();

            // Clean up previous runs
            try {
                stmt.execute("drop table cujosql.test_table");
            }
            catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create test table
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Table created.");

            // Insert some test rows
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ")");
            }
            System.out.println("Rows inserted");

            ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
            System.out.println("Query executed");

            // Create a new rowset and populate it...
            DB2CachedRowSet crs = new DB2CachedRowSet();
            crs.populate(rs);
            System.out.println("RowSet populated.");

            conn.close();
            System.out.println("RowSet is detached...");

            System.out.println("Delete the first three rows");
            crs.next();
            crs.deleteRow();
        }
    }
}
```

```

crs.next();
crs.deleteRow();
crs.next();
crs.deleteRow();

crs.beforeFirst();
System.out.println("Insert the value -10 into the RowSet");
crs.moveToInsertRow();
crs.updateShort(1, (short)-10);
crs.insertRow();
crs.moveToCurrentRow();

System.out.println("Update the rows to be the negative of what they now are");
crs.beforeFirst();
while (crs.next()) {
    short value = crs.getShort(1);
    value = (short)-value;
    crs.updateShort(1, value);
    crs.updateRow();
}

System.out.println("Now accept the changes to the database");

crs.setUrl("jdbc:db2:*local");
crs.setTableName("cujosql.test_table");

crs.acceptChanges();
crs.close();

System.out.println("And the database table looks like this:");
conn = DriverManager.getConnection("jdbc:db2:localhost");
stmt = conn.createStatement();
rs = stmt.executeQuery("select col1 from cujosql.test_table");
while (rs.next()) {
    System.out.println("Value from table is " + rs.getShort(1));
}

conn.close();
}
catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
}
}
}

```

### その他の *DB2CachedRowSet* の機能:

いくつかの例で示したように、*ResultSet* のような動作に加え、*DB2CachedRowSet* クラスには、さらに柔軟に使用できるいくつかの追加機能があります。これらのメソッドは、完全な Java Database Connectivity (JDBC) *RowSet*、またはその一部を Java コレクションに変換します。さらに、それらが切断状態にあるため、*DB2CachedRowSet* は *ResultSet* と絶対的な 1 対 1 の関係を持っていません。

いくつかの例で示したように、*ResultSet* のような動作に加え、*DB2CachedRowSet* クラスには、さらに柔軟に使用できるいくつかの追加機能があります。これらのメソッドは、完全な Java Database Connectivity (JDBC) *RowSet*、またはその一部を Java コレクションに変換します。さらに、それらが切断状態にあるため、*DB2CachedRowSet* は *ResultSet* と絶対的な 1 対 1 の関係を持っていません。

*DB2CachedRowSet* で提供されているメソッドを使って、次のようなタスクを実行できます。

## DB2CachedRowSets からコレクションを取得する

DB2CachedRowset オブジェクトからいくつかの形式のコレクションを戻すには、3 つのメソッドがあります。以下のメソッドです。

- **toCollection** は、ベクトル (1 項目が 1 列) の ArrayList (1 項目が 1 行) で戻します。
- **toCollection(int columnIndex)** は、各行に指定された列の値を格納したベクトルを戻します。
- **getColumn(int columnIndex)** は、各列に指定された列の値を格納した配列を戻します。

toCollection(int columnIndex) と getColumn(int columnIndex) の大きな違いは、getColumn メソッドはプリミティブ・タイプの配列を戻すことができる点です。したがって、columnIndex で整数データを持つ列を指定した場合、整数の配列が戻され、java.lang.Integer オブジェクトの配列が戻されるわけではありません。

以下に、これらのメソッドを使い方を示します。

**例:** DB2CachedRowSets からコレクションを取得する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2CachedRowSet;
import java.util.*;

public class RowSetSample4
{
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                ex.getMessage());
            // No need to go any further.
            System.exit(1);
        }

        try {
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");
            Statement stmt = conn.createStatement();

            // Clean up previous runs
            try {
                stmt.execute("drop table cujosql.test_table");
            }
            catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create test table
            stmt.execute("Create table cujosql.test_table (col1 smallint, col2 smallint)");
            System.out.println("Table created.");

            // Insert some test rows
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ", " + (i + 100) + ")");
            }
            System.out.println("Rows inserted");

            ResultSet rs = stmt.executeQuery("select * from cujosql.test_table");
            System.out.println("Query executed");
        }
    }
}
```

```

// Create a new rowset and populate it...
DB2CachedRowSet crs = new DB2CachedRowSet();
crs.populate(rs);
System.out.println("RowSet populated.");

conn.close();
System.out.println("RowSet is detached...");

System.out.println("Test the toCollection() method");
Collection collection = crs.toCollection();
ArrayList map = (ArrayList) collection;

System.out.println("size is " + map.size());
Iterator iter = map.iterator();
int row = 1;
while (iter.hasNext()) {
    System.out.print("row [" + (row++) + "]: ¥t");

    Vector vector = (Vector)iter.next();
    Iterator innerIter = vector.iterator();
    int i = 1;
    while (innerIter.hasNext()) {
        System.out.print(" [" + (i++) + "]= " + innerIter.next() + "; ¥t");
    }
    System.out.println();
}
System.out.println("Test the toCollection(int) method");
collection = crs.toCollection(2);
Vector vector = (Vector) collection;

iter = vector.iterator();

while (iter.hasNext()) {
    System.out.println("Iter: Value is " + iter.next());
}

System.out.println("Test the getColumn(int) method");
Object values = crs.getColumn(2);
short[] shorts = (short [])values;

for (int i =0; i < shorts.length; i++) {
    System.out.println("Array: Value is " + shorts[i]);
}
}
catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
}
}
}

```

## RowSet のコピーを作成する

createCopy メソッドは、DB2CachedRowSet のコピーを作成します。 RowSet に関連したすべてのデータが、すべての制御構造、プロパティ、および状況フラグと共に複製されます。

以下に、このメソッドを使い方を示します。

**例:** RowSet のコピーを作成する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;

```

```

import java.io.*;

public class RowSetSample5
{
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                ex.getMessage());
            // No need to go any further.
            System.exit(1);
        }

        try {
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");

            Statement stmt = conn.createStatement();

            // Clean up previous runs
            try {
                stmt.execute("drop table cujosql.test_table");
            }
            catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create test table
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Table created.");

            // Insert some test rows
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ")");
            }
            System.out.println("Rows inserted");

            ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
            System.out.println("Query executed");

            // Create a new rowset and populate it...
            DB2CachedRowSet crs = new DB2CachedRowSet();
            crs.populate(rs);
            System.out.println("RowSet populated.");

            conn.close();
            System.out.println("RowSet is detached...");

            System.out.println("Now some new RowSets from one.");
            DB2CachedRowSet crs2 = crs.createCopy();
            DB2CachedRowSet crs3 = crs.createCopy();

            System.out.println("Change the second one to be negated values");
            crs2.beforeFirst();
            while (crs2.next()) {
                short value = crs2.getShort(1);
                value = (short)-value;
                crs2.updateShort(1, value);
                crs2.updateRow();
            }

            crs.beforeFirst();
            crs2.beforeFirst();
            crs3.beforeFirst();
        }
    }
}

```



```

System.out.println("Now look at all three of them again");

while (crs.next()) {
    crs2.next();
    crs3.next();
    System.out.println("Values: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
        ", crs3: " + crs3.getShort(1));
}
}
catch (Exception ex) {
    System.out.println("SQLException: " + ex.getMessage());
    ex.printStackTrace();
}
}
}
}

```

## RowSet の共用を作成する

createShared メソッドは、高レベルの状況情報付きの新しい RowSet オブジェクトを作成し、2つの RowSet オブジェクトが同一の基礎となる物理データを共用できるようにします。

以下に、このメソッドを使い方を示します。

**例:** RowSet の共用を作成する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.*;
import java.io.*;

public class RowSetSample5
{
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        }
        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                ex.getMessage());
            // No need to go any further.
            System.exit(1);
        }

        try {
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");

            Statement stmt = conn.createStatement();

            // Clean up previous runs
            try {
                stmt.execute("drop table cujosql.test_table");
            }
            catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create test table
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Table created.");
        }
    }
}

```

```

// Insert some test rows
for (int i = 0; i < 10; i++) {
    stmt.execute("insert into cujosql.test_table values (" + i + ")");
}
System.out.println("Rows inserted");

ResultSet rs = stmt.executeQuery("select col1 from cujosql.test_table");
System.out.println("Query executed");

// Create a new rowset and populate it...
DB2CachedRowSet crs = new DB2CachedRowSet();
crs.populate(rs);
System.out.println("RowSet populated.");

conn.close();
System.out.println("RowSet is detached...");

System.out.println("Test the createShared functionality (create 2 shares)");
DB2CachedRowSet crs2 = crs.createShared();
DB2CachedRowSet crs3 = crs.createShared();

System.out.println("Use the original to update value 5 of the table");
crs.absolute(5);
crs.updateShort(1, (short)-5);
crs.updateRow();

crs.beforeFirst();
crs2.afterLast();

System.out.println("Now move the cursors in opposite directions of the same data.");

while (crs.next()) {
    crs2.previous();
    crs3.next();
    System.out.println("Values: crs: " + crs.getShort(1) + ", crs2: " + crs2.getShort(1) +
        ", crs3: " + crs3.getShort(1));
}
crs.close();
crs2.close();
crs3.close();
}
catch (Exception ex) {
    System.out.println("SQLException: " + ex.getMessage());
    ex.printStackTrace();
}
}
}

```

### DB2JdbcRowSet:

DB2JdbcRowSet は接続された RowSet で、基盤となっている Connection オブジェクト、PreparedStatement オブジェクト、または ResultSet オブジェクトのサポートでのみ使用できます。そのインプリメンテーションは、JdbcRowSet の記述に厳密に従っています。

### DB2JdbcRowSet の使用

DB2JdbcRowSet オブジェクトは Java Database Connectivity (JDBC) 3.0 仕様で記述されているすべての RowSet のイベントをサポートしているため、ローカル・データベースと、データベースのデータの変更が通知される必要のある他のオブジェクトとの中間オブジェクトとして動作することができます。

たとえば、メイン・データベースと、それに接続するために無線プロトコルを使用するいくつかの PDA という環境で作業することを想定します。 DB2JdbcRowSet オブジェクトは、サーバー上で動作するマスタ

ー・アプリケーションを使用した、行への移動とその更新に使用することができます。行を更新すると、RowSet コンポーネントによってイベントが生成されます。もし PDA に対する更新の送信を担当するサービスが動作していれば、これを RowSet の「リスナー」として登録することができます。RowSet イベントを受信するたびに、無線デバイスに対して適切な更新および送信を生成することができます。

詳しくは、例: DB2JdbcRowSet イベントを参照してください。

## JDBCRowSets の作成

DB2JDBCRowSet オブジェクトを生成するために、いくつかのメソッドが提供されています。以下にそれぞれを概説します。

### DB2JdbcRowSet プロパティおよび DataSource を使用する

DB2JdbcRowSet には、SQL 照会および DataSource 名を受け取るプロパティがあります。その後、DB2JdbcRowSet は使用可能になります。以下に、この方法の例を示します。BaseDataSource という名前の DataSource への参照が、事前に有効な DataSource としてセットアップされていることを想定しています。

#### 例: DB2JDBCRowSet プロパティおよび DataSource を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Create a new DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Set the properties that are needed for
// the RowSet to be processed.
jrs.setDataSourceName("BaseDataSource");
jrs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This method causes
// the RowSet to use the DataSource and SQL query
// specified to prepare itself for data processing.
jrs.execute();

// Loop through the data in the RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventually, close the RowSet.
jrs.close();
```

### DB2JdbcRowSet プロパティおよび JDBC URL を使用する

DB2JdbcRowSet には、SQL 照会および JDBC URL を受け取るプロパティがあります。その後、DB2JdbcRowSet は使用可能になります。以下に、この方法の例を示します。

#### 例: DB2JdbcRowSet プロパティおよび JDBC URL を使用する

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Create a new DB2JdbcRowSet
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Set the properties that are needed for
// the RowSet to be processed.
jrs.setUrl("jdbc:db2:*local");
jrs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
```

```

// the RowSet to use the URL and SQL query specified
// previously to prepare itself for data processing.
jrs.execute();

// Loop through the data in the RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventually, close the RowSet.
jrs.close();

```

### **setConnection(Connection) メソッドを使って、既存のデータベース接続を使用する**

JDBC Connection オブジェクトの再利用を促進するため、DB2JdbcRowSet は確立された接続を DB2JdbcRowSet に渡すことができます。この接続は、execute メソッドが呼び出されると、その使用の準備のために DB2JdbcRowSet によって使用されます。

#### **例: setConnection メソッド**

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

// Establish a JDBC Connection to the database.
Connection conn = DriverManager.getConnection("jdbc:db2:*local");

// Create a new DB2JdbcRowSet.
DB2JdbcRowSet jrs = new DB2JdbcRowSet();

// Set the properties that are needed for
// the RowSet to use an established connection.
jrs.setConnection(conn);
jrs.setCommand("select col1 from cujosql.test_table");

// Call the RowSet execute method. This causes
// the RowSet to use the connection that it was provided
// previously to prepare itself for data processing.
jrs.execute();

// Loop through the data in the RowSet.
while (jrs.next()) {
    System.out.println("v1 is " + jrs.getString(1));
}

// Eventually, close the RowSet.
jrs.close();

```

### **データのアクセスおよびカーソル移動**

DB2JdbcRowSet を介したカーソル位置の操作、およびデータベース・データへのアクセスは、基盤になる ResultSet オブジェクトによって処理されます。ResultSet オブジェクトで完了できるタスクは、DB2JdbcRowSet オブジェクトにも適用されます。

### **データの変更、および元のデータベースへの変更の反映**

DB2JdbcRowSet を介したデータベースの更新のサポートは、基盤となる ResultSet オブジェクトによってすべて処理されます。ResultSet オブジェクトで完了できるタスクは、DB2JdbcRowSet オブジェクトにも適用されます。

#### **DB2JdbcRowSet イベント:**

すべての RowSet インプリメンテーションは、他のコンポーネントにとって興味ある状態を処理するイベントをサポートしています。このサポートにより、アプリケーション・コンポーネントでイベントが発生したとき、それらのコンポーネントがお互いに「話し合う」ことが可能です。たとえば、RowSet を介してデータベースの行が更新されたときに、グラフィカル・ユーザー・インターフェース (GUI) で更新された表を表示させることができます。

以下の例で、メインメソッドは RowSet の更新を行うコア・アプリケーションです。リスナーは、そのアプリケーション領域内で切断されたクライアントが使用する、無線サーバーの一部です。これにより、2 つの処理が混在するコードを使うことなく、ビジネスの 2 つの側面を互いに結び付けることができます。RowSet のこのイベント・サポートは、主にデータベース・データによって GUI を更新することを目的に設計されており、このタイプのアプリケーションの問題に対して完全に動作します。

#### 例: DB2JdbcRowSet イベント

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import java.sql.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.DB2JdbcRowSet;

public class RowSetEvents {
    public static void main(String args[])
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException: " +
                ex.getMessage());
            // No need to go any further.
            System.exit(1);
        }

        try {
            // Obtain the JDBC Connection and Statement needed to set
            // up this example.
            Connection conn = DriverManager.getConnection("jdbc:db2:*local");
            Statement stmt = conn.createStatement();

            // Clean up any previous runs.
            try {
                stmt.execute("drop table cujosql.test_table");
            } catch (SQLException ex) {
                System.out.println("Caught drop table: " + ex.getMessage());
            }

            // Create the test table
            stmt.execute("Create table cujosql.test_table (col1 smallint)");
            System.out.println("Table created.");

            // Populate the table with data.
            for (int i = 0; i < 10; i++) {
                stmt.execute("insert into cujosql.test_table values (" + i + ")");
            }
            System.out.println("Rows inserted");

            // Remove the setup objects.
            stmt.close();
            conn.close();

            // Create a new rowset and set the properties need to
            // process it.
```

```

DB2JdbcRowSet jrs = new DB2JdbcRowSet();
jrs.setUrl("jdbc:db2:*local");
jrs.setCommand("select col1 from cujosql.test_table");
jrs.setConcurrency(ResultSet.CONCUR_UPDATEABLE);

// Give the RowSet object a listener. This object handles
// special processing when certain actions are done on
// the RowSet.
jrs.addRowSetListener(new MyListener());

// Process the RowSet to provide access to the database data.
jrs.execute();

// Cause a few cursor change events. These events cause the cursorMoved
// method in the listener object to get control.
jrs.next();
jrs.next();
jrs.next();

// Cause a row change event to occur. This event causes the rowChanged method
// in the listener object to get control.
jrs.updateShort(1, (short)6);
jrs.updateRow();

// Finally, cause a RowSet change event to occur. This causes the
// rowSetChanged method in the listener object to get control.
jrs.execute();

// When completed, close the RowSet.
jrs.close();
} catch (SQLException ex) {
    ex.printStackTrace();
}
}

/**
 * This is an example of a listener. This example prints messages that show
 * how control flow moves through the application and offers some
 * suggestions about what might be done if the application were fully implemented.
 */
class MyListener
implements RowSetListener {
    public void cursorMoved(RowSetEvent rse) {
        System.out.println("Event to do: Cursor position changed.");
        System.out.println(" For the remote system, do nothing ");
        System.out.println(" when this event happened. The remote view of the data");
        System.out.println(" could be controlled separately from the local view.");
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("row is " + rs.getRow() + ". %n%n");
        } catch (SQLException e) {
            System.out.println("To do: Properly handle possible problems.");
        }
    }

    public void rowChanged(RowSetEvent rse) {
        System.out.println("Event to do: Row changed.");
        System.out.println(" Tell the remote system that a row has changed. Then,");
        System.out.println(" pass all the values only for that row to the ");
        System.out.println(" remote system.");
        try {
            DB2JdbcRowSet rs = (DB2JdbcRowSet) rse.getSource();
            System.out.println("new values are " + rs.getShort(1) + ". %n%n");
        } catch (SQLException e) {
            System.out.println("To do: Properly handle possible problems.");
        }
    }
}

```

```

    }
}

public void rowSetChanged(RowSetEvent rse) {
    System.out.println("Event to do: RowSet changed.");
    System.out.println(" If there is a remote RowSet already established, ");
    System.out.println(" tell the remote system that the values it ");
    System.out.println(" has should be thrown out. Then, pass all ");
    System.out.println(" the current values to it.¥n¥n");
}
}

```

## IBM Developer Kit for Java の JDBC ドライバーに関するパフォーマンス上のヒント

IBM Developer Kit for Java の JDBC ドライバーは、データベースを処理する高性能の Java インターフェースとして設計されています。ただし、最高のパフォーマンスを得るためには、JDBC ドライバーが提供する能力を利用するようにアプリケーションを構築する必要があります。以下に挙げるヒントを適用すれば、JDBC プログラミングを効果的に行えるでしょう。ほとんどは、ネイティブ JDBC ドライバーに特有の情報ではありません。ですから、ここに示される指針に従って作成されたアプリケーションは、ネイティブ JDBC ドライバー以外の JDBC ドライバーと共に使用される場合でも高いパフォーマンスを示します。

### SELECT \* SQL 照会を避ける

SELECT \* FROM... は、SQL の照会を記述する一般的な方法です。しかし、すべてのフィールドを照会する必要がない場合もよくあります。戻されるそれぞれの列ごとに、JDBC ドライバーは、行をバインドして戻す余分の作業をしなければなりません。アプリケーションが特定の列を使用しない場合でも、JDBC ドライバーはその列を認識しなければならず、それを使用するためのスペースを予約しなければなりません。使用されない列がテーブル内にほとんどない場合、このことは重大なオーバーヘッドにはなりません。しかし、使用されない列が多数ある場合、このオーバーヘッドは重大となる可能性があります。これを解決するための良い方法は、アプリケーションと関係のある列を以下のように個々にリストすることです。

```
SELECT COL1, COL2, COL3 FROM...
```

### getXXX(String) の代わりに getXXX(int) を使用する

ResultSet getXXX メソッドを使用する際に、列名をとるバージョンの代わりに数値をとるバージョンを使用します。数値定数の代わりに列名を使用できることは利点のように思えますが、データベース自体は列索引を処理する方法しか認識していません。したがって、列名をとる各 getXXX メソッドを呼び出す場合、それらメソッドのはデータベースに渡される前に JDBC ドライバーによって解決されなければなりません。getXXX メソッドは通常、何度も実行されるループ内部で呼び出されるので、この小さなオーバーヘッドは急激に蓄積します。

### Java プリミティブ・タイプの getObject 呼び出しを避ける

プリミティブ・タイプ (int, long, float など) 値をデータベースから取得するときは、プリミティブ・タイプ固有の get メソッド (getInt, getLong, getFloat) を使用する方が、getObject を使用するより早く取得できます。getObject 呼び出しは、プリミティブ・タイプに対して取得作業を行った後にオブジェクトを作成してユーザーに戻します。これは通常ループの中で行われますが、存続期間の短い無数のオブジェクトが作成される可能性があります。プリミティブ・コマンドの getObject を使用することには、ガーベッジ・コレクターが頻繁に活動化されてパフォーマンスが低下するという欠点があります。

### Statement よりも PreparedStatement を使用する

何回も使用される SQL ステートメントを作成する場合、Statement オブジェクトよりも PreparedStatement を使用する方がパフォーマンスが向上します。ステートメントを実行するたびに、2 つのステップのプロセ



ス、つまりステートメントが準備されてからステートメントが処理されます。 `PreparedStatement` を使用する際、そのステートメントの準備は、実行されるたびに行われるのではなく、構成されたときだけに行われます。 `PreparedStatement` の方が `Statement` よりも実行が速いことは知られていますが、プログラマーたちはこの利点をしばしば無視します。 `PreparedStatement` によるパフォーマンス向上を考えるなら、アプリケーションを設計する際、可能な場合にはいつも `PreparedStatement` を使用するのが賢明と言えます (以下の 190 ページの『`PreparedStatement` プーリングの使用を考慮する』を参照してください)。

## DatabaseMetaData 呼び出しを避ける

`DatabaseMetaData` 呼び出しの中には費用がかかるものがあることに注意してください。特に、`getBestRowIdentifier`、`getCrossReference`、`getExportedKeys`、および `getImportedKeys` メソッドは費用がかかります。一部の `DatabaseMetaData` 呼び出しには、システム・レベル・テーブルに対する複雑な結合条件が伴います。ただ便利だからという理由でそれらを使用せずに、その情報が必要な場合のみ使用してください。

## アプリケーションに対して適切なコミット・レベルを使用する

JDBC は複数のコミット・レベルを提供しており、システム内で複数のトランザクションが互いにどのように影響しあうかはこれによって決まります (詳しくは、トランザクションを参照してください)。デフォルトでは、最低のコミット・レベルが使用されます。つまり、トランザクションはコミット境界を介して互いの作業の一部を知ることができます。これは、ある種のデータベース異常を生じさせる可能性があります。そのため、一部のプログラマーはコミット・レベルを上げて、そのような異常の発生を心配しなくてもよいようにしています。コミット・レベルを上げると、より粗い細分度のロックでのデータベースのハングを引き起こすことになることに注意してください。これは、システムで可能な並行度を制限するので、いくつかのアプリケーションのパフォーマンスが極度に低下します。そもそもアプリケーションの設計によって、ロック上の異常な状況は起きない場合もよくあります。行おうとしていることを時間を取って理解し、トランザクションの分離レベルを、安全に使用できる最低レベルまでに制限してください。

## Unicode 形式でのデータの保管を考慮する

Java では、処理するすべての文字データ (String) が Unicode 形式でなければなりません。そのため、Unicode データを持たないテーブルはすべて、データベースにデータを挿入したりデータベースからデータを検索したりする際にデータを変換するため、JDBC ドライバーを必要とします。テーブルがすでに Unicode 形式の場合、JDBC ドライバーはデータを変換する必要はないので、データベースのデータをより早く取り出すことができます。Unicode 形式のデータは非 Java アプリケーションでは使用できないことがある、という点を理解しておく必要があります。なぜならそれらのアプリケーションは、Unicode を処理できないからです。また、文字データ以外の場合は、データの変換がないため、パフォーマンスが変わらないことも覚えておいてください。別の考慮事項として、Unicode 形式で保管されたデータは、単一バイトのデータと比べて 2 倍のスペースを使用します。ただし、何度も読み取られる文字列がたくさんある場合は、Unicode 形式でデータを保管することでパフォーマンスが大きく向上する場合があります。

## ストアード・プロシージャを使用する

Java ではストアード・プロシージャの使用がサポートされています。ストアード・プロシージャは、JDBC ドライバーが動的 SQL の代わりに静的 SQL を実行できるようにすることによって、パフォーマンスを向上させることができます。ストアード・プロシージャは、プログラムで実行する個々の SQL ステートメントごとには作成しないでください。ただし、可能な場合は、SQL ステートメントのグループを実行するストアード・プロシージャを作成してください。

## Numeric または Decimal の代わりに BigInt を使用する

スケールが 0 である Numeric フィールドまたは Decimal フィールドを使用する代わりに、BigInt データ・タイプを使用します。BigInt は Java プリミティブ・タイプの Long に直接変換されますが、Numeric または Decimal データ・タイプは、String オブジェクトまたは BigDecimal オブジェクトに変換されます。189 ページの『DatabaseMetaData 呼び出しを避ける』で述べたように、プリミティブ・データ・タイプの使用は、オブジェクトの作成が必要なタイプの使用より望ましいと言えます。

## 必要がなくなった JDBC リソースを明示的にクローズする

ResultSet、Statements、および Connections は、必要がなくなったときにアプリケーションによって明示的にクローズする必要があります。これにより、リソースは最も効率的にクリーンアップされ、パフォーマンスを向上させることができます。さらに、データベース・リソースが明示的にクローズされないと、リソース・リークが生じたり、データベース・ロックの時間が必要以上に長くなったりします。これは、アプリケーション障害の発生や、アプリケーションでの並行性の減少につながる可能性があります。

## 接続プーリングを使用する

Connection プーリングは、各ユーザー要求で独自の Connection オブジェクトを作成する代わりに、複数のユーザーで JDBC Connection オブジェクトを再利用する戦略です。Connection オブジェクトを作成するには費用がかかります。各ユーザーが新規のオブジェクトを作成する代わりに、パフォーマンスが重要なアプリケーションでオブジェクトのプールを共有する必要があります。多くの製品 (WebSphere など) は Connection プーリング・サポートを提供しており、これは、ユーザーの側の少し余分な努力で使用できます。Connection プーリング・サポートを持つ製品を使用しない場合や、プールの動作やパフォーマンスをよりよく制御するために独自のオブジェクトの作成を望む場合は、そのほうが合理的で容易でしょう。

## PreparedStatement プーリングの使用を考慮する

Statement プーリングの動作は、Connection プーリングの動作と類似しています。ただし、Connection を単にプールに入れる代わりに、Connection および PreparedStatement を含むオブジェクトをプールに入れます。その後、そのオブジェクトを検索し、使用する特定のステートメントにアクセスします。これによりパフォーマンスは劇的に向上します。

## 効率的な SQL を使用する

JDBC は SQL に基づいて作成されているので、SQL の効率を上げることは、JDBC の効率を上げることになります。したがって、照会の最適化、賢明な索引の選択、およびすぐれた SQL 設計は、JDBC にとって有益です。

## IBM Developer Kit for Java の DB2 SQLJ サポートを使用するデータベースへのアクセス

DB2 Structured Query Language for Java (SQLJ) サポートは、SQLJ ANSI 規格に基づいています。DB2 SQLJ サポートは、IBM Developer Kit for Java に含まれています。DB2 SQLJ サポートによって、Java アプリケーションの組み込み SQL を作成、構築、および実行することができます。

IBM Developer Kit for Java に備わっている SQLJ サポートには SQLJ ランタイム・クラスが含まれていて、それは /QIBM/ProdData/Java400/ext/runtime.zip から入手できます。

## SQLJ のセットアップ

サーバー上の Java アプリケーションで SQLJ を使用するためには、まずサーバーで SQLJ を使用するための準備を行う必要があります。詳しくは、204 ページの『SQLJ を使用するためのサーバーのセットアップ』を参照してください。

## SQLJ ツール

以下のツールも、IBM Developer Kit for Java に備わっている SQLJ サポートに含まれています。

- SQLJ 変換プログラムである `sqlj` は、組み込み SQL ステートメントを Java ソース・ステートメントに置き換えて、SQLJ プログラム内に存在する SQLJ 操作に関する情報を含む一連のプロファイルを生成します。
- DB2 SQLJ プロファイル・カスタマイザーである `db2profc` は、生成されたプロファイルに格納された SQL ステートメントをプリコンパイルして、DB2 データベース内にパッケージを生成します。
- DB2 SQLJ プロファイル・プリンターである `db2profp` は、DB2 のカスタマイズ済みプロファイルの内容を通常のテキスト形式で印刷します。
- SQLJ プロファイル監査プログラム・インストーラーである `profdb` は、デバッグ・クラス監査プログラムをバイナリー・プロファイルの既存のセット内にインストール、およびアンインストールします。
- SQLJ プロファイル変換ツールである `profconv` は、一連のプロファイル・インスタンスを Java クラス形式に変換します。

注: これらのツールは、Qshell インタープリターで実行しなければなりません。

## DB2 SQLJ の制約事項

SQLJ を使用して DB2 アプリケーションを作成する場合、以下の制約事項に注意してください。

- DB2 SQLJ サポートは、SQL ステートメントを発行するための標準 DB2 Universal Database™ 制約に従っています。
- DB2 SQLJ プロファイル・カスタマイザーを実行できるのは、ローカル・データベースへの接続に関連したプロファイル上だけです。
- SQLJ Reference Implementation では、JDK 1.1 以降が必要です。Java Development Kit の複数のバージョンを実行することに関する詳細は、複数の Java Development Kits (JDK) のサポートを参照してください。

SQL を Java アプリケーション内で使用することについての情報は、SQL ステートメントを Java アプリケーションに組み込むおよび SQLJ プログラムのコンパイルおよび実行を参照してください。

## Structured Query Language for Java のプロファイル

プロファイルは、SQLJ ソース・ファイルを変換するときに、SQLJ 変換プログラム `sqlj` によって生成されます。プロファイルは、一連のバイナリー・ファイルです。そのため、これらのファイルには `.ser` 拡張子があります。これらのファイルには、関連した SQLJ ソース・ファイルからの SQL ステートメントが含まれます。

SQLJ ソース・コードからプロファイルを生成するには、SQLJ 変換プログラム、`sqlj` を `.sqlj` ファイル上で実行します。

詳しくは、SQLJ プログラムのコンパイルおよび実行を参照してください。

## Structured Query Language for Java (SQLJ) 変換プログラム (sqlj)

SQLJ 変換プログラム、sqlj は、SQLJ プログラム内で見つかった SQL 操作に関する情報を含む一連のプロファイルを生成します。SQLJ 変換プログラムは、/QIBM/ProdData/Java400/ext/translator.zip ファイルを使用します。

プロファイルの詳細については、プロファイルを参照してください。

## DB2 SQLJ プロファイル・カスタマイザー、db2profc を使用するプロファイル内での SQL ステートメントのプリコンパイル

DB2 SQLJ プロファイル・カスタマイザー、db2profc を使用して、Java アプリケーションがデータベース内でより効率的に作動するようにすることができます。

DB2 SQLJ プロファイル・カスタマイザーは、以下の事柄を行います。

- プロファイル内に格納された SQL ステートメントをプリコンパイルして、DB2 データベース内にパッケージを生成する。
- 作成されたパッケージ内の関連したステートメントを参照する SQL ステートメントを置き換えることにより、SQLJ プロファイルのカスタマイズする。

プロファイル内の SQL ステートメントをプリコンパイルするためには、Qshell コマンド・プロンプトに以下を入力します。

```
db2profc MyClass_SJProfile0.ser
```

ここで、MyClass\_SJProfile0.ser はプリコンパイルしたいプロファイルの名前です。

## DB2 SQLJ プロファイル・カスタマイザーの使用法および構文

```
db2profc[options] <SQLJ_profile_name>
```

ここで、SQLJ\_profile\_name は印刷するプロファイル名、options は使用したいオプションのリストです。

db2profp で使用可能なオプションは、以下のとおりです。

- -URL=<JDBC\_URL>
- -user=<username>
- -password=<password>
- -package=<library\_name/package\_name>
- -commitctrl=<commitment\_control>
- -datefmt=<date\_format>
- -datesep=<date\_separator>
- -timefmt=<time\_format>
- -timesep=<time\_separator>
- -decimalpt=<decimal\_point>
- -stmtCCSID=<CCSID>
- -sorttbl=<library\_name/sort\_sequence\_table\_name>
- -langID=<language\_identifier>

以下は、これらのオプションに関する説明です。

**-URL=<JDBC\_URL>**

ここで、*JDBC\_URL* は JDBC 接続の URL です。URL の構文は、次のとおりです。

"jdbc:db2:systemName"

詳しくは、IBM Developer Kit for Java の JDBC ドライバーを使用して iSeries データベースにアクセスするを参照してください。

**-user=<username>**

ここで、*username* はユーザー名です。デフォルト値は、ローカル接続にサインオンした現行ユーザーのユーザー ID です。

**-password=<password>**

ここで、*password* はパスワードです。デフォルト値は、ローカル接続にサインオンした現行ユーザーのパスワードです。

**-package=<library name/package name>**

ここで、*library name* はパッケージを入れるライブラリー、*package name* は生成されるパッケージの名前です。デフォルトのライブラリー名は、**QUSRSYS** です。デフォルトのパッケージ名は、プロファイルの名前から生成されます。パッケージ名の最大長は、10 文字です。SQLJ プロファイル名は常に 10 文字よりも長いので、作成されるデフォルトのパッケージ名はプロファイル名とは異なるものとなります。デフォルトのパッケージ名は、プロファイル名の最初の数文字とプロファイル・キー番号とを連結して作成されます。プロファイル・キー番号が 10 文字を超える長さである場合、プロファイル・キー番号の最後の 10 文字がデフォルトのパッケージ名に使用されます。たとえば、以下の図表は一部のプロファイル名とデフォルトのパッケージ名とを示しています。

| プロファイル名                  | デフォルトのパッケージ名 |
|--------------------------|--------------|
| App_SJProfile0           | App_SJPro0   |
| App_SJProfile01234       | App_S01234   |
| App_SJProfile012345678   | A012345678   |
| App_SJProfile01234567891 | 1234567891   |

**-commitctl=<commitment\_control>**

ここで、*commitment\_control* は必要なコミットメント制御のレベルです。コミットメント制御は、以下の文字値の 1 つを持つことができます。

| 値 | 定義   |
|---|--|
| C | *CHG。ダーティ・リード、繰り返し不可の読み取り、およびファントム・リードが可能。             |
| S | *CS。ダーティ・リードは不可。繰り返し不可の読み取り、およびファントム・リードは可能。           |
| A | *ALL。ダーティ・リードおよび繰り返し不可の読み取りは不可。ファントム・リードは可能。           |
| N | *NONE。ダーティ・リード、繰り返し不可の読み取り、およびファントム・リードは不可。これはデフォルトです。 |

**-datefmt=<date\_format>**

ここで、*date\_format* は使用したい日付形式のタイプです。日付形式には、以下の値の 1 つを指定できます。



| 値   | 定義  |
|-----|---|
| USA | IBM USA 標準規格 (mm.dd.yyyy, hh:mm a.m., hh:mm p.m.) |
| ISO | 国際標準化機構 (yyyy-mm-dd, hh.mm.ss)。これがデフォルトです。        |
| EUR | IBM 欧州標準規格 (dd.mm.yyyy, hh.mm.ss)                 |
| JIS | 日本工業規格 (JIS) 西暦 (yyyy-mm-dd, hh:mm:ss)            |
| MDY | 月/日/年 (mm/d/yy)                                   |
| DMY | 日/月/年 (dd/mm/yy)                                  |
| YMD | 年/月/日 (yy/mm/dd)                                  |
| JUL | ユリウス暦 (yy/ddd)                                    |

日付形式は、日付の結果の欄にアクセスするときに使用されます。すべての出力日付フィールドは、指定した形式で戻されます。入力日付ストリングでは、日付が有効な形式で指定されたかどうかを判別するために指定値が使用されます。デフォルト値は ISO です。

**-datesep=<date\_separator>**

ここで、*date\_separator* は使用したい区切り記号のタイプです。日付区切り記号は、日付の結果の欄にアクセスするときに使用されます。区切り記号には、以下の値の 1 つを使用できます。

| 値    | 定義                     |
|------|------------------------|
| /    | スラッシュが使用される。           |
| .    | ピリオドが使用される。            |
| ,    | コンマが使用される。             |
| -    | ダッシュが使用される。これはデフォルトです。 |
| ブランク | スペースが使用される。            |

**-timefmt=<time\_format>**

ここで、*time\_format* は時刻フィールドの表示に使用したい形式です。時刻形式は、時刻の結果の欄にアクセスするときに使用されます。入力時刻ストリングでは、時刻が有効な形式で指定されたかどうかを判別するために指定値が使用されます。時刻形式には、以下の値の 1 つを指定できます。

| 値   | 定義  |
|-----|---|
| USA | IBM USA 標準規格 (mm.dd.yyyy, hh:mm a.m., hh:mm p.m.) |
| ISO | 国際標準化機構 (yyyy-mm-dd, hh.mm.ss)。これがデフォルトです。        |
| EUR | IBM 欧州標準規格 (dd.mm.yyyy, hh.mm.ss)                 |
| JIS | 日本工業規格 (JIS) 西暦 (yyyy-mm-dd, hh:mm:ss)            |
| HMS | 時/分/秒 (hh:mm:ss)                                  |

**-timesep=<time\_separator>**

ここで、*time\_separator* は時刻の結果の欄にアクセスするときに使用したい文字です。時刻区切り記号には、以下の値の 1 つを指定できます。

| 値    | 定義                     |
|------|------------------------|
| :    | コロンが使用される。             |
| .    | ピリオドが使用される。これはデフォルトです。 |
| ,    | コンマが使用される。             |
| ブランク | スペースが使用される。            |

**-decimalpt=<decimal\_point>**

ここで、*decimal\_point* は使用したい小数点です。小数点は、SQL ステートメント内で数値定数に使用されます。小数点には、以下の値の 1 つを指定できます。

| 値 | 定義                     |
|---|------------------------|
| . | ピリオドが使用される。これはデフォルトです。 |
| , | コンマが使用される。             |

**-stmtCCSID=<CCSID>**

ここで、*CCSID* はパッケージ内に備わっている SQL ステートメントのためのコード化文字セット ID です。カスタマイズ時間中のジョブの値がデフォルト値となります。

**-sorttbl=<library\_name/sort\_sequence\_table\_name>**

ここで、*library\_name/sort\_sequence\_table\_name* は使用したい分類順序テーブルのロケーションおよびテーブル名です。分類順序テーブルは、SQL ステートメント内でストリングを比較するために使用されます。ライブラリー名および分類順序テーブル名は、それぞれ 10 文字の制限があります。デフォルト値は、カスタマイズ時間中のジョブから取得されます。

**-langID=<language\_identifier>**

ここで、*language identifier* は使用したい言語 ID です。言語 ID のデフォルト値は、カスタマイズ時間中の現行ジョブから取得されます。言語 ID は、分類順序テーブルと一緒に使用します。

これらのフィールドのいずれかに関するより詳細な情報は、DB2 for iSeries SQL プログラミング

(SD88-5034)  を参照してください。

## DB2 SQLJ プロファイル (db2profp および profp) の内容の印刷

DB2 SQLJ プロファイル・プリンター である db2profp は、DB2 のカスタマイズ済みプロファイルの内容を通常のテキスト形式で印刷します。プロファイル・プリンター、profp は、SQLJ 変換プログラムによって生成されたプロファイルの内容を通常のテキスト形式で印刷します。

SQLJ 変換プログラムによって生成されたプロファイルの内容を通常のテキスト形式で印刷するには、以下のように profp ユーティリティを使用します。

```
profp MyClass_SJProfile0.ser
```

ここで、*MyClass\_SJProfile0.ser* は印刷したいプロファイルの名前です。

プロファイルの DB2 カスタマイズ済みバージョンの内容を通常のテキスト形式で印刷するには、以下のよう db2profp ユーティリティを使用します。

```
db2profp MyClass_SJProfile0.ser
```

ここで、*MyClass\_SJProfile0.ser* は印刷したいプロファイルの名前です。



注: db2profp をカスタマイズしていないプロファイル上で実行すると、プロファイルがカスタマイズされていないことが通知されます。 profp をカスタマイズ済みプロファイル上で実行すると、カスタマイズされていないプロファイルの内容が表示されます。

## DB2 SQLJ プロファイル・プリンターの使用法および構文

```
db2profp [options] <SQLJ_profile_name>
```

ここで、*SQLJ\_profile\_name* は印刷するプロファイル名、*options* は使用したいオプションのリストです。

db2profp で使用可能なオプションは、以下のとおりです。

### -URL=<JDBC\_URL>

ここで、*JDBC\_URL* は接続したい URL です。詳しくは、IBM Developer Kit for Java の JDBC ドライバーを使用して iSeries データベースにアクセスするを参照してください。

### -user=<username>

ここで、*username* はユーザー・プロファイル内のユーザー名です。

### -password=<password>

ここで、*password* はユーザー・プロファイルのパスワードです。

## SQLJ プロファイル監査プログラム・インストーラー (profdb)

SQLJ プロファイル監査プログラム・インストーラー (profdb) は、デバッグ・クラス監査プログラムをインストール、およびアンインストールします。デバッグ・クラス監査プログラムは、バイナリー・プロファイルの既存のセットにインストールされます。デバッグ・クラス監査プログラムがインストールされた後は、アプリケーションの実行時間中に呼び出されるすべての RTStatement および RTResultSet 呼び出しがログに記録されます。それらはファイルまたは標準出力にログ記録することができます。その後、ログを検査してアプリケーションの動作の検証、およびエラーのトレースを行うことができます。実行時の基礎となる RTStatement および RTResultSetcall インターフェースに対する呼び出しだけが監査されることに注意してください。

デバッグ・クラス監査プログラムをインストールするには、Qshell コマンド・プロンプトに以下を入力します。

```
profdb MyClass_SJProfile0.ser
```

ここで、*MyClass\_SJProfile0.ser* は SQLJ 変換プログラムによって生成されたプロファイルの名前です。

デバッグ・クラス監査プログラムをアンインストールするには、Qshell コマンド・プロンプトに以下を入力します。

```
profdb -Cuninstall MyClass_SJProfile0.ser
```

ここで、*MyClass\_SJProfile0.ser* は SQLJ 変換プログラムによって生成されたプロファイルの名前です。

## SQLJ プロファイル変換ツール (profconv) を使用して、一連のプロファイル・インスタンスを Java クラス形式に変換する

SQLJ プロファイル変換ツール (profconv) は、一連のプロファイル・インスタンスを Java クラス形式に変換します。一部のブラウザーはアプレットに関連したリソース・ファイルから一連のオブジェクトをロードすることをサポートしていないため、profconv ツールが必要になります。profconv ユーティリティーを実行して、変換を行います。

profconv ユーティリティーを実行するには、Qshell コマンド行に以下を入力します。

```
profconv MyApp_SJProfile0.ser
```

ここで、*MyApp\_SJProfile0.ser* は変換したいプロファイル・インスタンスの名前です。

profconv ツールは、`sqlj -ser2class` を起動します。コマンド行オプションについては、`sqlj`を参照してください。

## SQL ステートメントを Java アプリケーションに組み込む

SQLJ 内の静的 SQL ステートメントは、SQLJ 文節に含まれています。SQLJ 文節は、`#sql` で開始して、セミコロン (;) 文字で終了します。

Java アプリケーション内に SQLJ 文節を作成する前に、以下のパッケージをインポートしてください。

- `import java.sql.*;`
- `import sqlj.runtime.*;`
- `import sqlj.runtime.ref.*;`

最も簡単な SQLJ 文節は、処理できる文節であり、トークン `#sql` およびそれに続く中括弧で囲まれた SQL ステートメントで構成されます。たとえば、以下の SQLJ 文節は Java ステートメントが適正に存在できる位置であればどこにでも存在できます。

```
#sql { DELETE FROM TAB };
```

上記の例は、TAB という名前のすべての行を削除します。

**注:** SQLJ アプリケーションのコンパイルおよび実行に関して詳しくは、SQLJ プログラムのコンパイルおよび実行を参照してください。

SQLJ プロセス文節で中括弧の内側にあるトークンは、SQL トークンまたはホスト変数です。すべてのホスト変数は、コロン (:) 文字によって識別されます。SQL トークンが SQLJ プロセス文節の中括弧外側に存在することはありません。たとえば、以下の Java メソッドは引数を SQL テーブルに挿入します。

```
public void insertIntoTAB1 (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

メソッド本体は、ホスト変数 `x`、`y`、および `z` を含む SQLJ プロセス文節で構成されます。詳しくは、SQLJ 内のホスト変数を参照してください。

一般に、SQL トークンは大文字小文字を区別しない (二重引用符で囲まれた ID を除く) ので、大文字、小文字、またはそれらの混合による記述が可能です。しかし、Java トークンは大文字小文字を区別します。例の中で明白にするために、大文字小文字を区別しない SQL トークンは大文字で示し、Java トークンは小文字または大文字小文字混合で示します。このトピックを通して、小文字の `null` は Java "null" 値を表すために使用され、大文字の `NULL` は SQL "null" 値を表すために使用されます。

以下のタイプの SQL 構成が、SQLJ プログラム内に存在することがあります。

- 照会: `SELECT` ステートメントおよび式など。
- SQL データ変更ステートメント (DML): `INSERT`、`UPDATE`、`DELETE`、など。
- データ・ステートメント: `FETCH`、`SELECT..INTO`、など。
- トランザクション制御ステートメント: `COMMIT`、`ROLLBACK`、など。
- データ定義言語 (DDL、スキーマ操作言語とも呼ばれる) ステートメント: `CREATE`、`DROP`、`ALTER` など。
- ストアド・プロシージャへの呼び出し: `CALL MYPROC(:x, :y, :z)` など

- ストアド関数の呼び出し: VALUES( MYFUN(:x) ) など

### Structured Query Language for Java 内のホスト変数:

組み込み SQL ステートメントへの引数は、ホスト変数を介して渡されます。ホスト変数は、ホスト言語による変数であり、SQL ステートメントに含めることができます。

ホスト変数は、以下に示す 3 つまでの部分から構成されます。

- コロン (:) 接頭部。
- パラメーター、変数、またはフィールドの Java ID である、Java ホスト変数。
- 任意指定のパラメーター・モード ID。

このモード ID には、以下の 1 つを使用できます。

IN, OUT, または INOUT。

Java ID を評価しても Java プログラムには副次作用がありません。そのためそれは、SQLJ 文節を置き換えるために生成された Java コード内に何回も出現することがあります。

以下の QUERY にはホスト変数 :x が含まれています。このホスト変数は、照会を含むスコープ内で可視の Java 変数、フィールド、またはパラメーター :x です。

```
SELECT COL1, COL2 FROM TABLE1 WHERE :x > COL3
```

### 例: SQL ステートメントを Java アプリケーションに組み込む:

以下の SQLJ アプリケーション例、App.sqlj は、静的 SQL を使用して更新データを DB2 サンプル・データベースの EMPLOYEE テーブルから検索します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // 1
#sql iterator App_Cursor2 (String) ;

class App
{
    /*****
    ** Register Driver **
    *****/

    static
    {
        try
        {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    /*****
    ** Main **
    *****/
}
```

```

public static void main(String argv[])
{
    try
    {
        App_Cursor1 cursor1;
        App_Cursor2 cursor2;

        String str1 = null;
        String str2 = null;
        long count1;

        // URL is jdbc:db2:dbname
        String url = "jdbc:db2:sample";

        DefaultContext ctx = DefaultContext.getDefaultContext();
        if (ctx == null)
        {
            try
            {
                // connect with default id/password
                Connection con = DriverManager.getConnection(url);
                con.setAutoCommit(false);
                ctx = new DefaultContext(con);
            }
            catch (SQLException e)
            {
                System.out.println("Error: could not get a default context");
                System.err.println(e);
                System.exit(1);
            }
            DefaultContext.setDefaultContext(ctx);
        }

        // retrieve data from the database
        System.out.println("Retrieve some data from the database.");
        #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

        // display the result set
        // cursor1.next() returns false when there are no more rows
        System.out.println("Received results:");
        while (cursor1.next()) // 3
        {
            str1 = cursor1.empno(); // 4
            str2 = cursor1.firstnme();

            System.out.print (" empno= " + str1);
            System.out.print (" firstnme= " + str2);
            System.out.println("");
        }
        cursor1.close(); // 9

        // retrieve number of employee from the database
        #sql { SELECT count(*) into :count1 FROM employee }; // 5
        if (1 == count1)
            System.out.println ("There is 1 row in employee table");
        else
            System.out.println ("There are " + count1
                + " rows in employee table");

        // update the database
        System.out.println("Update the database.");
        #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

        // retrieve the updated data from the database
        System.out.println("Retrieve the updated data from the database.");
        str1 = "000010";
    }
}

```

```

#sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

// display the result set
// cursor2.next() returns false when there are no more rows
System.out.println("Received results:");
while (true)
{
    #sql { FETCH :cursor2 INTO :str2 }; // 7
    if (cursor2.endFetch()) break; // 8

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
}
cursor2.close(); // 9

// rollback the update
System.out.println("Rollback the update.");
#sql { ROLLBACK work };
System.out.println("Rollback done.");
}
catch( Exception e )
{
    e.printStackTrace();
}
}
}

```

<sup>1</sup> 反復子を宣言する。このセクションでは、次の 2 種類の反復子を宣言します。

- App\_Cursor1: 列データのタイプおよび名前を宣言して、列名 (列に結び付けられた名前) に応じた列の値を戻します。
- App\_Cursor2: 列データのタイプを宣言して、列位置 (列に結び付けられた定位置) に応じた列の値を戻します。

<sup>2</sup> 反復子を初期設定する。反復子オブジェクト cursor1 が照会の結果を使用して初期設定されます。照会は結果を cursor1 に格納します。

<sup>3</sup> 反復子を次の行に進める。 cursor1.next() メソッドは、検索する行がなくなった場合にブール値の偽を戻します。

<sup>4</sup> データを移動する。名前付きアクセス機構メソッド empno() は、現在の行にある empno という名前の列の値を戻します。名前付きアクセス機構メソッド firstnme() は、現在の行にある firstnme() という名前の列の値を戻します。

<sup>5</sup> データをホスト変数に SELECT する。 SELECT ステートメントは、テーブル内の行数をホスト変数 count1 に渡します。

<sup>6</sup> 反復子を初期設定する。反復子オブジェクト cursor2 が照会の結果を使用して初期設定されます。照会は結果を cursor2 に格納します。

<sup>7</sup> データを検索する。 FETCH ステートメントは、結果テーブルから ByPos カーソル内で宣言された最初の列の現行値を、ホスト変数 str2 に戻します。

<sup>8</sup> FETCH.INTO ステートメントが成功したかを検査する。 endFetch() メソッドは、反復子が行に位置していない場合、つまり行を取り出す前回の試行が失敗した場合に、ブール値の真を戻します。 endFetch() メ

ソッドは、行を取り出す前回の試行が成功した場合に、偽を戻します。DB2 は next() メソッドが呼び出されたときに行の取り出しを試行します。FETCH...INTO ステートメントは、暗黙的に next() メソッドを呼び出します。

<sup>9</sup> 反復子をクローズする。close() メソッドは、反復子が保持しているリソースを解放します。反復子を明示的にクローズして、システム・リソースが適時に解放されるようにしてください。

この例に関する背景情報は、SQL ステートメントを Java アプリケーションに組み込むを参照してください。

## SQLJ プログラムのコンパイルおよび実行

Java プログラムに組み込み SQLJ ステートメントがある場合、それをコンパイルおよび実行するためには特別の手順に従う必要があります。

Java プログラムに組み込み SQLJ ステートメントがある場合、それをコンパイルおよび実行するためには特別の手順に従う必要があります。

1. SQLJ を使用するためのサーバーのセットアップを行います。
2. SQLJ 変換プログラム、sqlj を Java ソース・コード上で組み込み SQL と共に使用して、Java ソース・コードおよび関連したプロファイルを生成します。接続ごとに、1 つのプロファイルが生成されます。

たとえば、以下のコマンドを入力します。

```
sqlj MyClass.sqlj
```

ここで、MyClass.sqlj は SQLJ ファイルの名前です。

この例では、SQLJ 変換プログラムは MyClass.java ソース・コード・ファイル、および関連したプロファイルを生成します。関連したプロファイルの名前は、

MyClass\_SJProfile0.ser、MyClass\_SJProfile1.ser、MyClass\_SJProfile2.ser、以下同様となります。

**注:** SQLJ 変換プログラムは、-compile=false 文節によってコンパイル・オプションを明示的にオフにしなければ、変換済み Java ソース・コードを自動的にコンパイルしてクラス・ファイルを生成します。

3. SQLJ プロファイル・カスタマイザー・ツール、db2profrc を使用して、DB2 SQLJ Customizers を生成されたプロファイルにインストールして、DB2 パッケージをローカル・システム上に作成します。

たとえば、以下のコマンドを入力します。

```
db2profrc MyClass_SJProfile0.ser
```

ここで MyClass\_SJProfile0.ser は DB2 SQLJ Customizer が実行されるプロファイルの名前です。

**注:** この手順はオプションですが、実行時パフォーマンスを向上させるために勧められています。

4. Java クラス・ファイルを他の Java クラス・ファイルと同じ方法で実行します。

たとえば、以下のコマンドを入力します。

```
java MyClass
```

ここで、MyClass は Java クラス・ファイルの名前です。

**関連概念**



197 ページの『SQL ステートメントを Java アプリケーションに組み込む』

SQLJ 内の静的 SQL ステートメントは、SQLJ 文節に含まれています。SQLJ 文節は、#sql で開始して、セミコロン (;) 文字で終了します。

## Java SQL ルーチン

iSeries サーバーには、SQL ステートメントおよびプログラムから Java プログラムにアクセスできる機能が備わっています。これは、Java ストアド・プロシージャおよび Java ユーザー定義関数 (UDF) を使用して行います。iSeries サーバーは、Java ストアド・プロシージャおよび Java UDF を呼び出すための、DB2 と SQLJ の両方の規則をサポートしています。Java ストアド・プロシージャと Java UDF は両方とも、JAR ファイルに保管されている Java クラスを使用できます。iSeries サーバーは、JAR ファイルをデータベースに登録するために、SQLJ Part 1 規格で定義されたストアド・プロシージャを使用します。

### Java SQL ルーチンの使用

Java プログラムには、SQL ステートメントおよびプログラムからアクセスできます。これは、Java ストアド・プロシージャおよび Java ユーザー定義関数 (UDF) を使用して行います。

Java SQL ルーチンを使用するには、以下の操作を行ってください。

1. SQLJ を使用可能にする。

Java SQL ルーチンはどれも SQLJ を使用する可能性があるため、Java 2 Software Development Kit (J2SDK) の実行時には、SQLJ ランタイム・サポートを常に使用可能にしておいてください。J2SDK で SQLJ のランタイム・サポートを使用可能にするには、拡張ディレクトリーから SQLJ runtime.zip ファイルへのリンクを追加します。詳しくは、204 ページの『SQLJ を使用するためのサーバーのセットアップ』を参照してください。

2. ルーチン用の Java メソッドを作成する。

Java SQL ルーチンは、Java メソッドを SQL から処理します。このメソッドは、DB2 パラメーター引き渡し規則または SQLJ パラメーター引き渡し規則を使用して作成する必要があります。Java SQL ルーチンで使用されるメソッドのコーディング方法の詳細については、Java ストアド・プロシージャ、Java ユーザー定義関数、および Java ユーザー定義テーブル関数を参照してください。

3. Java クラスをコンパイルする。

Java パラメーター・スタイルを使用して作成された Java SQL ルーチンは、追加のセットアップなしでコンパイルできます。ただし、DB2GENERAL パラメーター・スタイルを使用した Java SQL ルーチンの場合は、com.ibm.db2.app.UDF クラスまたは com.ibm.db2.app.StoredProc クラスを拡張する必要があります。これらのクラスは、/QIBM/ProdData/Java400/ext/db2routines\_classes.jar という JAR ファイルに入っています。これらのルーチンをコンパイルするために javac を使用する場合は、この JAR ファイルが CLASSPATH の中になければなりません。たとえば、次のコマンドは、DB2GENERAL パラメーター・スタイルを使用するルーチンが入った Java ソース・ファイルをコンパイルします。

```
javac -DCLASSPATH=/QIBM/ProdData/Java400/ext/db2routines_classes.jar
source.java
```

4. データベースによって使用される Java 仮想マシン (JVM) が、コンパイル済みクラスを使用できるようにする。

データベース JVM によって使用されるユーザー定義クラスは、/QIBM/UserData/OS400/SQLLib/Function ディレクトリーか、データベースに登録されている JAR ファイルに置くことができます。



/QIBM/UserData/OS400/SQLLib/Function は、/sqlib/function の iSeries に相当します。このディレクトリーは、DB2 UDB が、他のプラットフォーム上で Java ストアド・プロシージャと Java UDF を保管する場所です。クラスが Java パッケージの一部である場合は、クラスは適切なサブディレクトリーになければなりません。たとえば、runit クラスが foo.bar パッケージの一部として作成される場合、ファイル runnit.class は統合ファイル・システムの /QIBM/ProdData/OS400/SQLLib/Function/foo/bar というディレクトリーになければなりません。

クラス・ファイルは、データベースに登録されている JAR ファイルに置くこともできます。JAR ファイルは、SQLJ.INSTALL\_JAR ストアド・プロシージャを使用して登録します。このストアド・プロシージャは、JAR ID を JAR ファイルに割り当てるために使用されます。この JAR ID は、クラス・ファイルが存在する JAR ファイルを識別するために使用されます。SQLJ.INSTALL\_JAR と、JAR ファイルを操作するための他のストアド・プロシージャの詳細については、JAR ファイルを操作する SQLJ プロシージャを参照してください。

#### 5. ルーチンをデータベースに登録する。

Java SQL ルーチンは、CREATE PROCEDURE および CREATE FUNCTION SQL ステートメントを使用してデータベースに登録します。これらのステートメントには、以下の要素が含まれます。

##### CREATE キーワード

Java SQL ルーチンを作成するための SQL ステートメントは、CREATE PROCEDURE または CREATE STATEMENT で始まります。

##### ルーチンの名前

次いで、SQL ステートメントは、データベースに認識されているルーチンの名前を識別します。これは、SQL から Java ルーチンにアクセスするために使用される名前です。

##### パラメーターおよび戻り値

次いで、SQL ステートメントは、Java ルーチン用のパラメーターおよび戻り値 (該当する場合) を識別します。

##### LANGUAGE JAVA

SQL ステートメントは、ルーチンが Java で作成されたことを示すために、キーワード LANGUAGE JAVA を使用します。

##### PARAMETER STYLE KEYWORDS

次いで、SQL ステートメントは、キーワード PARAMETER STYLE JAVA または PARAMETER STYLE DB2GENERAL を使用して、パラメーター・スタイルを識別します。

**外部名** 次いで、SQL ステートメントは、Java SQL ルーチンとして処理される Java メソッドを識別します。外部名の形式は、以下のどちらかになります。

- メソッドが /QIBM/UserData/OS400/SQLLib/Function ディレクトリーの下クラス・ファイルに存在する場合、メソッドは *classname.methodname* という形式で識別されます。ここで、*classname* はクラスの完全修飾名で、*methodname* はメソッドの名前です。
- メソッドがデータベースに登録されている JAR ファイルに存在する場合、メソッドは *jarid:classname.methodname* という形式で識別されます。ここで、*jarid* は登録されている JAR ファイルの JAR ID、*classname* はクラスの名前、*methodname* はメソッドの名前です。

iSeries ナビゲーターを使用して、Java パラメーター・スタイルを使用するストアド・プロシージャまたはユーザー定義関数を作成することができます。

#### 6. Java プロシージャを使用する。

Java ストアド・プロシージャは、SQL CALL ステートメントを使用して呼び出します。Java UDF は、別の SQL ステートメントの一部として呼び出される関数です。

### SQLJ を使用するためのサーバーのセットアップ:

組み込み SQLJ ステートメントを含む Java プログラムを実行する前に、SQLJ をサポートするようサーバーを設定してください。SQLJ サポートのためには、サーバーの CLASSPATH 環境変数を変更する必要があります。

Java クラスパスの処理について詳しくは、以下のページを参照してください。

Java クラスパス

### SQLJ と J2SDK の使用

サポートされている任意のバージョンの J2SDK を実行しているサーバーで SQLJ を使用するには、以下のステップを実行してください。

1. 次のファイルをサーバーの CLASSPATH 環境変数に追加します。

- /QIBM/ProdData/Os400/Java400/ext/sqlj\_classes.jar
- /QIBM/ProdData/Os400/Java400/ext/translator.zip

注: translator.zip は、SQLJ 変換プログラム (sqlj コマンド) を実行する場合にのみ、追加する必要があります。SQLJ を使用するコンパイル済みの Java プログラムを実行する場合は translator.zip を追加する必要はありません。詳しくは、以下のページを参照してください。

#### SQLJ 変換プログラム (sqlj)

2. iSeries コマンド・プロンプトで、以下のコマンドを使用して、拡張機能ディレクトリーから runtime.zip へのリンクを追加します。コマンドを 1 行で入力してから、**Enter** を押します。

```
ADDLNK OBJ('/QIBM/ProdData/Os400/Java400/ext/runtime.zip')
NEWLNK('/QIBM/UserData/Java400/ext/runtime.zip')
```

拡張機能のインストールについて詳しくは、以下のページを参照してください。

#### IBM Developer Kit for Java の拡張機能をインストールする

##### リンク集

Java クラスパス

Java<sup>TM</sup> 仮想マシンは、実行時にクラスを検索するために、Java クラスパスを使用します。また、Java のコマンドとツールも、クラスパスを使ってクラスの位置を判別します。デフォルトのシステム・クラスパス、CLASSPATH 環境変数、および classpath コマンド・パラメーターはすべて、クラスを探すときにどのディレクトリーを検索するかを指定するために使用されます。

#### SQLJ 変換プログラム (sqlj)

SQLJ 変換プログラム、sqlj は、SQLJ プログラム内で見つかった SQL 操作に関する情報を含む一連のプロファイルを生成します。SQLJ 変換プログラムは、/QIBM/ProdData/Java400/ext/translator.zip ファイルを使用します。

#### IBM Developer Kit for Java の拡張機能をインストールする

拡張機能は、コア・プラットフォームの機能を拡張するために使用できる Java クラスのパッケージです。拡張機能は、1 つまたは複数の ZIP ファイルまたは JAR ファイルのパッケージであり、拡張クラス・ローダーによって Java 仮想マシンにロードされます。

## Java ストアード・プロシージャ

Java を使用してストアード・プロシージャを作成する場合、パラメーターを渡すスタイルを 2 つ使用できます。

推奨されているスタイルは JAVA パラメーター・スタイル です。これは、SQLj (SQL ルーチンの標準) で指定されているパラメーター・スタイルに一致します。2 番目のスタイルは DB2GENERAL です。これは、DB2 UDB によって定義されているパラメーター・スタイルです。パラメーター・スタイルでは、Java ストアード・プロシージャのコーディング時に使用しなければならない規則についても決定します。

さらに、Java ストアード・プロシージャに関するいくつかの制限も意識していなければなりません。

**JAVA パラメーター・スタイル:** JAVA パラメーター・スタイルを使用する Java ストアード・プロシージャをコード化する場合、以下の規則に従わなければなりません。

- Java メソッドは public void static (インスタンスではない) メソッドである必要がある。
- Java メソッドのパラメーターは、SQL 互換タイプである必要がある。
- パラメーターがヌル互換タイプ (String など) の場合、Java メソッドは SQL NULL 値をテストできる。
- 出力パラメーターは、単一エレメント配列の使用により戻される。
- Java メソッドは getConnection メソッドを使って、現行データベースにアクセスできる。

JAVA パラメーター・スタイルを使用する Java ストアード・プロシージャは、public static メソッドです。このクラスの中では、ストアード・プロシージャはそのメソッド名とシングニチャーによって識別されます。ストアード・プロシージャを呼び出すときは、シングニチャーは CREATE PROCEDURE ステートメントによって定義された変数タイプに基づいて、動的に生成されます。

ヌル値を許可する Java タイプでパラメーターが渡される場合、Java メソッドは、そのパラメーターをヌルと比較して、入力パラメーターが SQL NULL かどうかを判別できます。

以下の Java タイプはヌル値をサポートしていません。

- short
- int
- long
- float
- double

ヌル値をサポートしていない Java タイプにヌル値が渡されると、エラー・コード -20205 で SQL 例外が戻されます。

出力パラメーターは、1 つのエレメントを含む配列として渡されます。Java ストアード・プロシージャは配列の最初のエレメントを設定して、出力パラメーターを設定できます。

組み込みアプリケーション・コンテキストへの接続には、以下の Java Database Connectivity (JDBC) 呼び出しを使用してアクセスされます。

```
connection=DriverManager.getConnection("jdbc:default:connection");
```

その後、この接続は JDBC API を使って SQL ステートメントを実行します。

以下に示すのは、1つの入力と2つの出力を持った小さなストアド・プロシージャです。これは与えられた SQL 照会を実行し、結果内の行数と SQLSTATE の両方を戻します。

**例:** 1つの入力と2つの出力を持つストアド・プロシージャ

**注:** サンプル・コードをご使用の場合は、560ページの『コードに関する特記事項』に同意していただいているものとします。

```
package mystuff;

import java.sql.*;
public class sample2 {
    public static void donut(String query, int[] rowCount,
        String[] sqlstate) throws Exception {
        try {
            Connection c=DriverManager.getConnection("jdbc:default:connection");
            Statement s=c.createStatement();
            ResultSet r=s.executeQuery(query);
            int counter=0;
            while(r.next()){
                counter++;
            }
            r.close(); s.close();
            rowCount[0] = counter;
        }catch(SQLException x){
            sqlstate[0]= x.getSQLState();
        }
    }
}
```

SQLj 標準では、JAVA パラメーター・スタイルを使用するルーチン内の `ResultSet` を戻すために、`ResultSet` を明示的に設定しなければなりません。 `ResultSet` を戻すプロシージャが作成される場合、`ResultSet` パラメーターがパラメーター・リストの最後に追加されます。たとえば、以下のステートメントの場合、

```
CREATE PROCEDURE RETURNTWO()
DYNAMIC RESULT SETS 2
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'javaClass!returnTwoResultSets'
```

シグニチャー `public static void returnTwoResultSets(ResultSet[] rs1, ResultSet[] rs2)` で Java メソッドを呼び出します。

`ResultSet` の出力パラメーターは、以下の例で示されているように、明示的に設定しなければなりません。DB2GENERAL スタイルのように、`ResultSet` および対応するステートメントをクローズしてはなりません。

**例:** 2つの `ResultSet` を戻すストアド・プロシージャ

**注:** サンプル・コードをご使用の場合は、560ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
public class javaClass {
    /**
     * Java stored procedure, with JAVA style parameters,
     * that processes two predefined sentences
     * and returns two result sets
     *
     * @param ResultSet[] rs1    first ResultSet
     * @param ResultSet[] rs2    second ResultSet
     */
}
```

```

    */
public static void returnTwoResultSets (ResultSet[] rs1, ResultSet[] rs2) throws Exception
{
    //get caller's connection to the database; inherited from StoredProc
    Connection con = DriverManager.getConnection("jdbc:default:connection");

    //define and process the first select statement
    Statement stmt1 = con.createStatement();
    String sql1 = "select value from table01 where index=1";
    rs1[0] = stmt1.executeQuery(sql1);

    //define and process the second select statement
    Statement stmt2 = con.createStatement();
    String sql2 = "select value from table01 where index=2";
    rs2[0] = stmt2.executeQuery(sql2);
}
}

```

サーバーで、ResultSet の順序付けを判別するために、追加の ResultSet パラメーターが調べられることはありません。サーバー上の ResultSet は、オープンされた順序で戻されます。SQLj 標準との互換性を確保するために、前述のように、オープンされる順序で結果が割り当てられなければなりません。

### DB2GENERAL パラメーター・スタイル:

DB2GENERAL パラメーター・スタイルを使った Java ストアド・プロシージャをコーディングするときは、これらの規則に従う必要があります。

- Java ストアド・プロシージャで定義されるクラスは、Java com.ibm.db2.app.StoredProc クラスの *extend*、またはサブクラスである必要がある。
- Java メソッドは、public void インスタンス・メソッドである必要がある。
- Java メソッドのパラメーターは、SQL 互換タイプである必要がある。
- Java メソッドは、isNull メソッドを使って SQL NULL 値をテストすることができる。
- Java メソッドは、set メソッドを使って、明示的にパラメーターを設定および戻す必要がある。
- Java メソッドは getConnection メソッドを使って、現行データベースにアクセスできる。

Java ストアド・プロシージャを含むクラスは、com.ibm.db2.app.StoredProc を拡張したクラスである必要があります。Java ストアド・プロシージャは public インスタンス・メソッドです。このクラスの中では、ストアド・プロシージャはそのメソッド名とシグニチャーによって識別されます。ストアド・プロシージャを呼び出すときは、シグニチャーは CREATE PROCEDURE ステートメントによって定義された変数タイプに基づいて、動的に生成されます。

com.ibm.db2.app.StoredProc クラスは、isNull メソッドを提供しており、Java メソッドは入力パラメーターが SQL NULL であった場合に判別できます。com.ibm.db2.app.StoredProc クラスには、出力パラメーターを設定するための set...() メソッドも提供されています。出力パラメーターを設定するには、これらのメソッドを使用しなければなりません。出力パラメーターを設定しない場合は、出力パラメーターは SQL NULL 値を戻します。

com.ibm.db2.app.StoredProc クラスは、組み込みアプリケーション・コンテキストへの JDBC 接続をフェッチするための以下のルーチンを提供しています。組み込みアプリケーション・コンテキストへの接続は、以下の JDBC 呼び出しを使ってアクセスされます。

```
public Java.sql.Connection getConnection( )
```

その後、この接続は JDBC API を使って SQL ステートメントを実行します。

以下に示すのは、1 つの入力と 2 つの出力を持った小さなストアード・プロシージャです。これは与えられた SQL 照会を処理し、結果内の行の数と SQLSTATE の両方を戻します。

**例:** 1 つの入力と 2 つの出力を持つストアード・プロシージャ

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
package mystuff;

import com.ibm.db2.app.*;
import java.sql.*;
public class sample2 extends StoredProc {
    public void donut(String query, int rowCount,
        String sqlstate) throws Exception {
        try {
            Statement s=getConnection().createStatement();
            ResultSet r=s.executeQuery(query);
            int counter=0;
            while(r.next()){
                counter++;
            }
            r.close(); s.close();
            set(2, counter);
        }catch(SQLException x){
            set(3, x.getSQLState());
        }
    }
}
```

DB2GENERAL パラメーター・スタイルを使ったプロシージャ内の `ResultSet` を戻すには、プロシージャの終了時に、`ResultSet` および応答するステートメントが開かれたままになっている必要があります。戻される `ResultSet` は、クライアント・アプリケーションによってクローズされる必要があります。複数の `ResultSet` が戻される場合は、`ResultSet` が開かれた順序で戻されます。たとえば、以下のストアード・プロシージャは 2 つの `ResultSet` を戻します。

**例:** 2 つの `ResultSet` を戻すストアード・プロシージャ

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
public void returnTwoResultSets() throws Exception
{
    // get caller's connection to the database; inherited from StoredProc
    Connection con = getConnection ();
    Statement stmt1 = con.createStatement ();
    String sql1 = "select value from table01 where index=1";
    ResultSet rs1 = stmt1.executeQuery(sql1);
    Statement stmt2 = con.createStatement();
    String sql2 = "select value from table01 where index=2";
    ResultSet rs2 = stmt2.executeQuery(sql2);
}
```

### Java ストアード・プロシージャの制限:

これらの制限は、Java ストアード・プロシージャに適用されます。

- Java ストアード・プロシージャは追加スレッドを作成できません。追加のスレッドを作成できるのは、ジョブがマルチスレッド可能な場合だけです。SQL ストアード・プロシージャを呼び出すジョブで、マルチスレッドが可能であるとは限らないので、Java ストアード・プロシージャは、追加スレッドを作成できません。



- Java クラス・ファイルにアクセスするために、借用権限を使用することはできません。
- Java ストアド・プロシージャは、システムにインストールされている最新バージョンの Java Development Kit を常に使用します。
- Blob クラスと Clob クラスは java.sql パッケージと com.ibm.db2.app パッケージの両方にあるので、同一のプログラム中でこれらの両方のクラスを使用する場合は、プログラマーはこれらのクラスの名前全体を使用しなければならない。com.ibm.db2.app 中の Blob クラスと Clob クラスが、ストアド・プロシージャに渡されるパラメーターとして使用されていることを、プログラムが確認しなければなりません。
- Java ストアド・プロシージャが作成される場合、システムはライブラリー内にサービス・プログラムを生成します。このプログラムは、プロシージャ定義を保管するのに使用されます。プログラムには、システムによって生成された名前が付けられています。この名前は、ストアド・プロシージャを作成したジョブのジョブ・ログを調べることにより、取得できます。プログラムを保管してから復元すると、プロシージャ定義が復元されます。Java ストアド・プロシージャをシステム間で移動する場合は、Java クラスが含まれる統合ファイル・システムに加えて、関数の定義が含まれるサービス・プログラムも移動する必要があります。
- Java ストアド・プロシージャは、データベースへの接続に使用される、JDBC 接続のプロパティー (たとえば、システムの命名など) を設定できません。事前取り出しが使用不可の場合を除き、デフォルトの JDBC 接続プロパティーが常に使用されます。

## Java ユーザー定義スカラー関数

Java スカラー関数は、Java プログラムから 1 つの値をデータベースに戻します。たとえば、2 つの数の合計を戻す、スカラー関数を作成できます。

Java ストアド・プロシージャのように、Java スカラー関数は、Java と DB2GENERAL という 2 つのパラメーター・スタイルのいずれかを使用します。Java ユーザー定義関数 (UDF) をコード化する場合、Java スカラー関数に関する制限に注意しなければなりません。

### パラメーター・スタイル Java

Java パラメーター・スタイルは、*SQLJ Part 1: SQL Routines* 標準で指定されているスタイルです。Java UDF をコード化する場合、以下の規則を使用してください。

- Java メソッドは public static メソッドである必要がある。
- Java メソッドは SQL 互換タイプを戻す必要がある。この戻り値がこのメソッドの結果です。
- Java メソッドのパラメーターは SQL 互換タイプである必要がある。
- nul値が許可されている Java タイプの場合、Java メソッドは SQL NULL 値をテストすることができる。

たとえば、INTEGER を戻し、タイプ CHAR(5)、BLOB(10K)、および DATE の引数を取る、sample!test3 という UDF の場合、DB2 では、UDF の Java インプリメンテーションで以下のシグニチャーが必要です。

```
import com.ibm.db2.app.*;
public class sample {
    public static int test3(String arg1, Blob arg2, Date arg3) { ... }
}
```

Java メソッドのパラメーターは SQL 互換タイプでなければなりません。たとえば、UDF が SQL タイプ t1、t2、および t3 の引数を取り、タイプ t4 を戻すように宣言されている場合、以下のように、必要な Java シグニチャーを持った Java メソッドとして呼び出されます。



```
public static T4 name (T1 a, T2 b, T3 c) { .....}
```

ここで、各パラメーターは次のように定義されます。

- *name* はメソッド名
- T1 から T4 は、SQL タイプ *t1* から *t4* に対応する Java タイプ。
- *a*、*b*、および *c* は入力引数用の任意の変数名。

SQL タイプと Java タイプの相互関連については、ストアド・プロシージャおよび UDF のパラメーター引き渡し規則を参照してください。

SQL NULL 値は、初期化されていない Java 変数として表現されます。これらの変数がオブジェクト・タイプの場合は、Java nul値を持ちます。SQL NULL が int などの Java スカラー・データ・タイプに渡されると、例外条件が発生します。

Java パラメーター・スタイルを使っているときに Java UDF から結果を戻すには、単純にメソッドから結果を戻します。

```
{ ....  
  return value;  
}
```

UDF およびストアド・プロシージャで使用される C モジュールと同様に、Java UDF では Java 標準入出力ストリーム (System.in、System.out、および System.err) は使用できません。

## パラメーター・スタイル DB2GENERAL

パラメーター・スタイル DB2GENERAL は Java UDF によって使用されます。このパラメーター・スタイルでは、戻り値は関数の最後のパラメーターとして渡され、com.ibm.db2.app.UDF クラスの *set* メソッドを使って設定される必要があります。

Java UDF をコーディングする際は、以下の規則に従う必要があります。

- Java UDF を含むクラスは、Java com.ibm.db2.app.UDF クラスの *extend*、またはサブクラスである必要がある。
- DB2GENERAL パラメーター・スタイルでは、Java メソッドは `public void` インスタンス・メソッドである必要がある。
- Java メソッドのパラメーターは、SQL 互換タイプである必要がある。
- Java メソッドは、`isNull` メソッドを使って SQL NULL 値をテストすることができる。
- DB2GENERAL パラメーター・スタイルでは、Java メソッドは `set()` メソッドを使って、明示的にパラメーターを設定、および戻す必要がある。

Java UDF を含むクラスは、Java クラス com.ibm.db2.app.UDF を拡張する必要があります。

DB2GENERAL パラメーター・スタイルを使った Java UDF は、Java クラスの `void` インスタンス・メソッドとなる必要があります。たとえば、INTEGER を戻し、タイプ CHAR(5)、BLOB(10K)、および DATE の引数を取る、`sample!test3` という UDF の場合、DB2 では、UDF の Java インプリメンテーションで以下のシグニチャーが必要です。

```
import com.ibm.db2.app.*;  
public class sample extends UDF {  
  public void test3(String arg1, Blob arg2, String arg3, int result) { ... }  
}
```

Java メソッドのパラメーターは、SQL タイプでなければなりません。たとえば、SQL タイプ t1、t2、および t3 の引数を取り、タイプ t4 を戻すものとして宣言された UDF は、次の Java シグニチャーを持った Java メソッドとして呼び出されます。

```
public void name (T1 a, T2 b, T3 c, T4 d) { .....}
```

ここで、各パラメーターは次のように定義されます。

- *name* はメソッド名
- T1 から T4 は、SQL タイプ t1 から t4 に対応する Java タイプ。
- *a*、*b*、および *c* は入力引数用の任意の変数名。
- *d* は、算出対象の UDF 結果を表す任意の変数名。

SQL タイプと Java タイプの相互関連については、ストアド・プロシージャおよび UDF のパラメーター引き渡し規則のセクションに記載されています。

SQL NULL 値は、初期化されていない Java 変数として表現されます。Java 規則に従って、これらの変数は、プリミティブ・タイプの場合にはゼロの値を持ち、オブジェクト・タイプの場合には Java NULL になります。SQL NULL を通常のゼロと区別するために、入力引数について isNull メソッドを呼び出すことができます。

```
{ ....  
  if (isNull(1)) { /* argument #1 was a SQL NULL */  
    else          { /* not NULL */  
  }
```

前述の例では、引数の番号は 1 から始まります。isNull() 関数は、それに続く他の関数と同様、com.ibm.db2.app.UDF クラスから継承します。DB2GENERAL パラメーター・スタイルを使っているときに Java UDF から結果を戻すには、次のように、set() メソッドを UDF 内で使用します。

```
{ ....  
  set(2, value);  
}
```

ここで 2 は出力引数の指標で、*value* は互換タイプのリテラルまたは変数です。引数番号は、選択された出力の引数リストの指標です。このセクションの最初の例では、int 結果変数に指標 4 が含まれています。UDF が戻る前に設定されない出力引数には NULL 値が入ります。

UDF およびストアド・プロシージャで使用される C モジュールと同様に、Java UDF では Java 標準入出力ストリーム (System.in、System.out、および System.err) は使用できません。

通常、DB2 は UDF を、照会の入力または ResultSet の各行に対して 1 回ずつ、何回も呼び出します。UDF の CREATE FUNCTION ステートメントで SCRATCHPAD が指定されている場合、DB2 は、連続する UDF の呼び出しの間にいくらかの「継続性」が必要であると認識します。したがって、DB2GENERAL パラメーター・スタイルの機能の場合、Java クラスのインプリメントのインスタンスは各呼び出しのたびに作成されず、一般にはステートメントごとの UDF 参照ごとに 1 回作成されます。しかし UDF に NO SCRATCHPAD が指定されている場合、クラス・コンストラクターの呼び出しによって、UDF の呼び出しのたびに白紙のインスタンスが作成されます。

スクラッチパッドはすべての UDF の呼び出しの情報を保管するのに役立ちます。Java UDF はインスタンス変数を使用するか、あるいは、スクラッチパッドを設定して、呼び出し間に継続性を持たせることができます。Java UDF は com.ibm.db2.app.UDF で使用可能な getScratchPad および setScratchPad メソッドによってスクラッチパッドにアクセスします。CREATE FUNCTION ステートメントに FINAL CALL オプションを指定している場合は、照会の終了で、オブジェクトの public void close() メソッドが呼び出されま

す (DB2GENERAL パラメーター・スタイルの関数の場合)。このメソッドを定義していないと、stub 関数  
 がその後を引き継ぎ、イベントは無視されます。 com.ibm.db2.app.UDF クラスには、DB2GENERAL パラ  
 メーター・スタイルの UDF で使用できる有用な変数とメソッドが含まれています。これらの変数とメソッ  
 ドについては、以下の表で説明します。

| 変数およびメソッド  | 説明  |
|--|---|
| <ul style="list-style-type: none"> <li>• public static final int SQLUDF_FIRST_CALL = -1;</li> <li>• public static final int SQLUDF_NORMAL_CALL = 0;</li> <li>• public static final int SQLUDF_TF_FIRST = -2;</li> <li>• public static final int SQLUDF_TF_OPEN = -1;</li> <li>• public static final int SQLUDF_TF_FETCH = 0;</li> <li>• public static final int SQLUDF_TF_CLOSE = 1;</li> <li>• public static final int SQLUDF_TF_FINAL = 2;</li> </ul>  | スカラー UDF の場合、これらは、呼び出しが最初の呼び出しか、通常の呼び出しかを判別するための定数です。テーブル UDF の場合は、呼び出しが、最初の呼び出し、オープン呼び出し、フェッチ呼び出し、クローズ呼び出し、あるいは最終呼び出しのいずれであるかを判別するための定数です。   |
| public Connection getConnection();   | このメソッドは、このストアード・プロシージャ呼び出しのための JDBC 接続ハンドルを取得し、呼び出し側アプリケーションのデータベースへの接続を表す JDBC オブジェクトを戻します。それは、C ストアード・プロシージャにおけるヌルの SQLConnect() 呼び出しの結果に似ています。   |
| public void close();   | UDF が FINAL CALL オプションを指定して作成された場合、このメソッドは UDF 評価の終わりに、データベースにより呼び出されます。それは、C の UDF の場合の最終呼び出しに似ています。Java UDF クラスがこのメソッドを実装していない場合には、このイベントは無視されます。   |
| public boolean isNull(int i)   | このメソッドは、与えられた指標の入力引数が SQL NULL かどうかをテストします。   |
| <ul style="list-style-type: none"> <li>• public void set(int i, short s);</li> <li>• public void set(int i, int j);</li> <li>• public void set(int i, long j);</li> <li>• public void set(int i, double d);</li> <li>• public void set(int i, float f);</li> <li>• public void set(int i, BigDecimal bigDecimal);</li> <li>• public void set(int i, String string);</li> <li>• public void set(int i, Blob blob);</li> <li>• public void set(int i, Clob clob);</li> <li>• public boolean needToSet(int i);</li> </ul> | このメソッドは、出力引数を、与えられた値に設定します。次のような何らかの問題が生じると、例外が投げられます。 <ul style="list-style-type: none"> <li>• UDF 呼び出しが進行していない。</li> <li>• 指標が有効な出力引数を参照していない。</li> <li>• データ・タイプが一致していない。</li> <li>• データ長が一致していない。</li> <li>• コード・ページ変換のエラーが発生した。</li> </ul> |

| 変数およびメソッド  | 説明   |
|--|--|
| <pre>public void setSQLstate(String string);</pre>   | <p>このメソッドは UDF により呼び出され、この呼び出しから SQLSTATE が戻されるように設定します。 string が SQLSTATE として許容外の場合は、例外がスローされます。ユーザーは SQLSTATE を外部プログラムで設定し、関数からエラーまたは警告が戻されるようにすることができます。この場合、SQLSTATE には以下のいずれかが入ります。</p> <ul style="list-style-type: none"> <li>• '00000'。これは成功を示します。</li> <li>• '01Hxx'。ここで、xx は任意の 2 桁の数または英大文字で、警告を示します。</li> <li>• '38yxx'。ここで、y は 'I' から 'Z' の英大文字、xx は任意の 2 桁の数または英大文字で、エラーを示します。</li> </ul> |
| <pre>public void setSQLmessage(String string);</pre>   | <p>このメソッドは setSQLstate メソッドに類似しています。これは SQL メッセージ結果を設定します。string が許容外 (たとえば 70 文字を超えている) の場合は、例外がスローされます。</p>  |
| <pre>public String getFunctionName();</pre>  | <p>このメソッドは、処理中の UDF の名前を戻します。</p>  |
| <pre>public String getSpecificName();</pre>  | <p>このメソッドは、処理中の UDF の特定名を戻します。</p>   |
| <pre>public byte[] getDBinfo();</pre>  | <p>このメソッドは、処理中の UDF についての未加工・未処理の DBINFO 構造を、バイト配列として戻します。CREATE FUNCTION を使用し、DBINFO オプションを付けて、UDF を登録しておく必要があります。</p>  |
| <ul style="list-style-type: none"> <li>• public String getDBname();</li> <li>• public String getDBauthid();</li> <li>• public String getDBver_rel();</li> <li>• public String getDBplatform();</li> <li>• public String getDBapplid();</li> <li>• public String getDBapplid();</li> <li>• public String getDBtbschema();</li> <li>• public String getDBtbname();</li> <li>• public String getDBcolname();</li> </ul> | <p>これらのメソッドは、処理中の UDF の DBINFO 構造から、該当するフィールドの値を戻します。CREATE FUNCTION を使用し、DBINFO オプションを付けて、UDF を登録しておく必要があります。</p> <p>getDBtbschema()、getDBtbname()、getDBcolname() の各メソッドは、UPDATE ステートメントの SET 文節の右辺にユーザー定義関数が指定されている場合のみ、意味のある情報を戻します。</p>  |
| <pre>public int getCCSID();</pre>  | <p>このメソッドは、ジョブの CCSID を戻します。</p>   |
| <pre>public byte[] getScratchpad();</pre>  | <p>このメソッドは、現在処理中の UDF のスクラッチパッドのコピーを戻します。まず、SCRATCHPAD オプションを付けて UDF を宣言する必要があります。</p>   |
| <pre>public void setScratchpad(byte ab[]);</pre>   | <p>このメソッドは、現在処理中の UDF のスクラッチパッドを、与えられたバイト配列の内容で上書きします。まず、SCRATCHPAD オプションを付けて UDF を宣言する必要があります。バイト配列は、getScratchpad() の戻りと同じサイズでなければなりません。</p>   |

| 変数およびメソッド                            | 説明  |
|--------------------------------------|---|
| <pre>public int getCallType();</pre> | <p>このメソッドは、現在行われている呼び出しのタイプを戻します。これらの値は、<code>sqludf.h</code> に定義されている C の値に対応しています。戻される可能性のある値は以下のとおりです。</p> <ul style="list-style-type: none"> <li>• SQLUDF_FIRST_CALL</li> <li>• SQLUDF_NORMAL_CALL</li> <li>• SQLUDF_TF_FIRST</li> <li>• SQLUDF_TF_OPEN</li> <li>• SQLUDF_TF_FETCH</li> <li>• SQLUDF_TF_CLOSE</li> <li>• SQLUDF_TF_FINAL</li> </ul> |

### Java ユーザー定義関数に関する制約事項:

これらの制限は、Java ユーザー定義関数 (UDF) に適用されます。

- Java UDF が追加のスレッドを作成しないようにする必要があります。追加のスレッドを作成できるのは、ジョブがマルチスレッド可能な場合だけです。SQL ストアド・プロシージャを呼び出すジョブがマルチスレッド可能かどうかは保証できないので、Java ストアド・プロシージャは追加のスレッドを作成できません。
- データベースに定義する Java ストアド・プロシージャの完全名は、279 文字に限定されている。この制限は EXTERNAL\_NAME 列に基づきます。この列の最大幅は 279 文字です。
- 借用権限を使用して Java クラス・ファイルにアクセスできない。
- Java UDF は、システムにインストールされている最新バージョンの JDK を常に使用する。
- Blob クラスと Clob クラスは `java.sql` パッケージと `com.ibm.db2.app` パッケージの両方にあるので、同一のプログラム中でこれらの両方のクラスを使用する場合は、プログラマーはこれらのクラスの名前全体を使用しなければならない。 `com.ibm.db2.app` 中の Blob クラスと Clob クラスが、ストアド・プロシージャに渡されるパラメーターとして使用されていることを、プログラムが確認しなければなりません。
- Java UDF を作成する際には、ソース関数の場合と同様に、ライブラリー中のサービス・プログラムを使用して関数の定義を格納する。サービス・プログラムの名前は、システムによって生成され、関数を作成したジョブのジョブ・ログ中にあります。このオブジェクトを保管してから別のシステムに復元すると、関数の定義も復元されます。Java UDF をシステム間で移動する場合は、Java クラスが含まれる統合ファイル・システムに加えて、関数の定義が含まれるサービス・プログラムも移動する必要があります。
- データベースへの接続に使用する JDBC 接続のプロパティ (システムの命名など) を、Java UDF で設定できない。事前取り出しが使用不可の場合を除き、デフォルトの JDBC 接続プロパティが常に使用されます。

### Java ユーザー定義テーブル関数:

DB2 は、テーブルを戻す機能を関数に提供します。この機能は、データベースの外部からデータベースにテーブル形式で情報を公開するのに役立ちます。たとえば、Java ストアド・プロシージャと Java UDF (テーブルとスカラーの両方) で使用される、Java 仮想マシン (JVM) 中のプロパティ設定を公開するテーブルを作成できます。

SQLJ Part 1: SQL Routines 規格はテーブル関数をサポートします。したがって、テーブル関数を使用できるのは、パラメーター・スタイル DB2GENERAL を使用する場合だけです。

テーブル関数に対して 5 種類の呼び出しが行われます。以下の表にこれらの呼び出しが説明されています。以下の内容は、関数作成 SQL ステートメント上でスクラッチパッドが指定されていることを前提にしています。

| スキャンする時点                  | NO FINAL CALL LANGUAGE<br>JAVA SCRATCHPAD                             | FINAL CALL LANGUAGE JAVA<br>SCRATCHPAD                                 |
|---------------------------|---|--|
| テーブル関数を初めて OPEN する前       | 呼び出しなし  | クラス・コンストラクターが呼び出されます (新しいスクラッチパッドを意味します)。FIRST 呼び出しで UDF メソッドが呼び出されます。 |
| 毎回のテーブル関数の OPEN 時         | クラス・コンストラクターが呼び出されます (新しいスクラッチパッドを意味します)。OPEN 呼び出しで UDF メソッドが呼び出されます。 | OPEN 呼び出しで UDF メソッドが呼び出されます。   |
| 毎回のテーブル関数データの新規行の FETCH 時 | FETCH 呼び出しで UDF メソッドが呼び出されます。   | FETCH 呼び出しで UDF メソッドが呼び出されます。  |
| 毎回のテーブル関数の CLOSE 時        | CLOSE 呼び出しで UDF メソッドが呼び出されます。close() メソッドがある場合は呼び出されます。               | CLOSE 呼び出しで UDF メソッドが呼び出されます。  |
| テーブル関数を最後に CLOSE した後      | 呼び出しなし  | FINAL 呼び出しで UDF メソッドが呼び出されます。close() メソッドがある場合は呼び出されます。                |

## 例: Java テーブル関数

Java ユーザー定義テーブル関数の実行時に JVM 中のどのプロパティ設定を使用するかを判別する、Java テーブル関数の例を以下に示します。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
import com.ibm.db2.app.*;
import java.util.*;

public class JVMProperties extends UDF {
    Enumeration propertyNames;
    Properties properties ;

    public void dump (String property, String value) throws Exception
    {
        int callType = getCallType();
        switch(callType) {
            case SQLUDF_TF_FIRST:
                break;
            case SQLUDF_TF_OPEN:
                properties = System.getProperties();
                propertyNames = properties.propertyNames();
                break;
            case SQLUDF_TF_FETCH:
                if (propertyNames.hasMoreElements()) {
                    property = (String) propertyNames.nextElement();
                    value = properties.getProperty(property);
                    set(1, property);
                    set(2, value);
                }
            }
    }
}
```



```

        } else {
            setSQLstate("02000");
        }
        break;
    case SQLUDF_TF_CLOSE:
        break;
    case SQLUDF_TF_FINAL:
        break;
    default:
        throw new Exception("UNEXPECT call type of "+callType);
    }
}
}
}

```

テーブル関数をコンパイルして、そのクラス・ファイルを /QIBM/UserData/OS400/SQLLib/Function にコピーした後に、以下の SQL ステートメントを使用してその関数をデータベースに登録できます。

```

create function properties()
returns table (property varchar(500), value varchar(500))
external name 'JVMPProperties.dump' language java
parameter style db2general fenced no sql
disallow parallel scratchpad

```

関数を登録した後で、SQL ステートメントの一部として使用できます。たとえば、以下の SELECT ステートメントは、テーブル関数によって生成されたテーブルを戻します。

```

SELECT * FROM TABLE(PROPERTIES())

```

## JAR ファイルを操作する SQLJ プロシージャ

Java ストアード・プロシージャと Java UDF は両方とも、Java JAR ファイルに保管されている Java クラスを使用できます。

JAR ファイルを使用するには、*jar-id* を JAR ファイルに関連付ける必要があります。 *jar-ids* および JAR ファイルの操作を可能にする、SQLJ スキーマによるストアード・プロシージャが用意されています。それらのプロシージャにより、JAR ファイルのインストール、置換、および除去が可能です。また、JAR ファイルに関連した SQL カタログの使用と更新を行う機能もあります。

### SQLJ.INSTALL\_JAR:

SQLJ.INSTALL\_JAR ストアード・プロシージャは、JAR ファイルをデータベース・システム中にインストールします。後続の CREATE FUNCTION および CREATE PROCEDURE ステートメント中で、この JAR ファイルを使用できます。

## 権限

SYSJAROBJECTS および SYSJARCONTENTS カタログ・テーブルに関する以下の特権のうち 1 つ以上が、CALL ステートメントの許可 ID で保持されていなければなりません。

- 以下のシステム権限:
  - テーブルに対する INSERT および SELECT 特権
  - ライブラリー QSYS2 に対するシステム権限 \*EXECUTE
- 管理権限

以下の特権も、CALL ステートメントの許可 ID で保持されていなければなりません。

- *jar-url* パラメーターで指定されている、インストール対象の JAR ファイルに対する読み取り (\*R) アクセス権。



- JAR ファイルのインストール・ディレクトリーに対する書き込み、実行、および読み取り (\*RWX) アクセス権。このディレクトリーは /QIBM/UserData/OS400/SQLLib/Function/jar/schema (schema は jar-id のスキーマ) です。

これらの権限には、借用権限を使用できません。

## SQL 構文

```
>>-CALL--SQLJ.INSTALL_JAR-- (--'jar-url'--,--'jar-id'--,--deploy--)-->
>-----<
```

## 説明

**jar-url** インストールまたは置換対象の JAR ファイルを含む URL。サポートされている URL スキームは 'file:' だけです。

**jar-id** jar-url によって指定されたファイルに関連した、データベース中の JAR ID。jar-id には SQL 命名構文が使用され、JAR ファイルは暗黙または明示の修飾子によって指定されたスキーマやライブラリーにインストールされます。

**deploy** 配置記述子ファイルの install\_action の説明に使用される値。この整数がゼロ以外の値の場合は、install\_jar プロシーチャーの終了時に、配置記述子ファイルの install\_actions が実行される必要があります。ゼロの値をサポートしているのは現行バージョンの DB2 UDB for iSeries だけです。

## 使用上の注意

JAR ファイルのインストール時に、DB2 UDB for iSeries は SYSJAROBJECTS システム・カタログ中にその JAR ファイルを登録します。さらに、JAR ファイルから Java<sup>TM</sup> クラス・ファイルの名前を抽出し、個々のクラスを SYSJARCONTENTS システム・カタログ中に登録します。DB2 UDB for iSeries は、JAR ファイルを /QIBM/UserData/OS400/SQLLib/Function ディレクトリーの jar/schema サブディレクトリーにコピーします。DB2 UDB for iSeries は、jar-id 文節で指定されている名前を新しい JAR ファイルのコピーに付けます。DB2 UDB for iSeries によって /QIBM/UserData/OS400/SQLLib/Function/jar のサブディレクトリーにインストールされた JAR ファイルには、変更を加えることができません。その代わりに、CALL SQLJ.REMOVE\_JAR コマンドと CALL SQLJ.REPLACE\_JAR SQL コマンドを使用して、インストールされている JAR ファイルを除去するか置き換える必要があります。

## 例

以下のコマンドは、SQL 対話式セッションから発行されます。

```
CALL SQLJ.INSTALL_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar', 0)
```

file:/home/db2inst/classes/ ディレクトリーにある Proc.jar ファイルは、DB2 UDB for iSeries 中に myproc\_jar という名前でインストールされます。Procedure.jar ファイルを使用するそれ以降の SQL コマンドは、名前 myproc\_jar を使用して参照します。

## SQLJ.REMOVE\_JAR:

SQLJ.REMOVE\_JAR ストアード・プロシーチャーは、JAR ファイルをデータベース・システムから除去します。

## 権限

SYSJARCONTENTS および SYSJAROBJECTS カタログ・テーブルに関する以下の特権のうち 1 つ以上が、CALL ステートメントの許可 ID で保持されていなければなりません。

- 以下のシステム権限:
  - テーブルに対する SELECT および DELETE 特権
  - ライブラリー QSYS2 に対するシステム権限 \*EXECUTE
- 管理権限

以下の特権も、CALL ステートメントの許可 ID で保持されていなければなりません。

- 除去される JAR ファイルに対する \*OBJMGT 権限。この JAR ファイルの名前は /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile になります。

この権限には、借用権限を使用できません。

## 構文

```
>>-CALL--SQLJ.REMOVE_JAR(--'jar-id'--,--undeploy--)------><
```

## 説明

*jar-id* データベースから除去される JAR ファイルの JAR ID。

### *undeploy*

配置記述子ファイルの `remove_action` の説明に使用される値。この整数がゼロ以外の値の場合は、`install_jar` プロシーチャーの終了時に、配置記述子ファイルの `remove_actions` が実行される必要があります。ゼロの値をサポートしているのは現行バージョンの DB2 UDB for iSeries だけです。

## 例

以下のコマンドは、SQL 対話式セッションから発行されます。

```
CALL SQLJ.REMOVE_JAR('myProc_jar', 0)
```

JAR ファイル `myProc_jar` がデータベースから除去されます。

## SQLJ.REPLACE\_JAR:

SQLJ.REPLACE\_JAR ストアード・プロシーチャーは、データベース・システム中の JAR ファイルを置換します。

## 権限

SYSJAROBJECTS および SYSJARCONTENTS カタログ・テーブルに関する以下の特権のうち 1 つ以上が、CALL ステートメントの許可 ID で保持されていなければなりません。

- 以下のシステム権限:
  - テーブルに対する SELECT、INSERT、および DELETE 特権
  - ライブラリー QSYS2 に対するシステム権限 \*EXECUTE
- 管理権限

以下の特権も、CALL ステートメントの許可 ID で保持されていなければなりません。

- *jar-url* パラメーターで指定されている、インストール対象の JAR ファイルに対する読み取り (\*R) アクセス権。
- 除去される JAR ファイルに対する \*OBJMGT 権限。この JAR ファイルの名前は /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile になります。

これらの権限には、借用権限を使用できません。

## 構文

```
>>-CALL--SQLJ.REPLACE_JAR--(--'jar-url'--,--'jar-id'--)-----<
```

## 説明

*jar-url* 置換対象の JAR ファイルを含む URL。サポートされている URL スキームは 'file:' だけです。

*jar-id* *jar-url* によって指定されたファイルに関連した、データベース中の JAR ID。 *jar-id* には SQL 命名構文が使用され、JAR ファイルは暗黙または明示の修飾子によって指定されたスキーマやライブラリーにインストールされます。

## 使用上の注意

SQLJ.REPLACE\_JAR ストアード・プロシージャは、以前に SQLJ.INSTALL\_JAR を使用してデータベースにインストールされた JAR ファイルを置換します。

## 例

以下のコマンドは、SQL 対話式セッションから発行されます。

```
CALL SQLJ.REPLACE_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar')
```

*jar-id* myproc\_jar によって参照される現行の JAR ファイルが、file:/home/db2inst/classes/ ディレクトリー中の Proc.jar ファイルに置換されます。

## SQLJ.UPDATEJARINFO:

SQLJ.UPDATEJARINFO は、SYSJARCONTENTS カタログ・テーブルの CLASS\_SOURCE 列を更新します。このプロシージャは、SQLJ 規格の一部ではありませんが、DB2 UDB for iSeries ストアード・プロシージャ・ビルダーによって使用されます。

## 権限

SYSJARCONTENTS カタログ・テーブルに関する以下の特権のうち 1 つ以上が、CALL ステートメントの許可 ID で保持されていなければなりません。

- 以下のシステム権限:
  - テーブルに対する SELECT および UPDATEINSERT 特権
  - ライブラリー QSYS2 に対するシステム権限 \*EXECUTE
- 管理権限

CALL ステートメントを実行するユーザーに、以下の権限もなければなりません。

- *jar-url* パラメーターで指定されている JAR ファイルに対する読み取り (\*R) アクセス権。インストール対象の JAR ファイルに対する読み取り (\*R) アクセス権。
- JAR ファイルのインストール・ディレクトリーに対する書き込み、実行、および読み取り (\*RWX) アクセス権。このディレクトリーは /QIBM/UserData/OS400/SQLLib/Function/jar/schema (*schema* は *jar-id* のスキーマ) です。

これらの権限には、借用権限を使用できません。

## 構文

```
>>-CALL--SQLJ.UPDATEJARINFO--(--'jar-id'--,--'class-id'--,--'jar-url'--)-->-----<
```

## 説明

*jar-id* データベース中の、更新される JAR ID。

### *class-id*

更新されるクラスのパッケージ修飾クラス名。

*jar-url* JAR ファイルの更新に使用するクラス・ファイルを含む URL。サポートされている URL スキームは 'file:' だけです。

## 例

以下のコマンドは、SQL 対話式セッションから発行されます。

```
CALL SQLJ.UPDATEJARINFO('myproc_jar', 'mypackage.myclass',  
                          'file:/home/user/mypackage/myclass.class')
```

*jar-id* myproc\_jar に関連した JAR ファイルが、新しいバージョンの mypackage.myclass クラスによって更新されます。新しいバージョンのクラスは、file:/home/user/mypackage/myclass.class から入手されます。

## SQLJ.RECOVERJAR:

SQLJ.RECOVERJAR プロシージャは、SYSJAROBJECTS カタログに格納されている JAR ファイルを取り出し、/QIBM/UserData/OS400/SQLLib/Function/jar/jarschema/jar\_id.jar ファイルに復元します。

## 権限

SYSJAROBJECTS カタログ・テーブルに関する以下の特権のうち 1 つ以上が、CALL ステートメントの許可 ID で保持されていなければなりません。

- 以下のシステム権限:
  - テーブルに対する SELECT および UPDATEINSERT 特権
  - ライブラリー QSYS2 に対するシステム権限 \*EXECUTE
- 管理権限

CALL ステートメントを実行するユーザーに、以下の権限もなければなりません。

- JAR ファイルのインストール・ディレクトリーに対する書き込み、実行、および読み取り (\*RWX) アクセス権。このディレクトリーは /QIBM/UserData/OS400/SQLLib/Function/jar/schema (schema は jar-id のスキーマ) です。
- 除去される JAR ファイルに対する \*OBJMGT 権限。この JAR ファイルの名前は /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile になります。

## 構文

```
>>-CALL---SQLJ.RECOVERJAR---(--'jar-id'---)-----<<
```

## 説明

*jar-id* データベース中の、リカバリーされる JAR ID。

## 例

以下のコマンドは、SQL 対話式セッションから発行されます。

```
CALL SQLJ.UPDATEJARINFO('myproc_jar')
```

myproc\_jar と関連した JAR ファイルは、SYSJARCONTENT テーブルの内容で更新されます。このファイルは、/QIBM/UserData/OS400/SQLLib/Function/jar/jar\_schema myproc\_jar.jar にコピーされます。

### SQLJ.REFRESH\_CLASSES:

SQLJ.REFRESH\_CLASSES ストアド・プロシージャを使用すると、現行のデータベース接続で Java ストアド・プロシージャまたは Java UDF が使用しているユーザー定義クラスが再ロードされます。SQLJ.REPLACE\_JAR ストアド・プロシージャの呼び出しによって加えられた変更を取得するためには、既存のデータベース接続によってこのストアド・プロシージャが呼び出されなければなりません。

### 権限

なし

### 構文

```
>>-CALL--SQLJ.REFRESH_CLASSES-- ()-->
>-----<
```

### 例

MYJAR jarid で登録された jar ファイル内のクラスを使用する、Java ストアド・プロシージャ、MYPROCEDURE を呼び出します。

```
CALL MYPROCEDURE()
```

以下の呼び出しを使用して jar ファイルを置き換えます。

```
CALL SQLJ.REPLACE_JAR('MYJAR', '/tmp/newjarfile.jar')
```

後の MYPROCEDURE ストアド・プロシージャの呼び出しで、更新された jar ファイルを使用するためには、SQLJ.REFRESH\_CLASSES が呼び出されなければなりません。

```
CALL SQLJ.REFRESH_CLASSES()
```

ストアド・プロシージャを再び呼び出します。プロシージャが呼び出されると、更新されたクラス・ファイルが使用されます。

```
CALL MYPROCEDURE()
```

### Java ストアド・プロシージャおよび UDF 用のパラメーター引き渡し規則

以下の表には、Java ストアド・プロシージャと UDF 中で SQL データ・タイプが表される方法がリストされています。

| SQL データ・タイプ       | Java パラメーター・スタイル JAVA | Java パラメーター・スタイル DB2GENERAL |
|-------------------|-----------------------|-----------------------------|
| SMALLINT          | short                 | short                       |
| INTEGER           | int                   | int                         |
| BIGINT            | long                  | long                        |
| DECIMAL(p,s)      | BigDecimal            | BigDecimal                  |
| NUMERIC(p,s)      | BigDecimal            | BigDecimal                  |
| REAL または FLOAT(p) | float                 | float                       |

| SQL データ・タイプ                         | Java パラメーター・スタイル JAVA | Java パラメーター・スタイル DB2GENERAL |
|-------------------------------------|-----------------------|-----------------------------|
| DOUBLE PRECISION、FLOAT、または FLOAT(p) | double                | double                      |
| CHARACTER(n)                        | String                | String                      |
| CHARACTER(n) FOR BIT DATA           | byte[]                | com.ibm.db2.app.Blob        |
| VARCHAR(n)                          | String                | String                      |
| VARCHAR(n) FOR BIT DATA             | byte[]                | com.ibm.db2.app.Blob        |
| GRAPHIC(n)                          | String                | String                      |
| VARGRAPHIC(n)                       | String                | String                      |
| DATE                                | Date                  | String                      |
| TIME                                | Time                  | String                      |
| TIMESTAMP                           | Timestamp             | String                      |
| 標識変数                                | -                     | -                           |
| CLOB                                | -                     | com.ibm.db2.app.Clob        |
| BLOB                                | -                     | com.ibm.db2.app.Blob        |
| DBCLOB                              | -                     | com.ibm.db2.app.Clob        |
| DataLink                            | -                     | -                           |

## Java と他のプログラム言語

Java には、Java 以外の言語で作成されたコードを呼び出す方法がいくつかあります。

### Java ネイティブ・インターフェース

別の言語で作成されたコードを呼び出す 1 つの方法は、選択された Java メソッドを「ネイティブ・メソッド」として実装することです。ネイティブ・メソッドは、別の言語で作成されたプロシージャで、Java メソッドの実際の実装方法を示します。ネイティブ・メソッドは、Java ネイティブ・インターフェース (JNI) を使用して Java 仮想マシンにアクセスできます。これらネイティブ・メソッドは Java スレッドで実行されます。Java スレッドはカーネル・スレッドなので、ネイティブ・メソッドはスレッド・セーフなものである必要があります。関数は同一プロセス内のマルチスレッドで同時に開始できる場合、スレッド・セーフなものであると言えます。また、その関数内で呼び出されるすべての関数もスレッド・セーフなものである場合のみ、スレッド・セーフです。

ネイティブ・メソッドは、Java では直接サポートされていないシステム機能をアクセスしたり、既存のユーザー・コードとインターフェースするための「橋」のような役割を果たします。ネイティブ・メソッドを使用する際には、注意を要します。なぜなら、呼び出されるコードはスレッド・セーフなものではない場合があるからです。JNI および ILE ネイティブ・メソッドについて詳しくは、ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用するを参照してください。

### Java 呼び出し API

Java ネイティブ・インターフェース (JNI) 仕様の一部である Java 呼び出し API を使うと、Java 以外のアプリケーションで Java 仮想マシンを使用できます。また、Java コードをアプリケーションの拡張機能として使用することを可能にします。

### i5/OS PASE ネイティブ・メソッド



iSeries Java 仮想マシン (JVM) は、i5/OS PASE 環境で実行するネイティブ・メソッドの使用をサポートしています。Java用の i5/OS PASE ネイティブ・メソッドでは、AIX® で実行する Java アプリケーションを iSeries サーバーに容易にポータリングすることができます。クラス・ファイルと AIX ネイティブ・メソッド・ライブラリーを iSeries 上の統合ファイル・システムにコピーし、制御言語 (CL)、Qshell、または i5/OS PASE のいずれかの端末セッション・コマンド・プロンプトからそれらを実行できます。

### Teraspace ネイティブ・メソッド

iSeries Java 仮想マシン (JVM) は現在、Teraspace ストレージ・モデル・ネイティブ・メソッドの使用をサポートしています。Teraspace ストレージ・モデルは、ILE プログラム用の大規模処理のローカル・アドレス環境を提供します。Teraspace を使用すれば、ソース・コードをほとんどあるいは全く変更しないで、ネイティブ・メソッド・コードを他のオペレーティング・システムから i5/OS に移植することができます。

### java.lang.Runtime.exec()

java.lang.Runtime.exec() を使用して、Java プログラム内でプログラムまたはコマンドを呼び出すことができます。exec() メソッドは、任意の iSeries プログラムまたはコマンドを実行する別のプロセスを開始します。このモデルでは、プロセス間通信を行うのに、子プロセスの標準 in、標準 out、および標準 err を使用できます。

### プロセス間通信

親および子プロセスの間のプロセス間通信の 1 つの方法は、ソケットを使用することです。

また、プログラム間通信を行うのに、ストリーム・ファイルを使用することもできます。別のプロセスで実行中のプログラムと通信する方法の概要については、プロセス間通信の例を参照してください。

他の言語から Java を呼び出す場合の詳細は、例: C から Java を呼び出すまたは例: RPG から Java を呼び出すを参照してください。

IBM Toolbox for Java を使用して、iSeries サーバー上の既存のプログラムやコマンドを呼び出すこともできます。IBM Toolbox for Java を使ったプロセス間通信には、通常、データ待ち行列および iSeries メッセージが使用されます。

注: Runtime.exec()、IBM Toolbox for Java、または JNI のいずれかを使用することにより、Java プログラムの可搬性が損なわれることがあります。"pure" Java 環境では、これらのメソッドを使用しないでください。

## ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用する

ネイティブ・メソッドは、Pure Java ではプログラミングの要件を満たすことができない場合にのみ使用してください。

ネイティブ・メソッドの使用を制限し、それらを次の状況でのみ使用するようにします。

- Pure Java では使用できないシステム機能にアクセスする場合。
- パフォーマンスへの依存度が極めて高く、ネイティブ実装によるメリットが大きいメソッドを実装する場合。
- Java が別の API を呼び出すことを可能にする既存のアプリケーション・プログラム・インターフェース (API) とインターフェースする場合。



以下の指示は、C 言語での Java Native Interface (JNI) の使用に当てはまります。RPG 言語での JNI の使用について詳しくは、以下の資料を参照してください。

「WebSphere Development Studio: ILE RPG プログラマーの手引き (SD88-5042-04)」の第 11 章。

ネイティブ・メソッドのために Java ネイティブ・インターフェース (JNI) を使用するには、以下のステップに従ってください。

1. 標準の Java 言語構文を使用して、どのメソッドがネイティブ・メソッドであるかメソッドを指定することにより、クラスを設計します。
2. ネイティブ・メソッドの実装を含むサービス・プログラム (\*SRVPGM) のライブラリーとプログラム名を決定します。クラスの静的な初期化指定子で System.loadLibrary() メソッド呼び出しをコーディングする際には、サービス・プログラムの名前を指定してください。
3. javac ツールを使用して Java ソースをコンパイルすることにより、クラス・ファイルを作成します。
4. javah ツールを使用して、ヘッダー・ファイル (.h) を作成します。このヘッダー・ファイルには、ネイティブ・メソッドの実装を作成するための正確なプロトタイプが含まれます。ヘッダー・ファイルを作成するディレクトリーは、-d オプションで指定します。
5. 「ストリーム・ファイルからのコピー (CPYFRMSTMF)」コマンドを使用して、ヘッダー・ファイルを統合ファイル・システムからソース・ファイル内のメンバーにコピーします。C コンパイラーでヘッダー・ファイルを使用するには、それをソース・ファイル・メンバーにコピーする必要があります。C ソースおよび C ヘッダー・ファイルを統合ファイル・システムに残すには、「C モジュール作成 (CRTCMOD)」コマンドの新しいストリーム・ファイル・サポートを使用してください。CRTCMOD コマンドとストリーム・ファイルの使用法について詳しくは、「WebSphere Development Studio: ILE C/C++ Programmer's Guide (SC09-2712)」を参照してください。
6. ネイティブ・メソッド・コードを作成します。ネイティブ・メソッドのために使用される言語および関数の詳細については、Java ネイティブ・メソッドおよびスレッドに関する考慮事項を参照してください。
  - a. 前のステップで作成したヘッダー・ファイルを組み込みます。
  - b. ヘッダー・ファイル内のプロトタイプと正確に一致させます。
  - c. スtringを Java 仮想マシンに渡す場合は、ASCII コードに変換します。詳細については、Java 文字のエンコードを参照してください。
7. ネイティブ・メソッドが Java 仮想マシンと対話しなければならない場合は、JNI によって提供される関数を使用します。
8. CRTCMOD コマンドを使用して、C ソース・コードをコンパイルして、モジュール (\*MODULE) オブジェクトを作成します。
9. 「サービス・プログラム作成 (CRTSRVPGM)」コマンドを使用して、1 つまたは複数のモジュール・オブジェクトをバインドし、サービス・プログラム (\*SRVPGM) を作成します。このサービス・プログラムの名前は、System.load() または System.loadLibrary() 関数呼び出しの Java コードで指定した名前と同じでなければなりません。
10. Java コードで System.loadLibrary() 呼び出しを使用した場合は、以下のうち、実行している J2SDK に適したいずれかの操作を行ってください。
  - 必要なライブラリーのリストを LIBPATH 環境変数に組み込みます。LIBPATH 環境変数は、QShell で、および iSeries コマンド行から変更することができます。
    - Qshell コマンド・プロンプトから、次のように入力します。

```
export LIBPATH=/QSYS.LIB/MYLIB.LIB
java -Djava.version=1.5 myclass
```

|       - あるいは、コマンド行から、次のように入力します。

```
|       ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'  
|       JAVA PROP((java.version 1.5)) myclass
```

|       • あるいは、**java.library.path** プロパティでリストを提供します。 `java.library.path` プロパティは、QShell で、および iSeries コマンド行から変更することができます。

|       - Qshell コマンド・プロンプトから、次のように入力します。

```
|       java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.5 myclass
```

|       - あるいは、iSeries コマンド行から、次のように入力します。

```
|       JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.5')) myclass
```

|       ここで、`/QSYS.LIB/MYLIB.LIB` は、`System.loadLibrary()` 呼び出しを使用してロードしたいライブラリーで、`myclass` は Java アプリケーションの名前です。

11. `System.load(String path)` の `path` の構文は、次のいずれかにすることができます。

- `/qsys.lib/sysNMsp.srvpgm` (\*SRVPGM QSYS/SYSNMSP の場合)
- `/qsys.lib/mylib.lib/myNMsp.srvpgm` (\*SRVPGM MYLIB/MYNMSP の場合)
- `/qsys.lib/mylib.lib/myNMsp.srvpgm` にリンクする `/home/mydir/myNMsp.srvpgm` などのシンボリック・リンク

注: これは、`System.loadLibrary("myNMsp")` メソッドを使用することと同等です。

注: パス名は通常、引用符で囲んだストリング・リテラルです。たとえば、次のようなコードが使用できます。

```
System.load("/qsys.lib/mylib.lib/myNMsp.srvpgm")
```

12. `System.loadLibrary(String libname)` の `libname` パラメーターは通常、ネイティブ・メソッド・ライブラリーを示す、引用符で囲んだストリング・リテラルです。システムは、現行のライブラリーのリストと `LIBPATH` および `PASE_LIBPATH` 環境変数を使用して、そのライブラリー名と一致するサービス・プログラムまたは i5/OS PASE 実行可能プログラムを検索します。たとえば、`loadLibrary("myNMsp")` を使用すると、`MYNMSP` という名前の \*SRVPGM か、`libmyNMsp.a` または `libmyMNsp.so` という名前の i5/OS PASE 実行可能プログラムが検索されます。

JNI の詳細については、Java Native Interface by Sun Microsystems, Inc.、および The Source for Java Technology [java.sun.com](http://java.sun.com) を参照してください。

ネイティブ・メソッドのために JNI を使用方法の例については、例: ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用するを参照してください。

## Java 呼び出し API

Java ネイティブ・インターフェース (JNI) の一部である呼び出し API を使用すると、Java 以外のコードで Java 仮想マシン (JVM) を作成し、Java クラスをロードおよび使用することができます。この機能により、マルチスレッド化されたプログラムは、1 つの Java 仮想マシンで実行されている Java クラスを複数のスレッドで使用できるようになります。

IBM Developer Kit for Java は、以下のタイプの呼び出し元の Java 呼び出し API をサポートしています。

- STGMDL(\*SNGLVL) および DTAMD(\*P128) 用に作成された ILE プログラムまたはサービス・プログラム

- STGMDL(\*TERASPACE) および DTAMDLL(\*LLP64) 用に作成された ILE プログラムまたはサービス・プログラム
- 32 ビットまたは 64 ビット AIX 用に作成された i5/OS PASE 実行可能プログラム

Java 仮想マシンは、アプリケーションによって制御されます。アプリケーションでは、Java 仮想マシンを作成し、Java メソッドを呼び出し (アプリケーションがサブルーチンを呼び出すのと類似した方法で)、Java 仮想マシンを破棄することができます。Java 仮想マシンを作成すると、それは、アプリケーションによって明示的に破棄されるまで、プロセス内で実行可能な状態のまま残ります。Java 仮想マシンの破棄時には、ファイナライザーの実行、Java 仮想マシン・スレッドの終了、および Java 仮想マシン・リソースの解放などの終結処理が行われます。

実行可能な状態の Java 仮想マシンがあれば、C や RPG などの ILE 言語で書かれたアプリケーションは、その Java 仮想マシンを呼び出して関数を実行することができます。また、Java 仮想マシンから C アプリケーションに戻ったり、Java 仮想マシンを再び呼び出したりすることもできます。一度 Java 仮想マシンが作成されたならば、Java コードを実行するために Java 仮想マシンを呼び出す前に、それを再作成する必要はありません。

呼び出し API を使用して Java プログラムを実行する場合、STDOUT および STDERR の宛先は、QIBM\_USE\_DESCRIPTOR\_STDIO と呼ばれる環境変数によって制御されます。この環境変数が Y または I に設定されると (たとえば、QIBM\_USE\_DESCRIPTOR\_STDIO=Y)、Java 仮想マシンは、STDIN (fd 0)、STDOUT (fd 1)、および STDERR (fd 2) のファイル記述子を使用します。この場合、プログラムでは、これらのファイルをこのジョブの最初の 3 つのファイルまたはパイプとしてオープンすることによって、それらのファイル記述子を有効な値に設定しなければなりません。ジョブで最初にオープンされたファイルには 0 の fd が与えられ、2 番目は 1 の fd、3 番目は 2 の fd になります。spawn API によって開始されるジョブの場合、これらの記述子は、ファイル記述子マップを使用して事前に割り当てることができます (spawn API に関する資料を参照)。環境変数 QIBM\_USE\_DESCRIPTOR\_STDIO が設定されないか、またはその他の値に設定されると、STDIN、STDOUT、または STDERR についてファイル記述子は使用されません。その代わりに、STDOUT および STDERR は、現行ジョブによって所有されているスプール・ファイルに送られ、STDIN を使用すると、入出力例外が起ります。

呼び出し API を使用する例については、例: Java 呼び出し API を参照してください。IBM Developer Kit for Java によってサポートされる呼び出し API 関数の詳細については、呼び出し API 関数を参照してください。

### 呼び出し API 関数:

IBM Developer Kit for Java は、以下の呼び出し API 関数がサポートされます。

注: この API を使用する前に、ジョブがマルチスレッド対応であることを確認しなければなりません。マルチスレッド対応ジョブの詳細については、マルチスレッド・アプリケーションを参照してください。

#### • JNI\_GetCreatedJavaVMs

作成済みのすべての Java 仮想マシンに関する情報を戻します。この API は複数の Java 仮想マシン (JVM) の情報を戻すように設計されていますが、1 つのプロセスに存在する JVM は 1 つだけです。したがって、この API は、最大で 1 つの JVM を戻します。

シグニチャー:

```
jint JNI_GetCreatedJavaVMs (JavaVM **vmBuf,
                             jsize bufLen,
                             jsize *nVMs);
```

vmBuf は出力域であり、そのサイズは bufLen (ポインターの数) によって判別されます。それぞれの Java 仮想マシンは、java.h で定義された、関連する JavaVM 構造を持ちます。この API は、作成済みのそれぞれの Java 仮想マシンに関連する JavaVM 構造へのポインターを vmBuf に格納します (vmBuf が 0 でない限り)。JavaVM 構造へのポインターは、作成された対応する Java 仮想マシンの順序で格納されます。nVMs は、現在作成されている仮想マシンの数を戻します。ご使用の iSeries サーバーで、複数の Java 仮想マシンの作成がサポートされるため、1 よりも大きい値が予期されることもあります。この情報と、vmBuf のサイズにより、作成済みの各 Java 仮想マシンの JavaVM 構造へのポインターが戻されているかどうか判別されます。

#### • JNI\_CreateJavaVM

Java 仮想マシンを作成し、後でそれをアプリケーション内で使用することを可能にします。

シグニチャー:

```
jint JNI_CreateJavaVM(JavaVM **p_vm,  
                      void **p_env,  
                      void *vm_args);
```

p\_vm は、新たに作成された Java 仮想マシンを指す JavaVM ポインターのアドレスです。他のいくつかの JNI 呼び出し API では、p\_vm を使用して Java 仮想マシンを識別します。p\_env は、新たに作成された Java 仮想マシンへの JNI 環境ポインターのアドレスです。このポインターは、JNI 関数を開始する、関数のテーブルを指します。vm\_args は、Java 仮想マシンの初期設定パラメーターを含む構造です。

「Java プログラムの実行 (RUNJAVA)」コマンドまたは JAVA コマンドを開始する場合に、同等のコマンド・パラメーターがあるプロパティを指定すると、コマンド・パラメーターが優先され、プロパティは無視されます。たとえば、次のコマンドでは、os400.optimization パラメーターは無視されます。

```
JAVA CLASS(Hello) PROP((os400.optimization 0))
```

JNI\_CreateJavaVM API によってサポートされる OS/400 固有のプロパティのリストについては、Java システム・プロパティを参照してください。

注: iSeries サーバー上の Java は、単一のジョブまたはプロセス内で 1 つの Java 仮想マシン (JVM) の作成しかサポートしません。詳しくは、複数の Java 仮想マシンのサポートを参照してください。

#### • DestroyJavaVM

Java 仮想マシンを破棄します。

シグニチャー:

```
jint DestroyJavaVM(JavaVM *vm)
```

Java 仮想マシンの作成時には、vm が JavaVM ポインターとして戻されます。

#### • AttachCurrentThread

Java 仮想マシンにスレッドを付加して、それが Java 仮想マシン・サービスを使用できるようにします。

シグニチャー:

```
jint AttachCurrentThread(JavaVM *vm,  
                        void **p_env,  
                        void *thr_args);
```

JavaVM ポインター `vm` は、スレッドが付加される Java 仮想マシンを識別します。 `p_env` は、現行スレッドの JNI インターフェース・ポインターが置かれる場所へのポインターです。 `thr_args` には、VM 固有のスレッド付加引数が含まれます。

#### • **DetachCurrentThread**

シグニチャー:

```
jint DetachCurrentThread(JavaVM *vm);
```

`vm` は、スレッドが切り離される Java 仮想マシンを識別します。

呼び出し API 関数の詳細については、Java Native Interface Specification by Sun Microsystems, Inc.、または The Source for Java Technology [java.sun.com](http://java.sun.com) を参照してください。

#### 複数の Java 仮想マシンのサポート:

iSeries サーバー上の Java は、単一のジョブまたはプロセス内での複数の Java 仮想マシン (JVM) の作成をサポートしなくなりました。この制約事項の影響を受けるのは、Java Native Interface Invocation (JNI) API を使用して JVM を作成するユーザーのみです。サポートにおけるこの変更は、Java コマンドを使用して Java プログラムを実行する方法には影響しません。

1 つのジョブで `JNI_CreateJavaVM()` を正常に複数回呼び出すことはできず、`JNI_GetCreatedJavaVMs()` は結果のリストに複数の JVM を戻すことができません。

単一のジョブまたはプロセス内での単一の JVM のみの作成のサポートは、Sun Microsystems, Inc. の Java の参照インプリメンテーションの標準に従ったものです。

#### 例: Java 呼び出し API:

この例は、標準の呼び出し API パラダイムに従っています。

これは以下を実行します。

- `JNI_CreateJavaVM` を使用して Java 仮想マシンを作成する。
- Java 仮想マシンを使用して、実行したいクラス・ファイルを検索する。
- クラスの `main` メソッドの `methodID` を検索する。
- クラスの `main` メソッドを呼び出す。
- 例外が発生した場合に、エラーを報告する。

プログラムを作成する際は、`QJVAJNI` または `QJVAJNI64` サービス・プログラムが、`JNI_CreateJavaVM` 呼び出し API 機能を提供します。 `JNI_CreateJavaVM` は Java 仮想マシンを作成します。

注: `QJVAJNI64` は、`teraspace/LLP64` ネイティブ・メソッドと呼び出し API のサポートのための新しいサービス・プログラムです。

これらのサービス・プログラムは、システム・バインディング・ディレクトリーにあり、制御言語 (CL) 作成コマンドで明示的に示す必要はありません。たとえば、前述のサービス・プログラムを「プログラム作成 (CRTPGM)」コマンドや「サービス・プログラムの作成 (CRTSRVPGM)」コマンドを使用する際に明示的に示すことはしません。

このプログラムを実行する方法の 1 つは、以下の制御言語 (CL) コマンドを使用することです。

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```



Java 仮想マシンを作成するジョブは、マルチスレッド対応でなければなりません。主プログラムからの出力と、プログラムからのすべての出力は、最終的に QPRINT スプール・ファイルに送られます。「投入されたジョブの処理 (WRKSBMJOB)」制御言語 (CL) コマンドを使用し、「ジョブの投入 (SBMJOB)」CL コマンドで開始したジョブを表示すると、これらのスプール・ファイルを見ることができます。

## 例: Java 呼び出し API を使用する

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
#define OS400_JVM_12
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Specify the pragma that causes all literal strings in the
 * source code to be stored in ASCII (which, for the strings
 * used, is equivalent to UTF-8)
 */

#pragma convert(819)

/* Procedure: Oops
 *
 * Description: Helper routine that is called when a JNI function
 * returns a zero value, indicating a serious error.
 * This routine reports the exception to stderr and
 * ends the JVM abruptly with a call to FatalError.
 *
 * Parameters: env -- JNIEnv* to use for JNI calls
 * msg -- char* pointing to error description in UTF-8
 *
 * Note: Control does not return after the call to FatalError
 * and it does not return from this procedure.
 */

void Oops(JNIEnv* env, char *msg) {
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* This is the program's "main" routine. */
int main (int argc, char *argv[])
{
    JavaVMInitArgs initArgs; /* Virtual Machine (VM) initialization structure, passed by
 * reference to JNI_CreateJavaVM(). See jni.h for details
 */
    JVM* myJVM; /* JVM pointer set by call to JNI_CreateJavaVM */
    JNIEnv* myEnv; /* JNIEnv pointer set by call to JNI_CreateJavaVM */
    char* myClasspath; /* Changeable classpath 'string' */
    jclass myClass; /* The class to call, 'NativeHello'. */
    jmethodID mainID; /* The method ID of its 'main' routine. */
    jclass stringClass; /* Needed to create the String[] arg for main */
    jobjectArray args; /* The String[] itself */
    JavaVMOption options[1]; /* Options array -- use options to set classpath */
    int fd0, fd1, fd2; /* file descriptors for IO */

    /* Open the file descriptors so that IO works. */
    fd0 = open("/dev/null1", O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IWOTH);
    fd1 = open("/dev/null2", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);
```

```

fd2 = open("/dev/null13", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);

/* Set the version field of the initialization arguments for J2SDK v1.3. */
initArgs.version = 0x00010002;
/* To use J2SDK v1.4, set initArgs.version = 0x00010004; */
/* To use J2SDK v1.5, set initArgs.version = 0x00010005; */

/* Now, you want to specify the directory for the class to run in the classpath.
 * with Java2, classpath is passed in as an option.
 * Note: You must specify the directory name in UTF-8 format. So, you wrap
 * blocks of code in #pragma convert statements.
 */
options[0].optionString="-Djava.class.path=/CrtJvmExample";
/*To use J2SDK v1.4 or v1.5, replace the '1.3' with '1.4' or '1.5'.
options[1].optionString="-Djava.version=1.3" */

initArgs.options=options; /* Pass in the classpath that has been set up. */
initArgs.nOptions = 2; /* Pass in classpath and version options */

/* Create the JVM -- a nonzero return code indicates there was
 * an error. Drop back into EBCDIC and write a message to stderr
 * before exiting the program.
 */
if (JNI_CreateJavaVM("myJVM, (void **)myEnv, (void *)initArgs)) {
#pragma convert(0)
    fprintf(stderr, "Failed to create the JVM\n");
#pragma convert(819)
    exit(1);
}

/* Use the newly created JVM to find the example class,
 * called 'NativeHello'.
 */
myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
if (! myClass) {
    Oops(myEnv, "Failed to find class 'NativeHello'");
}

/* Now, get the method identifier for the 'main' entry point
 * of the class.
 * Note: The signature of 'main' is always the same for any
 * class called by the following java command:
 * "main", "([Ljava/lang/String;)V"
 */
mainID = (*myEnv)->GetStaticMethodID(myEnv,myClass,"main",
                                     "([Ljava/lang/String;)V");
if (! mainID) {
    Oops(myEnv, "Failed to find jmethodID of 'main'");
}

/* Get the jclass for String to create the array
 * of String to pass to 'main'.
 */
stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
if (! stringClass) {
    Oops(myEnv, "Failed to find java/lang/String");
}

/* Now, you need to create an empty array of strings,
 * since main requires such an array as a parameter.
 */
args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
if (! args) {
    Oops(myEnv, "Failed to create args array");
}

```



```

/* Now, you have the methodID of main and the class, so you can
 * call the main method.
 */
(*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);

/* Check for errors. */
if ((*myEnv)->ExceptionOccurred(myEnv)) {
    (*myEnv)->ExceptionDescribe(myEnv);
}

/* Finally, destroy the JavaVM that you created. */
(*myJVM)->DestroyJavaVM(myJVM);

/* All done. */
return 0;
}

```

詳細については、Java 呼び出し API を参照してください。

## Java ネイティブ・メソッドおよびスレッドに関する考慮事項

ネイティブ・メソッドは、Java では使用できない関数を利用する場合に使用できます。ネイティブ・メソッド付きの Java を使いこなすには、以下のような概念を理解する必要があります。

- Java または付加されたネイティブ・スレッドで作成されたかどうかにかかわらず、Java スレッドでは、浮動小数点例外がすべて使用不可能になっています。スレッドが浮動小数点例外を再度使用可能にするネイティブ・メソッドを実行する場合には、Java がそのメソッドを 2 度目にオフにすることはありません。ユーザー・アプリケーションが戻されて Java コードを実行する前にそれらのメソッドを使用不可にしなければ、浮動小数点例外が起きる際に Java コードは正常に動作しないかもしれません。ネイティブ・スレッドが Java 仮想マシンから切り離される場合、そのスレッドの浮動小数点例外マスクは、付加されていた時の値に戻されます。
- ネイティブ・スレッドが Java 仮想マシンに付加される際には、必要に応じて Java 仮想マシンはスレッドの優先順位を変更して、Java が定義する 1 から 10 のスレッド優先順位体系に準拠します。スレッドが切り離されると、優先順位が復元されます。スレッドが付加されると、スレッドはネイティブ・メソッド・インターフェースを使用して (たとえば、POSIX API) スレッド優先順位を変更できます。Java は、Java 仮想マシンへと移行する際には、スレッド優先順位を変更しません。
- Java ネイティブ・インターフェース (JNI) の呼び出し API 構成要素は、ユーザーが Java 仮想マシンをアプリケーション内に組み込むことを許可します。アプリケーションによって Java 仮想マシンが作成され、Java 仮想マシンが異常終了する場合、Java 仮想マシンが終了した際に初期スレッドが Java 仮想マシンに付加されたならば、MCH74A5 "Java Virtual Machine Terminated" iSeries 例外がプロセスの初期スレッドにシグナルされます。Java 仮想マシンは、以下のいずれかの理由で異常終了することがあります。
  - ユーザーが `java.lang.System.exit()` メソッドを呼び出す。
  - Java 仮想マシンが必要なスレッドが終了する。
  - Java 仮想マシン内で内部エラーが生じる。

この動作は、他のほとんどの Java プラットフォームとは異なります。他のほとんどのプラットフォームでは、Java 仮想マシンが終了すると、Java 仮想マシンを自動的に作成するプロセスは異常終了します。アプリケーションによって、シグナルが出された MCH74A5 例外のモニターおよび処理が行われると、そのプロセスは実行を継続するかもしれません。そうではない場合には、例外が処理されない時にプロセスは終了します。iSeries サーバー固有の MCH74A5 例外を扱うコードを追加すると、他のプラットフォームへのアプリケーションの可搬性を低下させることがあります。

ネイティブ・メソッドは常にマルチスレッド・プロセスで実行されるので、ネイティブ・メソッドのコードはスレッド・セーフなものでなければなりません。このため、ネイティブ・メソッドで使用される言語および関数には次の制約があります。

- ILE CL はネイティブ・メソッドには使用しないでください。なぜなら、この言語はスレッド・セーフなものではないからです。スレッド・セーフな CL コマンドを実行するには、C 言語の `system()` 関数か `java.lang.Runtime.exec()` メソッドを使用できます。
  - C または C++ ネイティブ・メソッドでスレッド・セーフな CL コマンドを実行するには、C 言語の `system()` 関数を使用します。
  - スレッド・セーフな CL コマンドを Java から直接実行するには、`java.lang.Runtime.exec()` メソッドを使用します。
- ILE C、ILE C++、ILE COBOL および ILE RPG を使用してネイティブ・メソッドを作成できますが、ネイティブ・メソッド内から呼び出す関数はすべてスレッド・セーフでなければなりません。

注: ネイティブ・メソッドを作成するためのコンパイル時サポートは、現時点では C、C++、および RPG 言語のみでしか提供されていません。他の言語でネイティブ・メソッドを作成することは可能ですが、ずっと複雑なものになります。

注意: 標準 C、C++、COBOL、または RPG 関数のすべてがスレッド・セーフなものであるとは限りません。

- C および C++ の `exit()` と `abort()` 関数は、ネイティブ・メソッド内で使用されるべきではありません。これらの関数は、Java 仮想マシンを実行するプロセス全体を停止させます。これには、プロセス内のすべてのスレッドが含まれています (Java のものであるかどうか関係なく)。

注: 参照されている `exit()` 関数は C および C++ 関数であり、`java.lang.Runtime.exit()` メソッドとは異なります。

iSeries サーバーでのスレッドの詳細については、マルチスレッド・アプリケーションを参照してください。

## ネイティブ・メソッドおよび Java ネイティブ・インターフェース

ネイティブ・メソッドとは、Java 以外の言語で開始される Java メソッドです。ネイティブ・メソッドは、Java では直接使用できないシステム固有の機能や API にアクセスできます。

ネイティブ・メソッドにはシステム固有のコードがあるので、ネイティブ・メソッドを使用するとアプリケーションのポータビリティが制限されます。ネイティブ・メソッドは、新しいネイティブ・コード・ステートメントまたは既存のネイティブ・コードを呼び出すネイティブ・コード・ステートメントのいずれかです。

ネイティブ・メソッドが必要な場合、ネイティブ・メソッドとそれを実行する Java 仮想マシンとの間の相互操作性が必要になります。Java ネイティブ・インターフェース (JNI) を使用すると、どのプラットフォームにも依存せずに、この相互運用性を容易に実現できます。

JNI は、一連のインターフェースであり、JNI を使うとネイティブ・メソッドと Java 仮想マシンとのさまざまな相互操作性を実現できます。たとえば、JNI には、新しいオブジェクトを作成してメソッドを呼び出すインターフェース、フィールドの取得と設定を行うインターフェース、例外を処理するインターフェース、ストリングおよび配列の操作を行うインターフェースなどが含まれています。

JNI の詳細については、Java Native Interface by Sun Microsystems, Inc.、または The Source for Java Technology [java.sun.com](http://java.sun.com) を参照してください。

## ネイティブ・メソッドのストリング

ほとんどの Java ネイティブ・インターフェース (JNI) 関数は、パラメーターとして C 言語形式のストリングを受け入れます。たとえば、JNI 関数の `FindClass()` は、クラス・ファイルの完全修飾名を指定するストリング・パラメーターを受け入れます。クラス・ファイルが検出されると、このクラス・ファイルは `FindClass` によってロードされ、その参照が `FindClass` の呼び出し元に戻ります。

JNI 関数のストリング・パラメーターは、UTF-8 でエンコードする必要があります。UTF-8 の詳細については、JNI 仕様に記載されていますが、通常は、7 ビット情報交換用米国標準コード (ASCII) 文字が UTF-8 表示と等価であることを確認するだけで十分です。7 ビット ASCII 文字は実際には 8 ビット文字ですが、先頭ビットは常に 0 になっています。ほとんどの C ストリングはすでに UTF-8 形式になっています。

iSeries サーバー・システムの C コンパイラーはデフォルトで拡張 2 進化 10 進コード (EBCDIC) で作動するので、JNI 関数に UTF-8 形式でストリングを渡すことができます。これには、リテラル・ストリングと動的ストリングという 2 つの方法があります。リテラル・ストリングとは、ソース・コードのコンパイル時に値が分かっているストリングです。動的ストリングとは、コンパイル時には値が不明ですが、実行時に実際の計算が行われるストリングです。

### ネイティブ・メソッド内のリテラル・ストリング:

リテラル・ストリングが 7 ビットの情報交換用米国標準コード (ASCII) 表記の文字で構成されている場合、ストリングを UTF-8 でエンコードすることは比較的簡単です。

大部分のストリングは ASCII で表現することができ、そのようなストリングは、コンパイラーの現行のコード・ページを変更する「プラグマ」ステートメントで囲むことができます。そうすると、コンパイラーは、JNI によって必要とされる UTF-8 形式でストリングを内部的に格納します。ストリングが ASCII で表現できない場合には、元の拡張 2 進化 10 進コード (EBCDIC) ストリングを動的ストリングとして扱い、それを JNI に渡す前に `iconv()` によって処理すると簡単です。動的ストリングの詳細については、動的ストリングを参照してください。

たとえば、`java/lang/String` という名前前のクラスを検索する場合、コードは次のようになります。

```
#pragma convert(819)
myClass = (*env)->FindClass(env,"java/lang/String");
#pragma convert(0)
```

番号 819 が指定された最初のプラグマは、コンパイラーに、後続のすべての二重引用符付きストリング (リテラル・ストリング) を ASCII で格納するよう指示します。番号 0 が指定された 2 番目のプラグマは、コンパイラーに、二重引用符付きストリングについてのコンパイラーのデフォルト・コード・ページ (通常、EBCDIC コード・ページ 37) に戻るよう指示します。このように、呼び出しをプラグマで囲むことによって、ストリング・パラメーターが UTF-8 でエンコードされるという JNI の要件を満たします。

**注意:** テキストの置換は慎重に行ってください。たとえば、コードが次のようになっている場合、

```
#pragma convert(819)
#define MyString "java/lang/String"
#pragma convert(0)
myClass = (*env)->FindClass(env,MyString);
```

結果のストリングは EBCDIC になります。これは、コンパイル時に `MyString` の値が `FindClass` 呼び出しの中で置換されるためです。この置換の時点で、番号 819 のプラグマは有効ではありません。したがって、リテラル・ストリングは ASCII で格納されません。

### 動的ストリングを EBCDIC、Unicode、および UTF-8 に変換する:

実行時に計算される文字列変数を処理するために、文字列を拡張 2 進化 10 進コード (EBCDIC)、Unicode および UTF-8 へ変換したり、この逆の変換を行う必要がある場合があります。

コード・ページ変換関数を提供するシステム API は、`iconv()` です。`iconv()` を使用するには、次の手順に従ってください。

1. `QtqIconvOpen()` で、変換記述子を作成する。
2. この記述子を使って文字列に変換するために、`iconv()` を呼び出す。
3. `iconv_close` を使用して、記述子をクローズする。

ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用する例 3は、ルーチン内で `iconv` 変換記述子を作成、使用、および削除します。この方式は、`iconv_t` 記述子をマルチスレッド式に使用することで問題を回避しますが、パフォーマンス依存コードの場合は、静的ストレージ内に変換記述子を作成し、相互除外 (mutex) やその他の同期機能を使用することにより、複数のアクセスを制限してください。

## Java 用の IBM i5/OS PASE ネイティブ・メソッド

iSeries Java 仮想マシン (JVM) は、i5/OS PASE 環境で実行するネイティブ・メソッドの使用をサポートしています。V5R2 より前は、ネイティブ iSeries JVM は ILE ネイティブ・メソッドのみを使用していました。

i5/OS PASE ネイティブ・メソッドのサポートには以下が含まれます。

- i5/OS PASE ネイティブ・メソッドからのネイティブ iSeries Java ネイティブ・インターフェース (JNI) の全面使用
- ネイティブ iSeries JVM から i5/OS PASE ネイティブ・メソッドを呼び出す機能

この新しいサポートにより、AIX で実行する Java アプリケーションを iSeries サーバーに容易にポータリングすることができます。クラス・ファイルと AIX ネイティブ・メソッド・ライブラリーを iSeries 上の統合ファイル・システムにコピーし、制御言語 (CL)、Qshell、または i5/OS PASE のいずれかの端末セッション・コマンド・プロンプトからそれらを実行できます。

### 関連情報

#### i5/OS PASE

この資料は、読者がすでに i5/OS PASE に精通していることを前提としています。PASE にまだ精通していない場合には、このトピックから Java における IBM i5/OS PASE ネイティブ・メソッドの使用についてさらに学習してください。

## Java i5/OS PASE 環境変数

Java 仮想マシン (JVM) は以下の変数を使用して i5/OS PASE 環境を始動します。Java 用の IBM i5/OS PASE ネイティブ・メソッドの例を実行するには、`QIBM_JAVA_PASE_STARTUP` 変数を設定する必要があります。

この例の環境変数を設定する方法については、以下のトピックを参照してください。

### IBM i5/OS PASE 例の環境変数

#### QIBM\_JAVA\_PASE\_STARTUP

以下の状態の両方が発生したときには、この環境変数を設定する必要があります。

- i5/OS PASE ネイティブ・メソッドを使用している
- iSeries コマンド・プロンプトまたは Qshell コマンド・プロンプトから Java を開始している



JVM はこの環境変数を使用して PASE 環境を始動します。変数の値は i5/OS PASE 始動プログラムを示します。iSeries サーバーには以下の 2 つの i5/OS PASE 始動プログラムが含まれています。

- /usr/lib/start32: 32 ビット i5/OS PASE 環境を始動する
- /usr/lib/start64: 64 ビット i5/OS PASE 環境を始動する

i5/OS PASE 環境で使用されるすべての共用ライブラリー・オブジェクトのビット・フォーマットは、i5/OS PASE 環境のビット・フォーマットと一致している必要があります。

i5/OS PASE 端末セッションから Java を始動するときには、この変数は使用できません。i5/OS PASE 端末セッションは常に 32-bit i5/OS PASE 環境を使用します。i5/OS PASE 端末セッションから始動された JVM はいずれも、その端末セッションと同じタイプの PASE 環境を使用します。

#### **QIBM\_JAVA\_PASE\_CHILD\_STARTUP**

2 次 JVM の i5/OS PASE 環境が、1 次 JVM 用の i5/OS PASE 環境と異なる必要があるとき、このオプションの環境変数を設定します。Java 中の Runtime.exec() を呼び出すと、2 次 (または子) JVM が開始します。

詳しくは、QIBM\_JAVA\_PASE\_CHILD\_STARTUP の使用を参照してください。

#### **QIBM\_JAVA\_PASE\_ALLOW\_PREV**

i5/OS PASE 環境が存在している場合で、現行の OS/400 PASE 環境を使用したいとき、このオプションの環境変数を設定してください。i5/OS PASE 環境がすでに存在しているかどうかを判別するのは、困難な場合があります。QIBM\_JAVA\_PASE\_ALLOW\_PREV および QIBM\_JAVA\_PASE\_STARTUP を組み合わせて使用すれば、JVM は既存の i5/OS PASE 環境を使用するか、あるいは新しく i5/OS PASE 環境を開始することができます。

詳しくは、QIBM\_JAVA\_PASE\_ALLOW\_PREV の使用を参照してください。

#### **例: IBM i5/OS PASE 例の環境変数:**

IBM i5/OS PASE native methods for Java 例を使用するには、環境変数を設定する必要があります。

#### **PASE\_LIBPATH**

iSeries サーバーはこの i5/OS PASE 環境変数を使用して、i5/OS PASE ネイティブ・メソッド・ライブラリーの位置を識別します。単一のディレクトリーか複数のディレクトリーのパスを設定できます。複数のディレクトリーを設定する場合は、コロン (;) を使用してエントリーを区切ります。サーバーは LIBPATH 環境変数を使用することもできます。

Java、ネイティブ・メソッド・ライブラリー、および PASE\_LIBPATH のこの例での使用法について詳しくは、Java、i5/OS PASE、およびネイティブ・メソッド・ライブラリーを使用するを参照してください。

#### **PASE\_THREAD\_ATTACH**

この i5/OS PASE 環境変数を Y に設定すると、i5/OS PASE によって開始されなかった ILE スレッドが、i5/OS PASE が i5/OS PASE プロシーチャーを呼び出すときに自動的に i5/OS PASE に接続されます。

i5/OS PASE 環境変数について詳しくは、i5/OS PASE 環境変数の処理の中の該当する項目を参照してください。

#### **QIBM\_JAVA\_PASE\_STARTUP**

JVM はこの環境変数を使用して i5/OS PASE 環境を始動します。変数の値は i5/OS PASE 始動プログラムを示します。

詳細については、Java i5/OS PASE 環境変数を参照してください。

### QIBM\_JAVA\_PASE\_CHILD\_STARTUP を使用する:

QIBM\_JAVA\_PASE\_CHILD\_STARTUP 環境変数は、任意の 2 次 JVM の i5/OS PASE 始動プログラムを示します。

QIBM\_JAVA\_PASE\_CHILD\_STARTUP は、以下のすべての条件が当てはまる場合に使用します。

- 実行する Java アプリケーションが、Runtime.exec() への Java 呼び出しを通して Java 仮想マシン (JVM) を作成する。
- 1 次と 2 次の両方の JVM が i5/OS PASE ネイティブ・メソッドを使用する。
- 2 次 JVM の i5/OS PASE 環境は、1 次 JVM の i5/OS PASE 環境と異ならなければならない。

上にリストされているすべての条件が当てはまる場合は、以下のアクションを実行します。

- QIBM\_JAVA\_PASE\_CHILD\_STARTUP 環境変数を、2 次 JVM の i5/OS PASE 始動プログラムに設定します。
- iSeries コマンド・プロンプトまたは Qshell コマンド・プロンプトから 1 次 JVM を始動するときに、QIBM\_JAVA\_PASE\_STARTUP 環境変数を 1 次 JVM の i5/OS PASE 始動プログラムに設定します。

注: i5/OS PASE 端末セッションから 1 次 JVM を始動するときは、QIBM\_JAVA\_PASE\_STARTUP を設定しないでください。

2 次 JVM のプロセスは QIBM\_JAVA\_PASE\_CHILD\_STARTUP 環境変数を継承します。さらに、i5/OS は 2 次 JVM プロセスの QIBM\_JAVA\_PASE\_STARTUP 環境変数を、親プロセスの QIBM\_JAVA\_PASE\_CHILD\_STARTUP 環境変数の値に設定します。

以下の表は、コマンド環境と QIBM\_JAVA\_PASE\_STARTUP および QIBM\_JAVA\_PASE\_CHILD\_STARTUP の定義のさまざまな組み合わせに対する、結果の i5/OS PASE 環境 (存在する場合) を示しています。

表 1. QIBM\_JAVA\_PASE\_STARTUP および QIBM\_JAVA\_PASE\_CHILD\_STARTUP の結果の PASE 環境

| コマンド環境             | 始動環境                   |                       | 結果動作                      |                       |
|--------------------|------------------------|-----------------------|---------------------------|-----------------------|
|                    | QIBM_JAVA_PASE_STARTUP | 1 次 JVM i5/OS PASE 始動 | 1 次 JVM i5/OS PASE 始動     | 2 次 JVM i5/OS PASE 始動 |
| CL または QSH         | 定義されている startX         | 定義されている startY        | startX を使用する              | startY を使用する          |
| CL または QSH         | 定義されている startX         | 定義されていない              | startX を使用する              | startX を使用する          |
| CL または QSH         | 定義されていない               | 定義されている startY        | i5/OS PASE 環境は使用しない       | startY を使用する          |
| CL または QSH         | 定義されていない               | 定義されていない              | i5/OS PASE 環境は使用しない       | i5/OS PASE 環境は使用しない   |
| i5/OS PASE 端末セッション | 定義されている startX         | 定義されている startY        | 許可されていない*                 | 許可されていない*             |
| i5/OS PASE 端末セッション | 定義されている startX         | 定義されていない              | 許可されていない*                 | 許可されていない*             |
| i5/OS PASE 端末セッション | 定義されていない               | 定義されている startY        | i5/OS PASE 端末セッション環境を使用する | startY を使用する          |

表 1. QIBM\_JAVA\_PASE\_STARTUP および QIBM\_JAVA\_PASE\_CHILD\_STARTUP の結果の PASE 環境 (続き)

| 始動環境               |          |          | 結果動作                      |                     |
|--------------------|----------|----------|---------------------------|---------------------|
| i5/OS PASE 端末セッション | 定義されていない | 定義されていない | i5/OS PASE 端末セッション環境を使用する | i5/OS PASE 環境は使用しない |

\* 「許可されていない」の表示がある行は、QIBM\_JAVA\_PASE\_STARTUP 環境変数が i5/OS PASE 端末セッションと競合する可能性がある状況を示しています。競合の可能性があるため、i5/OS PASE 端末セッションからの QIBM\_JAVA\_PASE\_STARTUP の使用は許可されていません。

#### QIBM\_JAVA\_PASE\_ALLOW\_PREV の使用:

i5/OS PASE 環境がすでに存在しているかどうかを判別するのは、困難な場合があります。オプションの環境変数 QIBM\_JAVA\_PASE\_ALLOW\_PREV を QIBM\_JAVA\_PASE\_STARTUP と組み合わせて使用すれば、JVM は現行の i5/OS PASE 環境 (存在する場合) を使用するか、新しい i5/OS PASE 環境を開始するかを決定できるようになります。

これら 2 つの環境変数を組み合わせて使用する場合は、これらを以下の値に設定してください。

- QIBM\_JAVA\_PASE\_STARTUP をデフォルトの始動プログラムに設定する
- QIBM\_JAVA\_PASE\_ALLOW\_PREV を 1 に設定する

たとえば、オプションで i5/OS PASE 環境を開始するアプリケーションは JVM を開始するプログラムを呼び出します。この場合、プログラムは前の設定を使用することによって、現行の i5/OS PASE 環境を使用する (存在する場合) か、あるいは新しく i5/OS PASE 環境を開始することができます。

以下の表は、i5/OS PASE 環境と QIBM\_JAVA\_PASE\_STARTUP および QIBM\_JAVA\_PASE\_ALLOW\_PREV の定義のさまざまな組み合わせから生じる i5/OS PASE 環境を示しています。

表 2. i5/OS PASE 環境と、QIBM\_JAVA\_PASE\_STARTUP および QIBM\_JAVA\_PASE\_ALLOW\_PREV の定義の組み合わせの結果の i5/OS PASE 環境

| 始動環境          |                        |                           | 結果動作                   |
|---------------|------------------------|---------------------------|------------------------|
| i5/OS PASE 環境 | QIBM_JAVA_PASE_STARTUP | QIBM_JAVA_PASE_ALLOW_PREV | JVM i5/OS PASE 始動      |
| なし            | 定義されていない               | 定義されていない *                | i5/OS PASE 環境は使用しない    |
| なし            | 定義されていない               | '1' に定義済み                 | i5/OS PASE 環境は使用しない    |
| なし            | 定義されている startX         | 定義されていない *                | startX を使用する           |
| なし            | 定義されている startX         | '1' に定義済み                 | startX を使用する           |
| 開始済み          | 定義されていない               | 定義されていない *                | 既存の i5/OS PASE 環境を使用する |
| 開始済み          | 定義されていない               | '1' に定義済み                 | 既存の i5/OS PASE 環境を使用する |
| 開始済み          | 定義されている startX         | 定義されていない *                | 許可されていない: 始動時の JVM エラー |
| 開始済み          | 定義されている startX         | '1' に定義済み                 | 既存の i5/OS PASE 環境を使用する |



\* 「定義されていない」とは、QIBM\_JAVA\_PASE\_ALLOW\_PREV が組み込まれていないか、あるいはこれに 1 以外の値が指定されていることを意味します。

上の表の最後の 2 行は、QIBM\_JAVA\_PASE\_ALLOW\_PREV の設定が役立つ状況です。i5/OS PASE 環境がすでに存在し、QIBM\_JAVA\_PASE\_STARTUP が設定されている場合、JVM は QIBM\_JAVA\_PASE\_ALLOW\_PREV を検査します。そうでない場合、JVM は QIBM\_JAVA\_PASE\_ALLOW\_PREV を無視します。

QIBM\_JAVA\_PASE\_ALLOW\_PREV および QIBM\_JAVA\_PASE\_CHILD\_STARTUP 環境変数は互いに独立しています。

## Java i5/OS PASE エラー・コード

i5/OS PASE ネイティブ・メソッドのトラブルシューティングに役立つため、このトピックでは、i5/OS ジョブ・ログ・メッセージおよび Java ランタイム例外によって示されたエラー状態について説明しています。以下のリストは、Java 用の i5/OS PASE ネイティブ・メソッドを使用するとき、始動時や実行時に発生する可能性のあるエラーを示しています。

### 始動時のエラー

始動時のエラーについては、適切なジョブ・ログ内のメッセージを検査してください。

### 実行時のエラー

JVM の Qshell 出力には、始動時のエラーに加えて、以下の PASEInternalError 例外や PASEExit Java 例外が表示されることがあります。

- PASEInternalError - 内部システム・エラーを示します。ライセンス内部コード・ログ・エントリーを調べてください。

PASEInternalError エラー・コードについて詳しくは、Qp2CallPASE を参照してください。

- PASEExit - i5/OS PASE アプリケーションが exit() 関数を呼び出したか、あるいは i5/OS PASE 環境が異常終了した。追加情報について、ジョブ・ログおよびライセンス内部コード・ログを検査してください。

## ネイティブ・メソッド・ライブラリーの管理

ネイティブ・メソッド・ライブラリーを使用する場合、とりわけ複数のバージョンのネイティブ・メソッド・ライブラリーを iSeries サーバーで管理するときには、Java ライブラリーの命名規則とライブラリー検索アルゴリズムの両方を理解している必要があります。

i5/OS は、Java 仮想マシン (JVM) がロードするライブラリーの名前と一致する最初のネイティブ・メソッドを使用します。i5/OS が正しいネイティブ・メソッドを見つけるようにするためには、ライブラリー名の競合や、JVM がどのネイティブ・メソッド・ライブラリーを使用するのかに関する混乱を避ける必要があります。

## i5/OS PASE および AIX Java ライブラリーの命名規則

Java コードが Sample という名前のライブラリーをロードする場合、対応する実行可能ファイルには libSample.a または libSample.so のいずれかの名前を付ける必要があります。

## Java ライブラリーの検索順序

JVM 用 i5/OS PASE ネイティブ・メソッドを使用可能にすると、サーバーは 3 つの異なるリストを (以下の順序で) 使用して、単一のネイティブ・メソッド・ライブラリー検索パスを作成します。

1. i5/OS ライブラリー・リスト
2. LIBPATH 環境変数
3. PASE\_LIBPATH 環境変数

検索を実行するため、i5/OS はライブラリー・リストを統合ファイル・システムでの形式に変換します。QSYS ファイル・システム・オブジェクトには、統合ファイル・システムでの等価の名前がありますが、一部の統合ファイル・システム・オブジェクトには、それと等価な QSYS ファイル・システムでの名前が存在しません。ライブラリー・ローダーは QSYS ファイル・システムと統合ファイル・システムの両方でオブジェクトを検索するため、i5/OS は統合ファイル・システムでの形式を使用してネイティブ・メソッド・ライブラリーを検索します。

以下の表は、i5/OS がどのようにライブラリー・リスト内のエントリーを統合ファイル・システムでの形式に変換するのかわを示しています。

| ライブラリー・リスト・エントリー | 統合ファイルシステムでの形式      |
|------------------|---------------------|
| QSYS             | /qsys.lib           |
| QSYS2            | /qsys.lib/qsys2.lib |
| QGPL             | /qsys.lib/qgpl.lib  |
| QTEMP            | /qsys.lib/qtemp.lib |

### 例: Sample2 ライブラリーの検索

以下の例では、LIBPATH が /home/user1/lib32:/samples/lib32 に設定され、PASE\_LIBPATH が /QOpenSys/samples/lib に設定されます。

以下の表は、上から下に向かって読むと、フル検索パスを表します。

| ソース          | 統合ファイル・システムのディレクトリー   |
|--------------|---|
| ライブラリー・リスト   | /qsys.lib<br>/qsys.lib/qsys2.lib<br>/qsys.lib/qgpl.lib<br>/qsys.lib/qtemp.lib |
| LIBPATH      | /home/user1/lib32<br>/samples/lib32   |
| PASE_LIBPATH | /QOpenSys/samples/lib   |

注: 大文字と小文字は /QOpenSys パスでのみ区別されます。

ライブラリー Sample2 を検索するため、Java ライブラリー・ローダーは以下の順序でファイル候補を検索します。

1. /qsys.lib/sample2.srvpgm
2. /qsys.lib/libSample2.a
3. /qsys.lib/libSample2.so

4. /qsys.lib/qsys2.lib/sample2.srvpgm
5. /qsys.lib/qsys2.lib/libSample2.a
6. /qsys.lib/qsys2.lib/libSample2.so
7. /qsys.lib/qgpl.lib/sample2.srvpgm
8. /qsys.lib/qgpl.lib/libSample2.a
9. /qsys.lib/qgpl.lib/libSample2.so
10. /qsys.lib/qtemp.lib/sample2.srvpgm
11. /qsys.lib/qtemp.lib/libSample2.a
12. /qsys.lib/qtemp.lib/libSample2.so
13. /home/user1/lib32/sample2.srvpgm
14. /home/user1/lib32/libSample2.a
15. /home/user1/lib32/libSample2.so
16. /samples/lib32/sample2.srvpgm
17. /samples/lib32/libSample2.a
18. /samples/lib32/libSample2.so
19. /QOpenSys/samples/lib/SAMPLE2.srvpgm
20. /QOpenSys/samples/lib/libSample2.a
21. /QOpenSys/samples/lib/libSample2.so

i5/OS は、実際に存在するリスト内の最初の候補を、ネイティブ・メソッド・ライブラリーとして JVM にロードします。検索で 'qsys.lib/libSample2.a' および 'qsys.lib/libSample2.so' のような候補が発生しても、/qsys.lib ディレクトリー内に統合ファイル・システム・ファイルやシンボリック・リンクを作成することはできません。したがって、i5/OS がこれらの候補ファイルを検査しても、/qsys.lib で始まる統合ファイル・システム・ディレクトリーの中でこれらを見つけることはありません。

ただし、他の統合ファイル・システム・ディレクトリーから QSYS ファイル・システムの中の i5/OS オブジェクトへの、任意のシンボリック・リンクを作成することはできます。このため、有効なファイル候補には、/home/user1/lib32/sample2.srvpgm などのファイルも含まれます。

### 例: Java 用の IBM i5/OS PASE ネイティブ・メソッド

この Java 用の IBM i5/OS PASE ネイティブ・メソッドの例では、Java ネイティブ・インターフェース (JNI) を使用して Java コードにコールバックを行うネイティブ C メソッドのインスタンスを呼び出します。この例では、Java コードから直接ストリングにアクセスするのではなく、JNI を通して Java にコールバックを行ってストリング値を取得するネイティブ・メソッドを呼び出します。

このソース・ファイル例の HTML 版を表示するには、以下のリンクを使用してください。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

- PaseExample1.java
- PaseExample1.c

この i5/OS PASE ネイティブ・メソッド例を実行するには、まず以下のタスクを完了する必要があります。

1. 使用している AIX ワークステーションへのソース・コード例のダウンロード

2. ソース・コード例の準備をする
3. iSeries サーバーの準備

## i5/OS PASE native method for Java 例の実行

上記のタスクを完了したら、この例を実行することができます。このプログラム例を実行するには、以下のコマンドのいずれかを使用します。

- iSeries サーバーのコマンド・プロンプトから:

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- Qshell コマンド・プロンプトまたは i5/OS PASE 端末セッションから:

```
cd /home/example  
java PaseExample1
```

## Java 用 Teraspace ストレージ・モデル・ネイティブ・メソッド

iSeries Java 仮想マシン (JVM) は現在、Teraspace ストレージ・モデル・ネイティブ・メソッドの使用をサポートしています。Teraspace ストレージ・モデルは、ILE プログラム用の大規模処理のローカル・アドレス環境を提供します。Teraspace を使用すれば、ソース・コードをほとんどあるいは全く変更しないで、ネイティブ・メソッド・コードを他のオペレーティング・システムから i5/OS に移植することができます。

Teraspace ストレージ・モデルを使用したプログラミングについては、以下の情報を参照してください。

「ILE 概念」の第 4 章

「WebSphere Development Studio ILE C/C++ Programmer's Guide」の第 17 章

Teraspace ストレージ・モデル用に作成された Java ネイティブ・メソッドの概念は、単一レベルのストレージを使用するネイティブ・メソッドの概念によく似ています。JVM は Teraspace ネイティブ・メソッドに、このメソッドが JNI 関数を呼び出すために使用できる Java Native Interface (JNI) 環境へのポインタを渡します。

Teraspace ストレージ・モデル・ネイティブ・メソッド用に、JVM は、Teraspace ストレージ・モデルと 8 バイトのポインタを使用する JNI 関数インプリメンテーションを用意しています。

## Teraspace ネイティブ・メソッドの作成

Teraspace ストレージ・モデル・ネイティブ・メソッドを正常に作成するには、Teraspace モジュール作成コマンドで、以下のオプションを使用する必要があります。

```
TERASPACE(*YES) STGMDL(*TERASPACE) DTAMDLL(*LLP64)
```

Teraspace ストレージ機能を使用するための以下のオプション (\*TSIFC) は、オプションです。

```
TERASPACE(*YES *TSIFC)
```

**注:** Teraspace ストレージ・モデル Java ネイティブ・メソッドを使用するときに DTAMDLL(\*LLP64) を使用しない場合、ネイティブ・メソッドを呼び出すと、実行時例外がスローされます。

## ネイティブ・メソッドを使用する Teraspace サービス・プログラムの作成

Teraspace ストレージ・モデル・サービス・プログラムを作成するためには、「サービス・プログラムの作成 (CRTSRVPGM)」制御言語 (CL) コマンドで以下のオプションを使用します。

```
CRTSRVPGM STGMDL(*TERASPACE)
```

さらに、ACTGRP(\*CALLER) オプションを使用する必要があります。このオプションを使用すると、JVM がすべての Teraspace ストレージ・モデル・ネイティブ・メソッド・サービス・プログラムを同一の Teraspace 活動化グループ内に活動化できるようになります。ネイティブ・メソッドが効率的に例外を処理するためには、このようにして Teraspace 活動化グループを作成することが重要になる場合があります。

プログラムの活動化と活動化グループについては、以下の情報を参照してください。

「ILE 概念」の第 3 章

## Teraspace ネイティブ・メソッドとの Java 呼び出し API の使用

JNI 環境ポインターがサービス・プログラムのストレージ・モデルと合わない場合は、呼び出し API GetEnv 関数を使用してください。呼び出し API GetEnv 関数は、常に正しい JNI 環境ポインターを戻します。詳しくは、以下のページを参照してください。

Java 呼び出し API

JNI の機能拡張 (JNI Enhancements)

JVM は単一レベルと Teraspace ストレージ・モデルのネイティブ・メソッドをサポートしていますが、これらの 2 つのストレージ・モデルは異なる JNI 環境を使用します。2 つのストレージ・モデルは異なる JNI 環境を使用するので、2 つのストレージ・モデルのネイティブ・メソッド間で JNI 環境ポインターをパラメーターとして渡すことはしないでください。

## 統合言語環境®と Java との比較

iSeries サーバー上の Java 環境は、統合言語環境 (ILE) とは異なります。Java は ILE 言語ではないので、iSeries サーバー上でプログラムやサービス・プログラムを作成するために ILE オブジェクト・モジュールとバインドすることはできません。

| ILE  | Java  |
|--|---|
| iSeries サーバー上のライブラリーまたはファイル構造のメンバーにソース・コードを格納する。                       | 統合ファイル・システム内のストリーム・ファイルにソース・コードを格納する。               |
| EBCDIC (拡張 2 進化 10 進コード) 形式のソース・ファイルを原始ステートメント入力キューティリティー (SEU) で編集する。 | 通常は、ASCII 形式のソース・ファイルをワークステーションのエディターで編集する。         |
| ソース・ファイルをコンパイルしてオブジェクト・コード・モジュールを生成し、iSeries サーバー上のライブラリーに格納する。        | ソース・コードをコンパイルしてクラス・ファイルを生成し、統合ファイル・システムに格納する。       |
| オブジェクト・モジュールは、プログラムまたはサービス・プログラム内で静的にバインドされる。                          | クラスは、実行時に必要に応じて動的にロードされる。                           |
| 他の ILE プログラミング言語で記述された関数を直接呼び出すことができる。                                 | Java から他の言語を呼び出すには、Java ネイティブ・インターフェースを使用しなければならない。 |
| ILE 言語は、常に機械命令としてコンパイルされ、実行される。  | Java プログラムは、インタープリター形式で実行することも、コンパイラー形式で実行することもできる。 |

## java.lang.Runtime.exec() を使用する

java.lang.Runtime.exec メソッドを使用して、Java プログラム内からプログラムまたはコマンドを呼び出します。java.lang.Runtime.exec() メソッドを使用すると、1 つ以上の追加のスレッド対応のジョブが作成されます。追加のジョブが、このメソッドに渡されたコマンド・ストリングを処理します。

注: java.lang.Runtime.exec メソッドは別個のジョブでプログラムを実行します。これは C の system() 関数とは異なる点です。C システム機能は、同一のジョブでプログラムを実行します。

発生する実際のプロセスは、以下の項目に依存します。

- java.lang.Runtime.exec() に渡すコマンドの種類
- os400.runtime.exec システム・プロパティの値

### 種々のタイプのコマンドを処理する

以下の表は、java.lang.Runtime.exec() が種々のタイプのコマンドを処理する方法と、os400.runtime.exec システム・プロパティの影響を示しています。

| コマンドのタイプ  | os400.runtime.exec システム・プロパティの値                                   |  |
|-----------|---|--|
|           | EXEC (デフォルト値)   | QSHELL   |
| Java コマンド | JVM を実行する 2 番目のジョブを開始します。JVM は、Java アプリケーションを実行する 3 番目のジョブを開始します。 | シェル・インタープリターである Qshell を実行する 2 番目のジョブを開始します。Qshell は、Java アプリケーション、プログラム、またはコマンドを実行するための 3 番目のジョブを開始します。 |
| プログラム     | 実行可能プログラム (i5/OS プログラムまたは i5/OS PASE プログラム) を実行する 2 番目のジョブを開始します。 |  |
| CL コマンド   | i5/OS プログラムを実行する 2 番目のジョブを開始します。i5/OS は、2 番目のジョブで CL コマンドを実行します。  |  |

注: CL コマンドまたは CL プログラムを呼び出す際は、呼び出すコマンドにパラメーターとして渡す文字がジョブの CCSID に含まれていることを確認してください。

2 番目または 3 番目のジョブの処理は、元のジョブの Java 仮想マシン (JVM) と並行して実行されます。これらのジョブにおける終了またはシャットダウン処理は、元の JVM には影響しません。

### os400.runtime.exec システム・プロパティ

os400.runtime.exec システム・プロパティの値は、EXEC (デフォルト値) か QSHELL に設定できます。os400.runtime.exec の値によって、java.lang.Runtime.exec() が EXEC インターフェースを使用するか、Qshell を使用するかが決まります。

QSHELL ではなく EXEC の値を使用すると、以下の利点があります。

- java.lang.Runtime.exec() を呼び出す Java プログラムの可搬性が向上します。

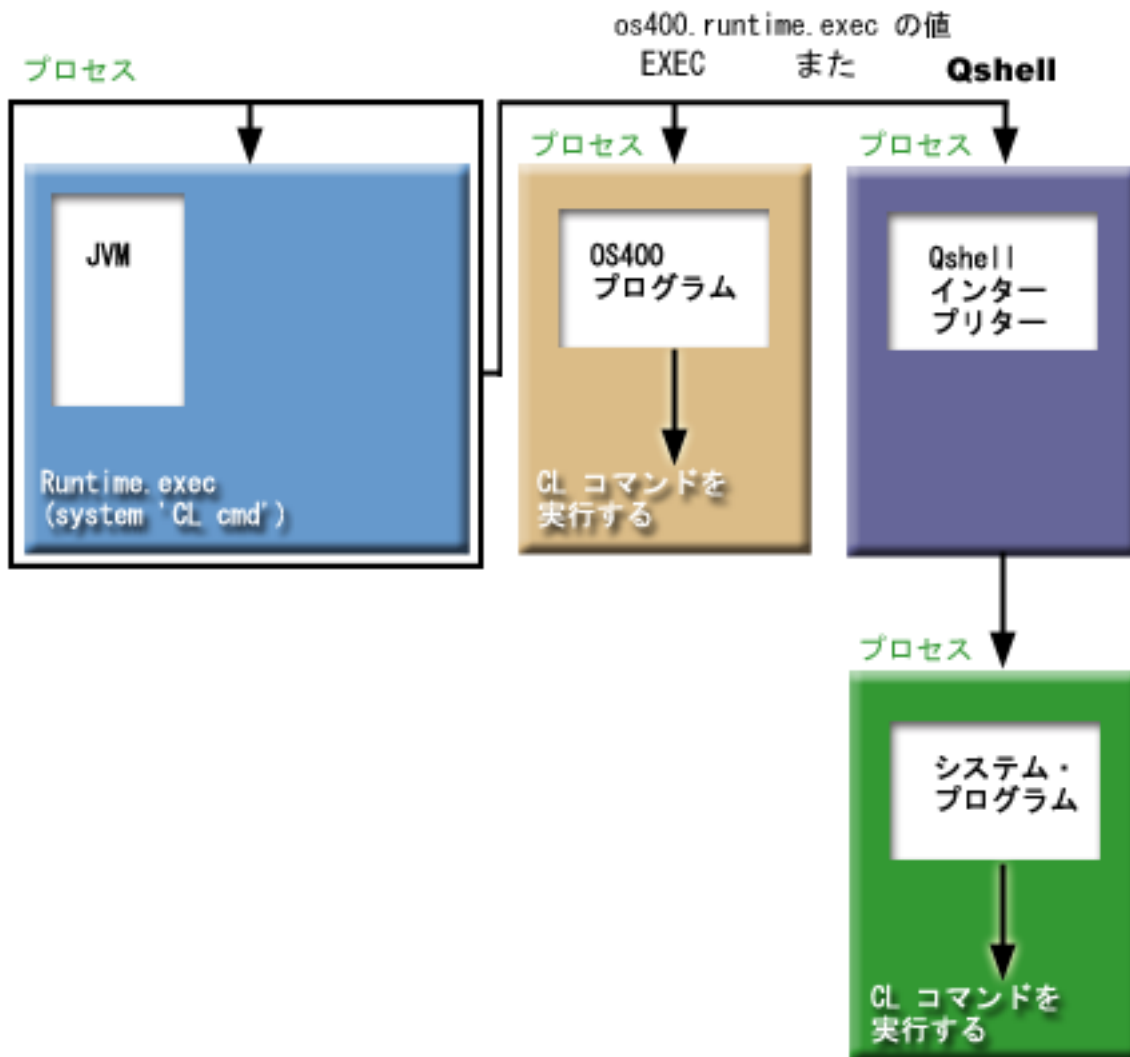


- `java.lang.Runtime.exec()` を使用して CL コマンドを呼び出すほうが、使用するシステム・リソースが少なくなります。

後方互換性のために必要な場合にだけ、`java.lang.Runtime.exec()` を使用して Qshell を実行する必要があります。`java.lang.Runtime.exec()` を使用して Qshell を実行する場合は、`os400.runtime.exec` が `QSHELL` に設定してある必要があります。

以下の図では、`QSHELL` の値を使用することによって、3 番目のジョブが立ち上げられ、その結果追加のシステム・リソースが消費されることになる過程が示されています。`QSHELL` の値を使用すると Java プログラムの移植性が低下するという点にご留意ください。

図 1. `os400.runtime.exec` システム・プロパティに `QSHELL` の値を使用する場合



また、`QSHELL` の値を使用する場合、`java.lang.Runtime.exec()` に CL コマンドを渡すには、特定の構文を使用する必要があります。詳しくは、CL コマンドを呼び出すための以下の例を参照してください。

`os400.runtime.exec` の設定について詳しくは、Java システム・プロパティのリストを参照してください。

## 例: java.lang.Runtime.exec() を使用して別の Java プログラムを呼び出す

この例では、java.lang.Runtime.exec() を使用して別の Java プログラムを呼び出す方法を示します。このクラスは、IBM Developer Kit for Java の一部として配布される Hello プログラムを呼び出します。Hello クラスが System.out に書き込みを行うときに、このプログラムは、ストリームへのハンドルを取得し、そこから読み取りを行うことができます。

注: プログラムを呼び出すには、Qshell インタープリターを使用します。

### 例 1: CallHelloPgm クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallHelloPgm
{
    public static void main(String args[])
    {
        Process theProcess = null;
        BufferedReader inStream = null;

        System.out.println("CallHelloPgm.main() invoked");

        // call the Hello class
        try
        {
            theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }

        // read from the called program's standard output stream
        try
        {
            inStream = new BufferedReader(
                new InputStreamReader( theProcess.getInputStream() ));
            System.out.println(inStream.readLine());
        }
        catch(IOException e)
        {
            System.err.println("Error on inStream.readLine()");
            e.printStackTrace();
        }

    } // end method
} // end class
```

背景情報については、java.lang.Runtime.exec() を使用するを参照してください。

## 例: java.lang.Runtime.exec() を使用して CL プログラムを呼び出す

この例では、Java プログラムから CL プログラムを実行する方法を示します。この例では、Java クラス CallCLPgm が CL プログラムを実行します。

CL プログラムは Java プログラム表示 (DSPJVAPGM) コマンドを使用して、Hello クラス・ファイルと関連しているプログラムを表示します。この例では、CL プログラムはコンパイル済みであり、JAVSAMPLIB と呼ばれるライブラリーに存在していることが前提となっています。CL プログラムからの出力は、QSYSPRT スプール・ファイルにあります。

Java プログラムから CL コマンドを呼び出す方法の例については、CL コマンドを呼び出すを参照してください。

注: JAVSAMPLIB は、IBM Developer Kit ライセンス・プログラム (LP) (番号 5722-JV1) のインストール・プロセスの一部としては作成されません。このライブラリーは明示的に作成する必要があります。

### 例 1: CallCLPgm クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallCLPgm
{
    public static void main(String[] args)
    {
        try
        {
            Process theProcess =
                Runtime.getRuntime().exec("/QSYS.LIB/JAVSAMPLIB.LIB/DSPJVA.PGM");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }
    } // end main() method
} // end class
```

### 例 2: Java CL プログラムの表示

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
          OUTPUT(*PRINT)
ENDPGM
```

背景情報については、`java.lang.Runtime.exec()` を使用するを参照してください。

### 例: `java.lang.Runtime.exec()` を使用して CL コマンドを呼び出す

この例では、Java プログラムから制御言語 (CL) コマンドを実行する方法を示します。

この例では、Java クラスが CL コマンドを実行します。CL コマンドは、「Java プログラムの表示 (DSPJVAPGM)」CL コマンドを使用して、Hello クラス・ファイルと関連しているプログラムを表示します。CL コマンドからの出力は、QSYSPRT スプール・ファイルにあります。

`os400.runtime.exec` システム・プロパティを (デフォルトの) EXEC に設定する場合、`Runtime.getRuntime().exec()` 関数に渡すコマンドは次のフォーマットになります。

```
Runtime.getRuntime().Exec("system CLCOMMAND");
```

ここで、`CLCOMMAND` は、実行しようとしている CL コマンドです。

注: os400.runtime.exec を QSHELL に設定する場合は、スラッシュと引用符 (¥) を追加する必要があります。たとえば、上のコマンドならば、次のようになります。

```
Runtime.getRuntime().Exec("system ¥"CLCOMMAND¥");
```

os400.runtime.exec と、それが java.lang.Runtime.exec() の使用に与える影響については、以下のページを参照してください。

java.lang.Runtime.exec() を使用する

Java システム・プロパティのリスト

## 例: CL コマンドを呼び出すためのクラス

以下のコードでは、os400.runtime.exec システム・プロパティにデフォルト値の EXEC を使用していることを想定しています。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallCLCom
{
    public static void main(String[] args)
    {
        try
        {
            Process theProcess =
                Runtime.getRuntime().exec("system DSPJVAPGM CLSF('/com/ibm/as400/system/Hello.class')
                OUTPUT(*PRINT)");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }
    } // end main() method
} // end class
```

背景情報については、java.lang.Runtime.exec() を使用するを参照してください。

## プロセス間通信

別のプロセスで実行されているプログラムと通信する際には、いくつかのオプションがあります。

オプションの 1 つは、プロセス間通信にソケットを使用することです。一方のプログラムは、サーバー・プログラムの役割を果たし、ソケット接続上で、クライアント・プログラムからの入力がないか listen します。クライアント・プログラムは、ソケットを使用してサーバーに接続します。ソケット接続が確立されると、どちらのプログラムでも情報を送受信することができます。

別のオプションは、プログラム間の通信にストリーム・ファイルを使用することです。これを行うには、System.in、System.out、および System.err クラスを使用します。

3 つ目のオプションは、データ待ち行列および iSeries メッセージ・オブジェクトを提供する IBM Toolbox for Java を使用することです。

また、他の言語から Java を呼び出すこともできます。詳細については、例: C から Java を呼び出すおよび 例: RPG から Java を呼び出すを参照してください。

## プロセス間通信のためにソケットを使用する

ソケット・ストリームは、異なるプロセス内で実行しているプログラム間での通信を行います。

プログラムは別個に開始することも、または `java.lang.Runtime.exec()` メソッドをメインの Java プログラム内から使用して開始することもできます。プログラムが Java 以外の言語で記述されている場合、情報交換用米国標準コード (ASCII) または拡張 2 進化 10 進コード (EBCDIC) 変換が確実に行われるようにしなければなりません。詳細については、Java 文字のエンコードを参照してください。

ソケットを使用する例については、例: プロセス間通信のためにソケットを使用するを参照してください。

### 例: プロセス間通信のためにソケットを使用する:

この例では、ソケットを使用して Java プログラムと C プログラムとの間で通信します。

最初に、ソケット上で聴取する C プログラムを開始してください。Java プログラムがソケットに接続された後は、ソケット接続を使用して C プログラムがそれにストリングを送ります。C プログラムから送られるストリングは、コード・ページ 819 の ASCII コードのストリングです。

Qshell インタープリターのコマンド行か、または他の Java プラットフォームで、コマンド `java TalkToC xxxxx nnnn` を使用して Java プログラムを開始しなくてはなりません。または iSeries コマンド行に `JAVA TALKTOC PARM(xxxxx nnnn)` を入力することにより、Java プログラムを開始します。xxxxx は、C プログラムが実行されているシステムのドメイン・ネームまたはインターネット・プロトコル (IP) アドレスです。nnnn は、C プログラムが使用するソケットのポート番号です。このポート番号は、C プログラムを呼び出すときに最初に渡すパラメーターとして指定する必要があります。

### 例 1: TalkToC クライアント・クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.net.*;
import java.io.*;

class TalkToC
{
    private String host = null;
    private int port = -999;
    private Socket socket = null;
    private BufferedReader inStream = null;

    public static void main(String[] args)
    {
        TalkToC caller = new TalkToC();
        caller.host = args[0];
        caller.port = new Integer(args[1]).intValue();
        caller.setUp();
        caller.converse();
        caller.cleanup();
    } // end main() method

    public void setUp()
    {
        System.out.println("TalkToC.setUp() invoked");

        try
        {
            socket = new Socket(host, port);
```

```

        inStream = new BufferedReader(new InputStreamReader(
                                   socket.getInputStream()));
    }
    catch(UnknownHostException e)
    {
        System.err.println("Cannot find host called: " + host);
        e.printStackTrace();
        System.exit(-1);
    }
    catch(IOException e)
    {
        System.err.println("Could not establish connection for " + host);
        e.printStackTrace();
        System.exit(-1);
    }
} // end setUp() method

public void converse()
{
    System.out.println("TalkToC.converse() invoked");

    if (socket != null && inStream != null)
    {
        try
        {
            System.out.println(inStream.readLine());
        }
        catch(IOException e)
        {
            System.err.println("Conversation error with host " + host);
            e.printStackTrace();
        }
    }

    } // end if
} // end converse() method

public void cleanUp()
{
    try
    {
        if(inStream != null)
        {
            inStream.close();
        }
        if(socket != null)
        {
            socket.close();
        }
    } // end try
    catch(IOException e)
    {
        System.err.println("Error in cleanup");
        e.printStackTrace();
        System.exit(-1);
    }
} // end cleanUp() method
} // end TalkToC class

```

SocketServ.C は、ポート番号のパラメーターを渡すことによって開始します。たとえば、CALL SocketServ '2001' とします。

## 例 2: SocketServ.C サーバー・プログラム



注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/time.h>

void main(int argc, char* argv[])
{
    int    portNum = atoi(argv[1]);
    int    server;
    int    client;
    int    address_len;
    int    sendrc;
    int    bndrc;
    char*  greeting;
    struct sockaddr_in local_Address;
    address_len = sizeof(local_Address);

    memset(&local_Address,0x00,sizeof(local_Address));
    local_Address.sin_family = AF_INET;
    local_Address.sin_port = htons(portNum);
    local_Address.sin_addr.s_addr = htonl(INADDR_ANY);

    #pragma convert (819)
    greeting = "This is a message from the C socket server.";
    #pragma convert (0)

    /* allocate socket */
    if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
    {
        printf("failure on socket allocation\n");
        perror(NULL);
        exit(-1);
    }

    /* do bind */
    if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
    {
        printf("Bind failed\n");
        perror(NULL);
        exit(-1);
    }

    /* invoke listen */
    listen(server, 1);

    /* wait for client request */
    if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
    {
        printf("accept failed\n");
        perror(NULL);
        exit(-1);
    }

    /* send greeting to client */
    if((sendrc = send(client, greeting, strlen(greeting),0))<0)
    {
        printf("Send failed\n");
        perror(NULL);
        exit(-1);
    }
}
```

```

    }

    close(client);
    close(server);
}

```

詳細については、プロセス間通信のためにソケットを使用するを参照してください。

## プロセス間通信に入出力ストリームを使用する

入出力ストリームは、別々のプロセスで実行されているプログラムの中で通信を行います。

`java.lang.Runtime.exec()` メソッドは、プログラムを実行します。親プログラムは、子プロセスの入出力ストリームへのハンドルを取得し、それらのストリームに対する書き込みおよび読み取りを行うことができます。子プログラムが Java 以外の言語で作成されている場合は、情報交換用米国標準コード (ASCII) コードまたは拡張 2 進化 10 進コード (EBCDIC) の変換が行われるようにしなければなりません。詳細については、Java 文字のエンコードを参照してください。

入出力ストリームを使用する例については、例: プロセス間通信に入出力ストリームを使用するを参照してください。

### 例: プロセス間通信に入出力ストリームを使用する:

この例では、Java から C プログラムを呼び出し、プロセス間通信に入出力ストリームを使用する方法を示します。

C プログラムは、その標準出力ストリームにストリングを書き込み、Java プログラムは、このストリングを読み取り、表示します。この例では、`JAVSAMPLIB` というライブラリーが作成されていることと、その中で `CSAMP1` プログラムが作成されていることを前提としています。

注: `JAVSAMPLIB` は、IBM Developer Kit ライセンス・プログラム (LP) (番号 5722-JV1) のインストール・プロセスの一部としては作成されません。明示的にそれを作成しなければなりません。

### 例 1: CallPgm クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.io.*;

public class CallPgm
{
    public static void main(String args[])
    {
        Process theProcess = null;
        BufferedReader inStream = null;

        System.out.println("CallPgm.main() invoked");

        // call the CSAMP1 program
        try
        {
            theProcess = Runtime.getRuntime().exec(
                "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }
    }
}

```

```

    }

    // read from the called program's standard output stream
    try
    {
        inStream = new BufferedReader(new InputStreamReader
            (theProcess.getInputStream()));
        System.out.println(inStream.readLine());
    }
    catch(IOException e)
    {
        System.err.println("Error on inStream.readLine()");
        e.printStackTrace();
    }

} // end method

} // end class

```

## 例 2: CSAMPI C プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
    /* Convert the string to ASCII at compile time */
    #pragma convert(819)
    printf("Program JAVSAMPLIB/CSAMP1 was invoked\n");
    #pragma convert(0)
    /* Stdout may be buffered, so flush the buffer */

    fflush(stdout);
}

```

詳細については、プロセス間通信に入出力ストリームを使用するを参照してください。

## 例: C から Java を呼び出す

次に示すのは、system() 関数を使用して Java Hello プログラムを呼び出す C プログラムの例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

#include <stdlib.h>

int main(void)
{
    int result;

    /* The system function passes the given string to the CL command processor
    for processing. */

    result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
}

```

## 例: RPG から Java を呼び出す

次に示すのは、QCMDEXC API を使用して Java Hello プログラムを呼び出す RPG プログラムの例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
D*          DEFINE  THE PARAMETERS FOR THE QCMDEXC API
D*
DCMDSTRING      S          25      INZ('JAVA CLASS(''com.ibm.as400.system.Hello'')')
DCMDLENGTH      S          15P 5  INZ(25)
D*          NOW THE CALL TO QCMDEXC WITH THE 'JAVA' CL COMMAND
C              CALL      'QCMDEXC'
C              PARM          CMDSTRING
C              PARM          CMDLENGTH
C*          This next line displays 'DID IT' after you exit the
C*          Java Shell via F3 or F12.
C          'DID IT'  DSDPLY
C*          Set On LR to exit the RPG program
C              SETON          LR
C
```

---

## Java プラットフォーム

Java プラットフォームは、Java アプレットおよびアプリケーションを開発して管理するための環境です。これは、3 つの主要なコンポーネント (Java 言語、Java パッケージ、および Java 仮想マシン) で構成されます。

Java 言語およびパッケージは、C++ およびそのクラス・ライブラリーと類似しています。Java パッケージにはクラスが含まれていて、どの準拠 Java 実装でも利用できます。アプリケーション・プログラミング・インターフェース (API) は、Java をサポートするどのシステムでも同じはずです。

Java が C++ のような従来型の言語と異なる点は、コンパイルして実行する方法です。従来型のプログラミング環境では、プログラムのソース・コードを作成してコンパイルし、特定ハードウェアおよびオペレーティング・システムのオブジェクト・コードにします。このオブジェクト・コードを別のオブジェクト・コード・モジュールにバインドして、実行プログラムを作成します。このコードは、特定のコンピューター・ハードウェア・セットに固有なものですから、変更を加えることなしには、別のシステムでは稼働しません。

## Java アプレットおよびアプリケーション

アプレットは、HTML Web 文書に含めるよう設計された Java プログラムです。Java アプレットを作成し、イメージを組み込むのとほとんど同じ方法で、HTML ページに組み込むことができます。Java を利用できるブラウザを使用して、アプレットを含む HTML ページを表示すると、アプレットのコードがシステムに転送され、ブラウザの Java 仮想マシンで実行されます。

HTML 文書には、Java アプレットの名前と、その URL が指定されたタグが含まれます。URL は、そのアプレットのバイトコードが存在するインターネット上の場所を示します。Java アプレットのタグが含まれた HTML 文書が表示されると、Java を使用できる Web ブラウザーはインターネットから Java バイトコードをダウンロードし、Web 文書内のコードを処理するために Java 仮想マシンを使用します。これらの Java アプレットを使って、Web ページにグラフィックスのアニメーション表示や、対話式のコンテンツを含めることができます。

また、Web ブラウザーを使用しない Java アプリケーションを作成することも可能です。

詳しくは、Sun Microsystems の Java アプレットのチュートリアル、Writing Applets を参照してください。このページには、アプレットの概要やアプレットの記述方法、およびアプレットに関連した一般的な問題が含まれています。

アプリケーションは、ブラウザを使用せずに実行できる、スタンドアロン・プログラムです。Java アプリケーションは、コマンド入力行から Java インタープリターを開始することによって、またコンパイルされたアプリケーションのファイルを指定することによって実行できます。通常、アプリケーションは配置されたシステム上にあります。アプリケーションはシステム上のリソースにアクセスしますが、そのアクセスは Java セキュリティー・モデルによって制限されます。

## Java 仮想マシン

Java 仮想マシンは、Web ブラウザーまたは任意のオペレーティング・システム (IBM i5/OS など) に追加できるランタイム環境です。Java 仮想マシンは Java コンパイラーが生成する命令を実行します。これは、バイトコード・インタープリターとランタイムで構成されています。このランタイムでは、もともと開発されたプラットフォームに関係なく、任意のプラットフォームで Java クラス・ファイルを実行できます。

クラス・ローダーおよびセキュリティ・マネージャーは、Java ランタイムの一部で、別のプラットフォームからのコードを隔離します。また、ロードされるクラスがアクセスするたびに、システム・リソースを制限できます。

注: Java アプリケーションは制限されません。制限されるのはアプレットだけです。アプリケーションは自由にシステム・リソースにアクセスして、ネイティブ・メソッドを使用できます。ほとんどの IBM Developer Kit for Java プログラムはアプリケーションです。

「Java プログラムの作成 (CRTJVAPGM)」コマンドを使用して、バイトコードを検査するために Java ランタイムによって課されている安全要件を、コードが満たしているか確認できます。これには、タイプ制限の施行、データ変換の検査、パラメーター・スタックのオーバーフローまたはアンダーフローが発生していないかの確認、およびアクセス違反の検査が含まれます。しかし、バイトコードの検査は明示的に行う必要はありません。前もって CRTJVAPGM コマンドを使用しない場合、クラスの最初の使用時に検査が行われます。バイトコードが検査されると、インタープリターはバイトコードをデコードし、目的の操作を実行するのに必要なマシン・インストラクションを実行します。

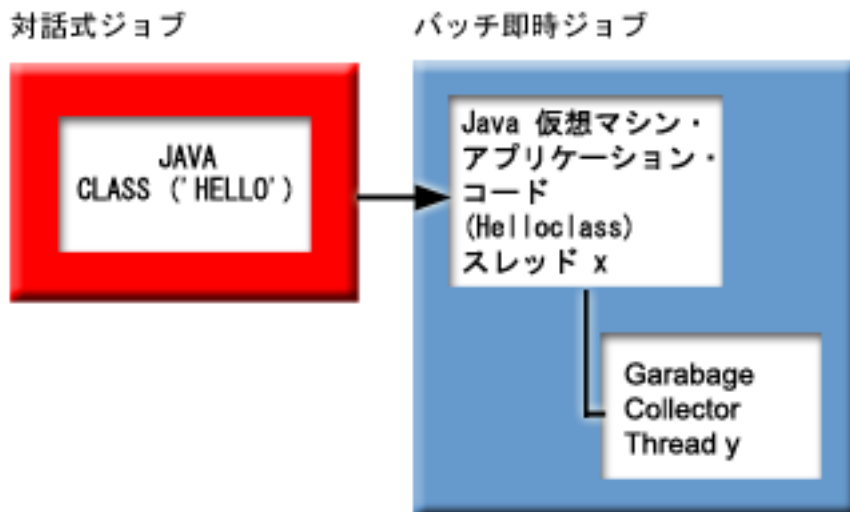
注: 256 ページの『Java インタープリター』は、OPTIMIZE(\*INTERPRET) または INTERPRET(\*YES) を指定した場合にのみ使用されます。

バイトコードのロードおよび実行のほか、Java 仮想マシンには、メモリーを管理するガーベッジ・コレクターが含まれています。ガーベッジ・コレクションは、バイトコードのロードおよび解釈と同時に実行されます。

## Java ランタイム環境

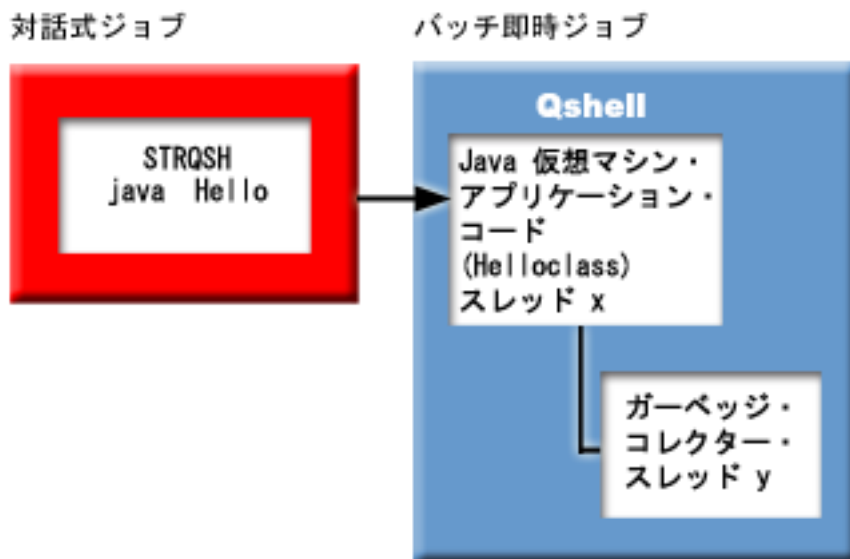
iSeries のコマンド行に「Java プログラムの実行 (RUNJVA)」コマンドまたは JAVA コマンドを入力すると、Java ランタイム環境が開始されます。Java 環境はマルチスレッドをサポートするので、バッチ即時 (BCI) ジョブなどのスレッドをサポートするジョブで Java 仮想マシンを実行する必要があります。以下の図で示されているように、Java 仮想マシンが起動されると、ガーベッジ・コレクターが実行するジョブで追加のスレッドを開始されます。

図 1: RUNJVA または JAVA CL コマンドを使用する場合の標準的な Java 環境



また、Qshell インタープリターから Qshell 内で java コマンドを使って Java ランタイム環境を開始することもできます。この環境では、Qshell インタープリターは対話式ジョブと関連付けられた BCI ジョブで実行されます。Java ランタイム環境は、Qshell インタープリターが実行されているジョブで開始されます。

図 2: Qshell で Java コマンドを使用する場合の Java 環境



Java ランタイム環境を対話式ジョブから開始すると、Java シェル画面が表示されます。この画面の入力行を使って、System.in ストリームにデータを入力することができます。また、System.out ストリームや System.err ストリームに書き込まれたデータも表示されます。



## Java インタープリター

Java インタープリターは、特定のハードウェア・プラットフォームで Java クラス・ファイルを解釈する、Java 仮想マシンの一部です。Java インタープリターは各バイトコードをデコードし、そのバイトコードに関する一連のマシン・インストラクションを実行します。

関連トピック:

- Java クラス・ファイル

## Java JAR とクラス・ファイル

Java ARchive (JAR) ファイルは、複数のファイルを 1 つに結合したファイル・フォーマットです。Java 環境と他のプログラミング環境の異なる点は、Java コンパイラーでは、ハードウェア固有の命令セット用にマシン・コードを生成しないということです。代わりに、Java コンパイラーは、Java ソース・コードを Java 仮想マシンの命令に変換し、それらを Java クラス・ファイルに保管します。JAR ファイルを使用して、クラス・ファイルを保管することができます。クラス・ファイルは特定のハードウェア・プラットフォームを対象にすることはありませんが、Java 仮想マシン・アーキテクチャーを対象とします。

JAR は一般的なアーカイブ・ツールとして使用でき、すべてのタイプ (アプレットも含む) の Java プログラムを配布できます。Java アプレットは、1 つずつ新しい接続をオープンするのではなく、一度の Hypertext Transfer Protocol (HTTP) トランザクションで、ブラウザにダウンロードします。この方式でのダウンロードでは、Web ページ上でアプレットがロードして機能を開始する速度が向上します。

JAR はクロスプラットフォームの唯一のアーカイブ・フォーマットです。また JAR は、オーディオ・ファイルおよびイメージ・ファイル、さらにクラス・ファイルを処理する唯一のフォーマットでもあります。JAR は Java で記述される、オープン・スタンダードで十分に拡張可能なフォーマットです。

また、JAR フォーマットは圧縮もサポートしています。これは、ファイルのサイズを減らし、ダウンロード時刻を短縮します。さらにアプレット作成者は、作成元を認証するために、JAR ファイル内のそれぞれの項目にデジタルで署名できます。

JAR ファイル内のクラスを更新する場合、Java jar ツールを参照してください。

**Java クラス・ファイル**は、Java コンパイラーがソース・ファイルをコンパイルするときに作成される、ストリーム・ファイルです。クラス・ファイルには、クラスの各フィールドおよびメソッドを記述するテーブルが含まれています。またこのファイルには、各メソッドのバイトコード、静的データ、および Java オブジェクトを表すのに使用される記述も含まれています。

## Java スレッド

スレッドとは、プログラム内で実行される、単独の独立したストリームのことを言います。Java はマルチスレッド・プログラミング言語であるため、Java 仮想マシン内では、一度の複数のスレッドを実行することができます。Java スレッドは、Java プログラムが同時に複数のタスクを実行するための手段として使用されます。スレッドは、本質的にプログラム内の制御のフローです。

スレッドは、並行プログラムをサポートし、アプリケーションのパフォーマンスとスケーラビリティを向上させるのに使用される、現代的なプログラミング構成要素です。ほとんどのプログラミング言語では、アドイン・プログラミング・ライブラリーを使用することによってスレッドをサポートします。Java の場合は、組み込みアプリケーション・プログラム・インターフェース (API) として、スレッドをサポートしています。

注: スレッドを使用すると、より多くのタスクが並行して実行されるため、対話性の向上、つまりキーボードでの待機時間の短縮がサポートされます。ただし、プログラムの対話機能は、必ずしもスレッドがあるというだけで向上するとは限りません。

スレッドは、実行時間の長い対話で待機しながら、プログラムがなおその他の作業も処理できるようにするためのメカニズムです。スレッドを使用すると、同じコード・ストリームの中で複数のフローをサポートすることができます。これは、**軽量プロセス**と呼ばれることもあります。Java 言語には、スレッドの直接サポートも組み込まれています。しかし、設計上、割り込みや複数の待ちがある非同期で非ブロッキングの入出力は、サポートされていません。

スレッドを使用すると、マシンに複数のプロセッサがある環境に適した、並列プログラムを作成できます。これは、適切に構成されれば、複数のトランザクションやユーザーの処理のためのモデルともなります。

Java プログラムのスレッドは、さまざまな状況で使用できます。プログラムの中には、複数のアクティビティに携わることができなければならず、なおかつユーザーからのさらに別の入力にも応答できなければならぬものがあります。たとえば、Web ブラウザーには、音声を再生しながらユーザーの入力に応答する能力が求められるでしょう。

スレッドでは、非同期メソッドを使用することもできます。2 つ目のメソッドを呼び出したときに、1 つ目のメソッドが完了するまで 2 つ目のメソッドが自身のアクティビティを続けるのを待つ必要はありません。

ただし、スレッドを使用しないほうが良い場合もたくさんあります。階層的な順次の論理が使用されるプログラムでは、1 つのスレッドでシーケンス全体を完了させることができます。このようなケースでは、複数のスレッドを使用してもプログラムが複雑になるだけで、何の益もありません。スレッドの作成と開始には、かなりの作業が伴います。操作に関するステートメントが 2 つか 3 つしかないのであれば、それは 1 つのスレッドで扱った方が速いでしょう。これは、その操作が概念的に非同期である場合でもそういえません。複数のスレッドがオブジェクトを共用すると、オブジェクトには、スレッド・アクセスを調整し、整合性を保守するための同期化が必要になります。同期化を行うとなれば、プログラムはそれだけ複雑になり、パフォーマンスを最適化するための調整を難しくしたり、プログラミングのソースにエラーを引き起こしてしまう可能性があります。

スレッドについての詳細は、マルチスレッド・アプリケーションの作成を参照してください。

## Sun Microsystems, Inc. Java Development Kit

Java Development Kit (JDK) は、Java 開発者のために、Sun Microsystems, Inc. によって配布されるソフトウェアです。このソフトウェアには、Java インタープリター、Java クラス、および Java 開発ツール (コンパイラー、デバッガー、逆アセンブラー、appletviewer、スタブ・ファイル・ジェネレーター、および文書ジェネレーター) が含まれています。

JDK では、一度開発されたアプリケーションを作成し、任意の Java 仮想マシン上の任意の場所で実行することができます。ある 1 つのシステムで JDK を使用して開発された Java アプリケーションを、コードの変更や再コンパイルを行うことなく、他のシステムでも使用することが可能です。Java クラス・ファイルは、標準の Java 仮想マシンであれば、そのマシンにでも移植できます。

現在の JDK に関する詳細な情報を得るには、ご使用の iSeries サーバーにインストールされている IBM Developer Kit for Java のバージョンを確認してください。

ご使用の iSeries サーバーで使用されているデフォルトの IBM Developer Kit for Java の Java 仮想マシンのバージョンは、以下のいずれかのコマンドを入力することで確認できます。

- `java -version` (Qshell コマンド・プロンプトの場合)
- `RUNJAVA CLASS(*VERSION)` (CL コマンド行の場合)

次に、The Source for Java Technology [java.sun.com](http://java.sun.com) のページで同じバージョンの Sun Microsystems, Inc. JDK を探し、具体的な資料を見つけてください。IBM Developer Kit for Java は、Sun Microsystems, Inc. の Java Technology と互換性のある製品であるため、その JDK 資料に精通しておくことは必要でしょう。

詳細は、以下のトピックを参照してください。

- 複数の Java Development Kits (JDK) のサポートでは、さまざまな Java 仮想マシンの使用に関する情報を扱っています。
- ネイティブ・メソッドおよび Java ネイティブ・インターフェースでは、ネイティブ・メソッドとは何か、そしてネイティブ・メソッドで何ができるかを定義しています。また、このトピックでは、Java ネイティブ・インターフェースについて簡単に説明しています。

## Java パッケージ

Java パッケージは、Java に関連するクラスとインターフェースをグループ化する 1 つの方法です。Java パッケージは、他の言語で利用可能なクラス・ライブラリーと同様のものです。

Java API を同梱する Java パッケージは、Sun Microsystems, Inc. Java Development Kit (JDK) の一部として入手できます。Java パッケージと Java API の情報の完全なリストについては、Java 2 Platform パッケージ (Java 2 Platform Packages) を参照してください。

## Java ツール

Sun Microsystems, Inc. Java Development Kit で提供されているツールの完全なリストは、Tools Reference by Sun Microsystems, Inc. を参照してください。IBM Developer Kit for Java がサポートしている個々のツールのそれぞれについては、IBM Developer Kit for Java がサポートする Java ツールを参照してください。

---

## 高度なトピック

このトピックでは、バッチ・ジョブで Java を実行する方法を説明し、Java プログラムを表示、実行、またはデバッグするために、統合ファイル・システムに必要な Java ファイル権限について説明します。

## Java クラス、パッケージ、およびディレクトリー

Java のクラスはそれぞれ、あるパッケージに属しています。どのクラスがどのパッケージに含まれるかは、Java ソース・ファイルの最初のステートメントに記述されます。ソース・ファイルにパッケージ・ステートメントがない場合、そのクラスは名前のないデフォルトのパッケージに含まれると見なされます。

パッケージ名は、クラスの位置するディレクトリー構造と関連があります。統合ファイル・システムでは、多くの PC システムや UNIX<sup>®</sup> システムと同様に、階層ファイル構造で Java クラスを格納できます。Java クラスを格納するディレクトリーの相対ディレクトリー・パスは、そのクラスが属するパッケージの名前と一致していなければなりません。たとえば、次の Java クラスで考えてみます。

```
package classes.geometry;
import java.awt.Dimension;

public class Shape {
```

```

Dimension metrics;

    // The implementation for the Shape class would be coded here ...
}

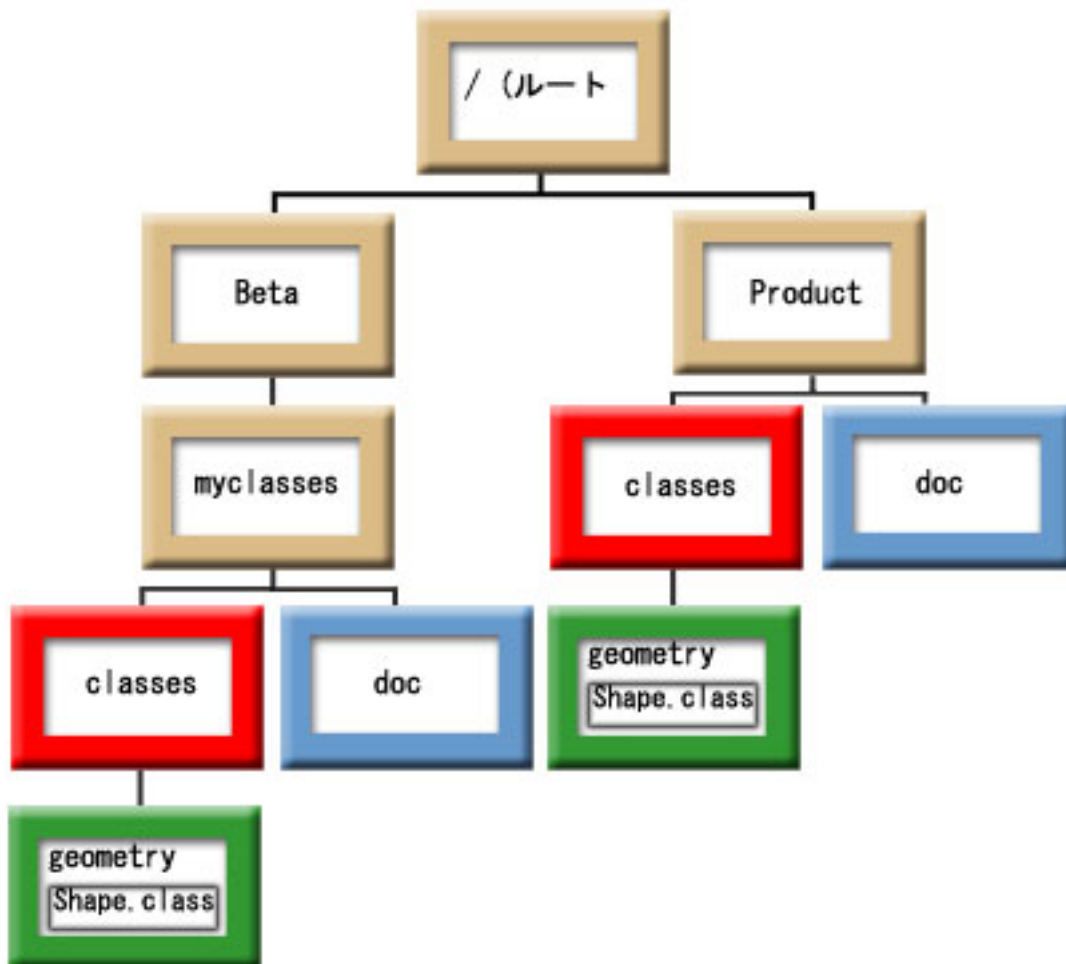
```

上記のコードのパッケージ・ステートメントは、Shape クラスが `classes.geometry` パッケージに属していることを示しています。したがって、Java ランタイムが Shape クラスを検出するためには、Shape クラスが相対ディレクトリ構造の `classes/geometry` に格納されていなければなりません。

注: パッケージ名は、クラスが格納されているディレクトリの相対ディレクトリ名に対応しています。Java 仮想マシンのクラス・ローダーは、クラスパスで指定された各ディレクトリに相対パス名を追加してクラスを探します。また、Java 仮想マシンのクラス・ローダーは、クラスパスで指定された ZIP ファイルまたは JAR ファイルを検索してクラスを検出することもできます。

たとえば、Shape クラスが「ルート」(/) ファイル・システムの `/Product/classes/geometry` ディレクトリに格納されている場合は、クラスパスに `/Product` を指定する必要があります。

図 1: 異なるパッケージ内にある同じ名前の Java クラスのディレクトリ構造の例



注: Shape クラスの複数のバージョンをディレクトリー構造に格納することができます。 Shape クラスのベータ版を使用するには、CLASSPATH で、Shape クラスが格納されている他のディレクトリーや ZIP ファイルの前に /Beta/myclasses を指定します。

Java コンパイラーは、Java ソース・コードをコンパイルするときに、Java クラスパス、パッケージ名、およびディレクトリー構造を使ってパッケージとクラスを探します。詳細については、Java クラスパスを参照してください。

## 統合ファイル・システム内のファイル

統合ファイル・システムには、Java 関連のクラス、ソース、ZIP、および JAR ファイルが、階層ファイル構造で格納されます。 IBM Developer Kit for Java は、統合ファイル・システム内のスレッド・セーフ・ファイル・システムを使用して Java 関連のクラス・ファイル、ソース・ファイル、および JAR ファイルを格納および処理するための支援をします。

スレッド・セーフ・ファイル・システムについて、およびファイル・システムの比較について詳しくは、以下を参照してください。

マルチスレッド・プログラミングでのファイル・システムについての考慮事項

ファイル・システムの比較

## 統合ファイル・システム内の Java ファイル権限

Java プログラムを実行およびデバッグするには、クラス・ファイル、JAR ファイル、および ZIP ファイルに読み取り権限 (\*R) が必要です。また、すべてのディレクトリーに読み取りおよび実行権限 (\*RX) が必要です。

「Java プログラムの作成 (CRTJVAPGM)」コマンドを使ってプログラムを最適化するには、クラス・ファイル、JAR ファイル、または ZIP ファイルに読み取り権限 (\*R) が必要であり、ディレクトリーに実行権限 (\*X) が必要です。クラス・ファイル名にパターンを使用している場合は、ディレクトリーに読み取りおよび実行権限 (\*RX) が必要です。

「Java プログラムの削除 (DLTJVAPGM)」コマンドを使って Java プログラムを削除するには、クラス・ファイルに対して読み取りおよび書き込み権限 (\*RW) が必要です。クラス・ファイル名にパターンを使用している場合は、ディレクトリーに読み取りおよび実行権限 (\*RX) が必要です。

「Java プログラムの表示 (DSPJVAPGM)」コマンドを使用して Java プログラムを表示するには、クラス・ファイルへの読み取り権限 (\*R) が必要であり、ディレクトリーには実行権限 (\*X) が必要です。

注: 実行権限 (\*X) がないファイルとディレクトリーは、常に QSECOFR 権限があるユーザーに対する実行権限 (\*X) があるように表示されます。ユーザーの両方が同じファイルに同じアクセスをしても、特定の状況で、異なるユーザーが異なる結果を得ることがあるかもしれません。このことは、Qshell インタープリターまたは `java.Runtime.exec()` を使用してシェル・スクリプトを実行するときに知っておく必要があります。

たとえば、一人のユーザーがシェル・スクリプトを呼び出すために `java.Runtime.exec()` を使用する Java を作成して、それを、QSECOFR 権限のあるユーザー ID を使用してテストします。シェル・スクリプトのファイル・モードに読み取りおよび書き込み権限が (\*RW) ある場合、統合ファイル・システムはそれを、QSECOFR 権限のあるユーザー ID が実行することを許可します。しかし、非 QSECOFR 権限ユーザーは同じ Java プログラムを実行しようとすることができますが、統合ファイル・システムは、\*X が欠落して



いるため、`java.Runtime.exec()` コードにシェル・スクリプトは実行できないことを知らせることができるでしょう。この場合、`java.Runtime.exec()` は入出力の例外をスローします。

Java プログラムによって統合ファイル・システム内に作成された新規ファイルに対して権限を割り当てることもできます。ファイルは `os400.file.create.auth` システム・プロパティ、ディレクトリーは `os400.dir.create.auth` を使用して、読み取り、書き込み、および実行権限の組み合わせを設定することができます。

詳しくは、プログラムおよび CL コマンド API または統合ファイル・システムを参照してください。

## バッチ・ジョブで Java を実行する

「ジョブ投入 (SBMJOB)」コマンドを使うと、Java プログラムはバッチ・ジョブ内で実行します。このモードでは、Java Qshell コマンド入力画面を、`System.in`、`System.out`、`System.err` ストリームを処理するために使用することはできません。

これらのストリームは、他のファイルに転送することができます。デフォルトの処理では、`System.out` および `System.err` ストリームはスプール・ファイルに送信されます。`System.in` からの読み取り要求で入出力例外を出すバッチ・ジョブは、スプール・ファイルを所有します。Java プログラム内で `System.in`、`System.out`、および `System.err` の転送を行うことができます。また、`os400.stdin`、`os400.stdout`、および `os400.stderr` システム・プロパティを使用して、`System.in`、`System.out`、および `System.err` を転送することもできます。

**注:** SBJJOB コマンドを実行すると、ユーザー・プロファイルで指定された HOME ディレクトリーが現行作業ディレクトリー (CWD) に設定されます。

**例:** バッチ・ジョブで Java を実行する

```
SBMJOB CMD(JAVA Hello OPTION(*VERBOSE)) CPYENVVAR(*YES)
```

上記の例で `JAVA` コマンドを実行すると、2 番目のジョブが作成されます。ですから、バッチ・ジョブが実行されるサブシステムは、複数のジョブを実行できなければなりません。

バッチ・ジョブが複数のジョブを実行できることを、以下のステップに従うことによって検証できます。

1. CL コマンド行で `DSPSBSD(MYSBSD)` と入力する。ここで、`MYSBSD` は、バッチ・ジョブのサブシステム記述を表します。
2. オプション 6 のジョブ待ち行列項目を選ぶ。
3. ジョブ待ち行列の `Max Active` フィールドを参照する。

---

## グラフィカル・ユーザー・インターフェースを使用しないホスト上で Java アプリケーションを実行する

Java アプリケーションを、iSeries サーバーなどのグラフィカル・ユーザー・インターフェース (GUI) のないホスト上で実行したい場合は、`Native Abstract Windowing Toolkit (NAWT)` を使用することができます。

NAWT を使用して、Java アプリケーションおよびサーブレットに、Java 2 Software Development Kit (J2SDK) Standard Edition AWT のグラフィックス機能の全機能を提供することができます。



## Native Abstract Windowing Toolkit

Native Abstract Windowing Toolkit (NAWT) は、Java アプリケーションおよびサーブレットが、Java 2 Software Development Kit (J2SDK), Standard Edition で提供される Abstract Windowing Toolkit (AWT) グラフィックス機能を使用できるようにします。

注: NAWT は現在はロケール固有および言語固有のフォントおよび文字セットをサポートしていません。NAWT を使用する際は、以下の要件を満たしてください。

- ISO8859-1 文字セットで定義されている文字のみを使用してください。
- font.properties ファイルを使用してください。font.properties ファイルは、/QIBM/ProdData/Java400/jdknn/lib ディレクトリー (*nn* は使用している J2SDK のバージョン番号) にあります。特に font.properties.xxx ファイル (*xxx* は言語またはその他の修飾子) は使用しないでください。

通常、NAWT は基本的なグラフィックス・エンジンとして X Window システムを使用します。X Window システムを使用するには、X サーバーが必要です。X サーバーは、X クライアント・プログラムからの接続および要求を受け入れる独立型のアプリケーションです。その場合は、基礎となる NAWT インフラストラクチャーが X クライアント・プログラムになります。

推奨される X サーバーは AT<sup>®</sup>&T Virtual Network Computing (VNC) サーバーです。VNC サーバーは専用のマウス、キーボード、およびグラフィックス機能付きモニターを必要としないため、iSeries サーバーによく適しています。IBM は、i5/OS Portable Application Solutions Environment (i5/OS Portable Application Solutions Environment (OS/400 PASE) で実行する VNC サーバーのバージョンを提供しています。i5/OS PASE は、IBM AIX オペレーティング・システム用にコンパイルされたほとんどのバイナリー実行可能プログラムを実行できる、UNIX に似た環境です。i5/OS PASE は、i5/OS の一部としてインストールされます。

i5/OS PASE で VNC サーバーを実行する場合は、iSeries サーバーがすべての NAWT グラフィックスの計算を実行するので、外部グラフィックス・サーバーは不要です。以下は NAWT および J2SDK についての情報であり、i5/OS PASE において VNC を入手し、セットアップする方法について説明しています。

NAWT のインストールと使用について詳しくは、以下を参照してください。

NAWT を実行するには i5/OS PASE および VNC を使用する必要があるため、これらのアプリケーションについて詳しく知る必要があるかもしれません。詳しくは、以下を参照してください。

i5/OS PASE

Virtual Network Computing

### NAWT サポートのレベル

使用する Java 2 Software Development Kit (J2SDK) Standard Edition のバージョンは、使用可能な Native Abstract Windowing Toolkit (NAWT) サポートの選択肢に影響します。NAWT をインストールする前に、どのタイプのサポートが自分の要件に合っているかを理解しておく必要があります。この情報を使用して、グラフィカル要件を見積もり、実行する必要がある J2SDK のバージョンを選択してください。この情報を使用して、グラフィカル要件を見積もり、実行する必要がある J2SDK のバージョンを選択してください。

### NAWT および J2SDK バージョン 1.3

J2SDK バージョン 1.3 の場合、NAWT は、直接のユーザー対話の必要のないグラフィカル Java アプリケーションのみをサポートします。このレベルのサポートは、iSeries サーバー上でイメージ・データ

(JPEG、GIFなどでエンコードされたもの)を生成する Java アプリケーション、サーブレット、およびグラフィックスのパッケージに適しています。

## NAWT および J2SDK バージョン 1.4 およびそれ以降

J2SDK バージョン 1.4 および後続のバージョンの場合、NAWT は、対話式のグラフィカル・ユーザー・インターフェース (GUI) と Java ヘッドレス AWT 環境を含むすべての Java Abstract Windowing Toolkit (AWT) の機能をサポートします。

J2SDK バージョン 1.4 およびそれ以降の実行時に使用可能な NAWT サポートについて詳しくは、以下を参照してください。

### J2SDK バージョン 1.3 での NAWT のインストールおよび使用:

Java 2 Software Development Kit (J2SDK) バージョン 1.3 で Native Abstract Windowing Toolkit (NAWT) をインストールするには、以下のタスクを完了してください。

1. NAWT ソフトウェア修正のインストール
2. iSeries Tools for Developers PRPQ のインストール

NAWT を使用したり、NAWT のインストールをテストしたりするには、まず Virtual Network Computing (VNC) サーバー用のパスワード・ファイルを作成する必要があります。以下に、追加の必須およびオプションのステップをリストします。

VNC パスワード・ファイルの作成

VNC サーバーの開始 (通常は各 IPL の後)

環境変数の構成 (Java の実行前に毎回)

Java システム・プロパティの構成 (Java の実行前に毎回)

NAWT インストールの検査 (オプション)

### リンク集

NAWT ソフトウェア修正のインストール

NAWT をインストールする場合は、使用する Java 2 Software Development Kit (J2SDK) Standard Edition のバージョン用の適切な NAWT サポートが組み込まれたソフトウェア修正をインストールする必要があります。

iSeries Tools for Developers PRPQ のインストール

Native Abstract Windowing Toolkit (NAWT) を実行するには、iSeries Tools for Developers PRPQ (5799PTL) をインストールする必要があります。PRPQ がない場合は、注文する必要があります。

VNC パスワード・ファイルの作成

Native Abstract Windowing Toolkit (NAWT) をインストールして実行するには、Virtual Network Computing (VNC) サーバーのパスワード・ファイルを作成する必要があります。VNC サーバーのデフォルトの設定においては、無許可ユーザーのアクセスから VNC の表示を保護するためにこれが使用するパスワード・ファイルが必要です。VNC パスワード・ファイルは、VNC サーバーを開始するプロファイルの下に作成する必要があります。

VNC サーバーの開始

## 環境変数の構成

NAWT を使用して Java を実行するときは、システム名、ディスプレイ番号、および各 X サーバーとそれに関連した .Xauthority ファイルの場所を Java に示す環境変数が設定されていなければなりません。

## Java システム・プロパティーの構成

### NAWT インストールの検査

## NAWT ソフトウェア修正のインストール:

NAWT をインストールする場合は、使用する Java 2 Software Development Kit (J2SDK) Standard Edition のバージョン用の適切な NAWT サポートが組み込まれたソフトウェア修正をインストールする必要があります。

ソフトウェア修正 (PTF) をインストールする前に、サーバー・ソフトウェアに、使用する J2SDK のバージョンに対応するライセンス・プログラム 5722JV1 のオプションが含まれていることを確認してください。ご使用のサーバー・ソフトウェアのオプションを検査するには、以下のステップを完了してください。

1. i5/OS コマンド行で、「ライセンス・プログラムの処理 (GO LICPGM)」を入力し、**ENTER** を押します。
2. オプション 10 (インストール済みライセンス・プログラムの表示) を選択して、使用する JDK のバージョンに対応するライセンス・プログラム 5722JV1 オプションがインストールされているか検査します。

必ず最新の Java グループ・ソフトウェア修正を適用して、任意の最新の NAWT 修正を取り入れてください。

以下の表には、NAWT を実行するための必須オプションとソフトウェア修正要件をリストしています。

| J2SDK のバージョン | 5722JV1 のオプション | Java ソフトウェア修正 ID          | PTF の日付 |
|--------------|----------------|---------------------------|---------|
| 1.3          | 5              | PTF グループ 5722-JV1 SF99269 | 最新      |
| 1.4          | 6              | PTF グループ 5722-JV1 SF99269 | 最新      |
| 1.5          | 7              | PTF グループ 5722-JV1 SF99269 | 最新      |

ソフトウェア修正について詳しくは、ソフトウェア修正の使用を参照してください。

## iSeries Tools for Developers PRPQ のインストール:

Native Abstract Windowing Toolkit (NAWT) を実行するには、iSeries Tools for Developers PRPQ (5799PTL) をインストールする必要があります。PRPQ がない場合は、注文する必要があります。

PRPQ の最近のバージョンには、プリコンパイルされた i5/OS PASE 対応バージョンの Virtual Network Computing (VNC) が組み込まれています。古いバージョンには VNC は組み込まれていません。PRPQ のインストール方法は、バージョンによって次のように異なります。

- 2002 年 6 月 14 日以降に注文された PRPQ のバージョンの場合: iSeries Virtual Innovation Center Web サイトにあるインストールの指示に従ってこのタスクを完了します。

**注:** PRPQ で利用可能な VNC サポートをインストールするには、Web サイトにあるインストールの指示だけに従ってください。セットアップの指示に従う必要はありません。

- 2002 年 6 月 14 日より前に注文された PRPQ の場合は、古いバージョンの iSeries Tools for Developers PRPQ のインストールを参照してこのタスクを完了してください。

## 古いバージョンの *iSeries Tools for Developers* のインストール:

2002 年 6 月 14 日より前に注文された *iSeries Tools for Developers* PRPQ (5799PTL) のバージョンには、プリコンパイルされた *i5/OS PASE* 対応バージョンの *Virtual Network Computing (VNC)* が組み込まれていません。

以下の説明に従って、拡張 PRPQ がインストールされているかどうか判別し、古いバージョンの PRPQ がインストールされている場合には VNC をインストールしてください。

**拡張 PRPQ がインストールされているかどうか判別します。**

PRPQ 5799-PTL はインストールされているが、VNC を含む拡張バージョンがインストールされているかどうか分からない場合は、以下のファイルが存在しているかどうか調べます。

```
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
```

拡張バージョンの PRPQ には `vncserver_java` ファイルが組み込まれていますが、古いバージョンには組み込まれていません。`vncserver_java` が *iSeries* サーバー上に存在していない場合は、最新バージョンの PRPQ を注文してインストールするか、あるいは以下の指示に従って VNC のインストールを行うことができます。

## VNC のインストール

古いバージョンの *iSeries Tools for Developers* PRPQ に VNC をインストールするには、以下のステップを実行します。

1. 以下のコマンドを実行し、*iSeries* サーバー上で保管ファイルを作成します。

```
crtlib vncsavf
crtsavf vncsavf/vncpasswd
crtsavf vncsavf/vnc
crtsavf vncsavf/fonts
crtsavf vncsavf/icewm
```

2. ワークステーションへの保管ファイルのダウンロードは、*iSeries Virtual Innovation Center Web* サイトから実行してください。
3. ワークステーションで以下のコマンドを実行することにより、FTP を使用して、ワークステーションから *iSeries* サーバーに保管ファイルを転送します。

```
ftp youriserieserver
bin
cd /qsys.lib/vncsavf.lib
put vnc.savf
put vncpasswd.savf
put fonts.savf
put icewm.savf
quit
```

4. *iSeries* サーバー上で以下のコマンドを実行して、保管ファイルを復元します。

```
RSTOBJ OBJ(*ALL) SAVLIB(VNCSAVF) DEV(*SAVF) SAVF(VNCSAVF/VNCPASSWD)
RST DEV('/Qsys.lib/vncsavf.lib/vnc.file') OBJ('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc*')
RST DEV('/Qsys.lib/vncsavf.lib/fonts.file') OBJ('/QOpenSys/QIBM/ProdData/DeveloperTools/fonts*')
RST DEV('/Qsys.lib/vncsavf.lib/icewm.file') OBJ('/QOpenSys/QIBM/ProdData/DeveloperTools/icewm*')
```

5. NAWT のインストールを続行します。

## *Virtual Network Computing* サーバーの開始:

*Virtual Network Computing (VNC)* サーバーを開始するには、コマンド行で以下のコマンドを入力し、**ENTER** を押します。

```
CALL PGM(QSYS/QP2SHELL) PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
```

*n* は使用するディスプレイ番号です。ディスプレイ番号は、1 から 99 の範囲の任意の整数にすることができます。

注: VNC サーバーを開始すると、iSeries システム名およびディスプレイ番号を示すメッセージ ("New 'X'desktop is *systemname:1*." など) が表示されます。ディスプレイ番号は、環境変数を構成するために使用するの、覚えておくか、あるいは書き留めておいてください。

同時に複数の VNC サーバーを実行する場合は、各 VNC サーバーごとに固有なディスプレイ番号が必要になります。VNC サーバーの開始時に明示的にディスプレイ値を指定すれば、後から DISPLAY 環境変数を構成するのが容易になります。環境変数は NAWT を使用して Java を実行するたびに構成する必要があります。

しかし、ディスプレイ番号を指定したくない場合は、上記のコマンドから ':n' を除去すれば、vncserver\_java プログラムが使用可能なディスプレイを見付けます。

### .Xauthority ファイル

VNC サーバーの開始プロセスでは、新規の .Xauthority ファイルが作成されるか、あるいは既存の .Xauthority ファイルが変更されます。X サーバー権限は、暗号化されたキー情報を含む .Xauthority ファイルを使用して、他のユーザーのアプリケーションが X サーバー要求を傍受することがないようにします。Java 仮想マシン (JVM) と VNC の間のセキュア通信のためには、JVM と VNC の両方が .Xauthority ファイル内の暗号化されたキー情報にアクセスできなければなりません。

.Xauthority ファイルは VNC を開始したプロファイルに属します。JVM と VNC の両方が共に .Xauthority ファイルにアクセスできるようにするための最も容易な方法は、同一のユーザー・プロファイルで VNC サーバーと JVM を実行することです。VNC サーバーと JVM の両方を同一のユーザー・プロファイルで実行できない場合は、XAUTHORITY 環境変数を構成して正しい .Xauthority ファイルを指し示すことができます。NAWT でのセキュア通信について詳しくは、以下のページを参照してください。

- 『NAWT 環境変数の構成』
- NAWT を WebSphere Application Server と共に使用するためのヒント

### NAWT 環境変数の構成:

NAWT を使用して Java を実行するときは、システム名、ディスプレイ番号、および各 X サーバーとそれに関連した .Xauthority ファイルの場所を Java に示す環境変数が設定されていなければなりません。

注: Virtual Network Computing (VNC) サーバーを開始すると、.Xauthority ファイルの場所とシステム名およびディスプレイ番号の値が決まります。これらの値を使用して NAWT 環境変数を正常に構成する必要があります。詳しくは、Virtual Network Computing サーバーの開始を参照してください。

## DISPLAY の構成

Java プログラムを実行するセッションで、DISPLAY 環境変数をシステム名およびディスプレイ番号に設定します。DISPLAY 環境変数を構成するには、i5/OS コマンド行で以下の制御言語 (CL) コマンドを入力し、**ENTER** を押します。

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
```

*systemname* は iSeries システムのホスト名または IP アドレスであり、*n* は VNC サーバーのディスプレイ番号です。



## XAUTHORITY の構成

また、XAUTHORITY 環境変数を `/home/VNCprofile/.Xauthority` (`VNCprofile` は VNC を開始したプロファイル) に設定します。

たとえば、iSeries コマンド・プロンプトで以下のコマンドを入力します。

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/VNCprofile/.Xauthority')
```

ここで、

- `systemname` は iSeries システムのホスト名または IP アドレスです。
- `n` は VNC サーバーのディスプレイ番号です。

注:

- これらの変数は、Java 仮想マシン (JVM) を実行する環境でのみ設定する必要があります。
- 同一のユーザー・プロファイルで VNC サーバーと Java 仮想マシンを実行することができない場合は、XAUTHORITY を構成して適切な `.Xauthority` ファイルを指し示すことができます。JVM を実行する他のプロファイルが、`.Xauthority` ファイルにアクセスできることを確認する必要があります。詳しくは、`.Xauthority` ファイルを参照してください。
- XAUTHORITY のセキュリティー・メカニズムは、VNCPASSWD のセキュリティー・メカニズムとは異なります。セキュア・グラフィックス・レンダリングのためには、両方の保護体系が必要です。VNCPASSWD について詳しくは、VNC パスワード・ファイルの作成を参照してください。

### Java システム・プロパティーの構成:

Native Abstract Windowing Toolkit (NAWT) を実行する際は、Java を実行する前に毎回特定の Java システム・プロパティーを設定しなければなりません。以下の各例において、最初の行は望ましい Java 2 Software Development Kit (J2SDK) 用の Java を構成し、2 番目の行は NAWT を使用可能にします。

#### J2SDK バージョン 1.3

J2SDK バージョン 1.3 で NAWT を実行する場合は、以下の Java システム・プロパティーを設定します。

```
java.version=1.3
os400.awt.native=true
```

#### J2SDK バージョン 1.4 の完全 GUI サポート

J2SDK バージョン 1.4 で完全な GUI サポートのある NAWT を実行する場合は、以下の Java システム・プロパティーを設定します。

```
java.version=1.4
os400.awt.native=true
```

#### J2SDK バージョン 1.4 のヘッドレス AWT サポート

J2SDK バージョン 1.4 でヘッドレス・モードの NAWT を実行する場合は、以下の Java システム・プロパティーを設定します。

```
java.version=1.4
java.awt.headless=true
```

#### I J2SDK バージョン 1.5 の完全 GUI サポート



J2SDK バージョン 1.5 で完全な GUI をフル・サポートする NAWT を実行する場合は、以下の Java システム・プロパティを設定します。

```
java.version=1.5
os400.awt.native=true
```

## I J2SDK バージョン 1.5 のヘッドレス AWT サポート

J2SDK バージョン 1.5 でヘッドレス・モードの NAWT を実行する場合は、以下の Java システム・プロパティを設定します。

```
java.version=1.5
java.awt.headless=true
```

Java システム・プロパティの設定について詳しくは、IBM Developer Kit for Java 用に iSeries サーバをカスタマイズするを参照してください。

### NAWT のインストールの検査:

NAWT のインストールとセットアップの手順を完了したら、Java テスト・プログラムを実行して NAWT のインストールを検査することができます。i5/OS コマンド行からテスト・プログラムを実行するには、以下のコマンドを入力し、ENTER を押します。

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400') PROP((os400.awt.native true))
```

テスト・プログラムは JPEG でコード化されたイメージを作成し、それを統合ファイル・システム内の以下のパスに保管します。

```
/tmp/NAWTtest.jpg
```

テスト・プログラムを実行したら、テスト・プログラムによってこのファイルが作成されており、Java 例外が生成されていないことを確認してください。イメージを表示する場合は、バイナリー・モードを使用して、グラフィックスを処理できるシステムにイメージ・ファイルをアップロードします。

### iceWM ウィンドウ・マネージャの構成:

Virtual Network Computing (VNC) サーバを対話式に使用したい場合は、NAWT をセットアップする際のオプション・ステップとして、iceWM ウィンドウ・マネージャ (iceWM) を構成してください。これには、たとえば、グラフィカル・ユーザ・インターフェース (GUI) を備えた Java アプリケーションを実行したい場合などがあります。

iceWM は、iSeries Tools For Developers PRPQ に組み込まれている、小さいけれども強力なウィンドウ・マネージャです。PRPQ のインストール方法について詳しくは、以下のページを参照してください。

#### iSeries Tools for Developers PRPQ のインストール

iceWM はバックグラウンドで実行して、VNC サーバの X Window 環境内で実行しているウィンドウの外観を制御します。iceWM は、多くのポピュラーなウィンドウ・マネージャに似たインターフェースおよびフィーチャーのセットを提供します。組み込みの vncserver\_java スクリプトのデフォルト動作では、VNC サーバが開始され、iceWM が実行されます。

このステップを完了すると、iceWM が必要とするいくつかの構成ファイルが作成されます。iceWM は必要に応じて使用不可にすることもできます。

### iceWM の構成

iceWM ウィンドウ・マネージャーを構成するには、i5/OS コマンド・プロンプトで以下のステップを実行してください。これらのステップは、必ず VNC サーバーを開始するために使用するプロファイルで実行してください。

1. 以下のコマンドを入力し、**ENTER** を押してインストールを開始します。 :

```
STRPTL CLIENT(IGNORE)
```

IGNORE 値は、NAWT が必要とする STRPTL の構成フィーチャーのみをコマンドが活動化するようにするためのプレースホルダーとして機能します。

2. 以下のコマンドを入力し、**ENTER** を押してサインオフします。

```
SIGNOFF
```

サインオフすると、STRPTL コマンドのセッション固有の結果が、NAWT を使用または構成するためにその後実行する操作に影響しなくなります。

**注:** STRPTL コマンドは、VNC サーバーを開始する各プロファイルごとに 1 回のみ実行してください。NAWT では、コマンドの使用可能なオプションの引数はいずれも必須ではありません。この記述は、5799-PTL iSeries Tools For Developers PRPQ に関連した STRPTL のセットアップ手順に優先します。

### iceWM を使用不可にする

VNC サーバーを開始すると、iceWM を実行するためのコマンドが含まれた、xstartup\_java というスクリプト・ファイルが作成されるか、あるいはこの名前の既存のファイルが変更されます。xstartup\_java スクリプト・ファイルは、以下の統合ファイル・システム・ディレクトリーにあります。

```
/home/VNCprofile/.vnc/
```

VNCprofile は VNC サーバーを開始したプロファイルの名前です。

iceWM を完全に使用不可にする場合は、テキスト・エディターを使用して、iceWM を開始するスクリプトの中の行をコメント化するか、あるいは除去してください。行をコメント化するには、行の先頭にポンド記号 (#) を挿入します。

### VNCviewer または Web ブラウザーを使用する:

iSeries サーバーでグラフィカル・ユーザー・インターフェース (GUI) を備えたアプリケーションを実行する場合は、VNCviewer または Web ブラウザーを使用して Virtual Network Computing (VNC) サーバーに接続しなければなりません。もちろん、VNCviewer または Web ブラウザーは、パーソナル・コンピューターなどの、グラフィックスを処理できるプラットフォームで実行する必要があります。

**注:** 以下のステップを実行するには、ディスプレイ番号と VNC パスワードが分かっている必要があります。Virtual Network Computing (VNC) サーバーを開始すると、ディスプレイ番号の値が決まります。VNC パスワード・ファイルを作成すると、VNC パスワードが設定されます。詳しくは、以下のページを参照してください。

- Virtual Network Computing サーバーの開始
- VNC パスワード・ファイルの作成

### VNCviewer を使用した VNC サーバーへのアクセス

VNCviewer を使用して VNC サーバーに接続するには、以下のステップを実行します。

1. VNCviewer アプリケーションをダウンロードしてインストールします。

- ほとんどのプラットフォーム用の VNCviewer は、AT&T Research VNC Web サイトから入手できます。
2. ダウンロードした VNCviewer を開始します。プロンプトで、システム名とディスプレイ番号を入力し、「OK」をクリックします。
  3. パスワード・プロンプトで、VNC パスワードを入力して VNC サーバー・ディスプレイにアクセスします。

## Web ブラウザーを使用した VNC サーバーへのアクセス

Web ブラウザー を使用して VNC サーバーに接続するには、以下のステップを実行します。

1. ブラウザーを開始し、以下の URL にアクセスします。

```
http://systemname:58nn
```

ここで、各パラメーターは次のように定義されます。

- *systemname* は、VNC サーバーを実行しているシステムの名前または IP アドレスです。
- *nn* は 2 桁表示の VNC サーバー・ディスプレイ番号です。

たとえば、システム名が *system\_one* で、ディスプレイ番号が 2 の場合、URL は次のようになります。

```
http://system_one:5802
```

2. URL に正常にアクセスすると、VNC サーバー・パスワードを求めるプロンプトが表示されます。パスワード・プロンプトで、VNC パスワードを入力して VNC サーバー・ディスプレイにアクセスします。

## J2SDK バージョン 1.4 および後続バージョンでの完全な GUI サポートのある NAWT のインストールおよび使用:

Java 2 Software Development Kit (J2SDK) バージョン 1.4 およびそれ以降で完全な GUI サポートのある Native Abstract Windowing Toolkit (NAWT) をインストールするには、以下のタスクを完了してください。

1. NAWT ソフトウェア修正のインストール
2. iSeries Tools for Developers PRPQ のインストール

## NAWT を使用する

NAWT を使用したり、NAWT のインストールをテストしたりするには、まず Virtual Network Computing (VNC) サーバー用のパスワード・ファイルを作成する必要があります。以下に、追加の必須およびオプションのステップをリストします。

VNC パスワード・ファイルの作成

VNC サーバーの開始 (通常は各 IPL の後)

環境変数の構成 (Java の実行前に毎回)

Java システム・プロパティの構成 (Java の実行前に毎回)

iceWM ウィンドウ・マネージャーの構成 (オプション - 対話式に使用する場合)

VNCviewer または Web ブラウザーを使用した GUI アプリケーションとの対話

NAWT インストールの検査 (オプション)

**J2SDK バージョン 1.4 および後続バージョンでのヘッドレス AWT モードの NAWT のインストールおよび使用:** Java 2 Software Development Kit (J2SDK) バージョン 1.4 および後続のバージョンで、ヘッドレス AWT モードの Native Abstract Windowing Toolkit (NAWT) をインストールするには、以下のタスクを完了してください。

NAWT ソフトウェア修正のインストール

## NAWT を使用する

以下に、NAWT を使用したり、NAWT のインストールをテストしたりする前に実行する必要のある、追加の必須およびオプションのステップをリストします。

Java システム・プロパティの構成 (Java の実行前に毎回)

NAWT インストールの検査 (オプション)

### リンク集

NAWT ソフトウェア修正のインストール

NAWT をインストールする場合は、使用する Java 2 Software Development Kit (J2SDK) Standard Edition のバージョン用の適切な NAWT サポートが組み込まれたソフトウェア修正をインストールする必要があります。

Java システム・プロパティの構成

NAWT インストールの検査

## Native Abstract Windowing Toolkit のインストールおよび使用

NAWT および VNC をインストールするための説明を参照してください。NAWT を使用する前に、いくつかの必須ステップを完了する必要があります。

詳しくは、NAWT サポートのレベルを参照してください。

## NAWT のインストールおよび使用

グラフィカル要件を見積もって、実行する J2SDK のバージョンを決定したら、以下の説明に従って NAWT をインストールして使用してください。

263 ページの『J2SDK バージョン 1.3 での NAWT のインストールおよび使用』

270 ページの『J2SDK バージョン 1.4 および後続バージョンでの完全な GUI サポートのある NAWT のインストールおよび使用』

『J2SDK バージョン 1.4 および後続バージョンでのヘッドレス AWT モードの NAWT のインストールおよび使用』

## NAWT および i5/OS PASE

NAWT は i5/OS PASE 環境を自動的に開始しますが、デフォルトでは 32 ビット・モードで開始します。i5/OS PASE を 64 ビット・モードで実行する必要がある場合は、JVM を開始する前に QIBM\_JAVA\_PASE\_STARTUP 環境変数を設定する必要があります。詳細については、Java i5/OS PASE 環境変数を参照してください。

## VNC の使用上のヒント

i5/OS 制御言語 (CL) コマンドを使用して Virtual Network Computing (VNC) サーバーを開始および停止し、現行で実行している VNC サーバーに関する情報を表示します。

### CL プログラムからの VNC ディスプレイ・サーバーの開始

以下の例は、DISPLAY 環境変数を設定し、制御言語 (CL) コマンドを使用して自動的に VNC を開始するための 1 つの方法を示しています。

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
```

ここで、各パラメーターは次のように定義されます。

- *systemname* は VNC が稼働している iSeries システムのホスト名または IP アドレスです。
- ここで、*n* は開始したいディスプレイ番号を表す数値です。

**注:** この例では、まだディスプレイ *n* を実行しておらず、必要な VNC パスワードを正常に作成してあることを想定しています。パスワード・ファイルの作成について詳しくは、VNC パスワード・ファイルの作成を参照してください。

### CL プログラムからの VNC ディスプレイ・サーバーの停止

以下のコードは、CL プログラムから VNC サーバーを停止するための 1 つの方法を表しています。

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' '-kill' ':n')
```

*n* は、終了するディスプレイ番号を表す数値です。

### 実行中の VNC ディスプレイ・サーバーの検査

現行でどの (存在する場合) VNC サーバーが iSeries システム上で実行しているのかを判別するには、以下のステップを実行してください。

1. i5/OS コマンド行から PASE シェルを開始します。

```
CALL QP2TERM
```

2. PASE シェル・プロンプトから、PASE ps コマンドを使用して VNC サーバーをリストします。

```
ps gaxuw | grep Xvnc
```

このコマンドの結果の出力では、実行中の VNC サーバーが以下のフォーマットで表示されます。

```
john 418 0.9 0.0 5020 0 - A Jan 31 222:26
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/Xvnc :1 -desktop X -httpd
jane 96 0.2 0.0 384 0 - A Jan 30 83:54
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/Xvnc :2 -desktop X -httpd
```

ここで、

- 最初の列は、サーバーを開始したプロファイルです。
- 2 番目の列はサーバーの PASE プロセス ID です。
- */QOpensys/* で始まる情報は、VNC サーバーを開始したコマンド (引数を含む) です。通常ディスプレイ番号は、Xvnc コマンドの引数リストの中の最初の項目です。

注: 上記の出力例で表されている Xvnc プロセスは、実際の VNC サーバー・プログラムの名前です。vncserver\_java スクリプトを実行する際は Xvnc を開始してください。これは Xvnc のための環境とパラメーターを準備してから、Xvnc を開始します。

### VNC パスワード・ファイルの作成:

Native Abstract Windowing Toolkit (NAWT) をインストールして実行するには、Virtual Network Computing (VNC) サーバーのパスワード・ファイルを作成する必要があります。VNC サーバーのデフォルトの設定においては、無許可ユーザーのアクセスから VNC の表示を保護するためにこれが使用するパスワード・ファイルが必要です。VNC パスワード・ファイルは、VNC サーバーを開始するプロファイルの下に作成する必要があります。

暗号化パスワードの作成方法は、使用している PRPQ のバージョンによって次のように異なります。

- 2002 年 6 月 14 日以降に注文された PRPQ のバージョンの場合は、iSeries コマンド・プロンプトで以下のコマンドを使用します。

```
MKDIR DIR('/home/VNCprofile/.vnc')
QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/VNCprofile/.vnc/passwd')
```

VNCprofile は VNC サーバーを開始したプロファイルです。

- 2002 年 6 月 14 日より前に注文された PRPQ の場合は、iSeries コマンド・プロンプトで以下のコマンドを使用します。

```
MKDIR DIR('/home/VNCprofile/.vnc')
VNCSAVF/VNCPASSWD USEHOME(*NO) PWDFILE('/home/VNCprofile/.vnc/passwd')
```

VNCprofile は VNC サーバーを開始したプロファイルです。

注:

### 任意のバージョンの J2SDK と共に NAWT を使用する場合

- VNC パスワード・ファイルを持っている必要があるのは、VNC サーバーを開始するプロファイルだけです。
- VNC サーバーを正常に開始するには、パスワード・ファイルを持っていないけません。

### J2SDK バージョン 1.4 およびそれ以降のバージョンと共に NAWT を使用する場合

- VNCviewer または Web ブラウザーを使用して VNC サーバーに対話式にアクセスするには、このステップで指定したパスワードを使用する必要があります。

### NAWT を WebSphere Application Server と共に使用するためのヒント

WebSphere Application Server の下で実行するグラフィカル Java プログラムで使用するために NAWT をセットアップします。WebSphere Application Server と NAWT を使用する場合は、Virtual Network Computing (VNC) サーバーと WebSphere Application Server との間のセキュア通信を可能にしなければなりません。

以下の情報を読む前に、必ず iSeries サーバーでの Native Abstract Windowing Toolkit (NAWT) のインストールおよび使用方法を理解しておいてください。特に、ご使用の Java 2 Software Development Kit (J2SDK) のバージョンと i5/OS のリリースでの NAWT の使用方法について知っておく必要があります。



## セキュア通信の確保

X 権限検査というメソッドが、WebSphere Application Server と VNC サーバーの間のセキュア通信を確保します。

VNC サーバーの開始プロセスでは、暗号化されたキー情報を含む .Xauthority ファイルが作成されます。WebSphere Application Server と VNC の間のセキュア通信のためには、WebSphere Application Server と VNC の両方が .Xauthority ファイル内の暗号化されたキー情報にアクセスできなければなりません。

## X 権限検査の使用

X 権限検査は以下のいずれかの方法で使用します。

### 同一のプロファイルを使用して WebSphere Application Server と VNC を実行する

WebSphere Application Server と VNC サーバーの間のセキュア通信を確実にするための 1 つの方法は、VNC サーバーを開始するために使用したものと同一のプロファイルから WebSphere Application Server を実行することです。WebSphere Application Server と VNC を同一のプロファイルを使用して実行する場合は、アプリケーション・サーバーが実行するユーザー・プロファイルを変更しなければなりません。

アプリケーション・サーバーのユーザー・プロファイルを、デフォルトのユーザー (QEJBSVR) から別のプロファイルに切り替えるには、以下の操作を行う必要があります。

1. WebSphere Application Server 管理コンソールを使用して、アプリケーション・サーバーの構成を変更します。
2. iSeries ナビゲーターを使用して新しいプロファイルを使用可能にします。

WebSphere Application Server 管理コンソールと iSeries ナビゲーターの使用法については、以下の資料を参照してください。

WebSphere Application Server

マネージメント・セントラルによるユーザーおよびグループの管理

### 別のプロファイルを使用して WebSphere Application Server と VNC を実行する

WebSphere Application Server と VNC で別々のプロファイルを使用したい場合は、WebSphere Application Server に .Xauthority ファイルを使用させることによって、セキュア通信を確保することができます。

WebSphere Application Server が .Xauthority ファイルを使用できるようにするには、以下のステップを実行してください。

1. ユーザー・プロファイルから VNC サーバーを開始することによって、新規の .Xauthority ファイルを作成 (または既存の .Xauthority ファイルを更新) します。i5/OS 制御言語 (CL) コマンド行から、以下のコマンドを入力して **ENTER** を押します。

```
CALL QP2SHELL PARM('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
```

*n* はディスプレイ番号 (1 から 99 の範囲の数値) です。

注: .Xauthority ファイルは、VNC サーバーを実行しているプロファイルのディレクトリーにあります。

2. 以下の CL コマンドを使用して、WebSphere Application Server を実行しているプロファイルに対し、.Xauthority ファイルの読み取り権限を付与します。

```
CHGAUT OBJ('/home') USER(WASprofile) DTAUT(*RX)
CHGAUT OBJ('/home/VNCprofile') USER(WASprofile) DTAUT(*RX)
CHGAUT OBJ('/home/VNCprofile/.Xauthority') USER(WASprofile) DTAUT(*R)
```

VNCprofile および WASprofile は、VNC サーバーと WebSphere Application Server を実行している該当のプロファイルです。

注: これらのステップに従う必要があるのは、VNCprofile と WASprofile とが異なるプロファイルである場合だけです。VNCprofile と WASprofile とが同じプロファイルの場合にこれらのステップに従うと、VNC が正しく機能しない原因となる場合があります。

3. WebSphere Application Server 管理コンソールから、アプリケーション用の DISPLAY および XAUTHORITY 環境変数を定義します。

- DISPLAY に対しては、system:n または localhost:n を使用してください。

system は iSeries システムのホスト名または IP アドレスであり、n は VNC サーバーを開始するために使用したディスプレイ番号です。

- XAUTHORITY に対しては、/home/VNCprofile/.Xauthority を使用してください。

VNCprofile は VNC サーバーを開始したプロファイルです。

4. WebSphere Application Server を再始動して、構成の変更をピックアップします。

WebSphere Application Server 管理コンソールの使用方法について詳しくは、以下の資料を参照してください。

WebSphere Application Server

---

## Java セキュリティー

このトピックでは、借用権限について詳述し、SSL を使用して Java アプリケーション中のソケット・ストリームを保護する方法を説明します。

Java アプリケーションは iSeries サーバー上の他のすべてのプログラムと同じセキュリティ上の制限を受けます。Java プログラムを iSeries サーバー上で実行するには、統合ファイル・システム内のクラス・ファイルに対する権限が必要です。プログラムが開始されると、それはユーザーの権限の下で実行されます。

借用権限を使用して、プログラムを実行しているユーザーの権限およびプログラム所有者の権限を持つオブジェクトにアクセスできます。借用権限は、ユーザーが当初はアクセス権を持っていなかったオブジェクトに対する権限を、一時的にユーザーに付与します。2 つの新しい借用権限パラメーター (USRPRF および USEADPAUT) に関する詳細は、「Java プログラムの作成 (CRTJVAPGM)」コマンド情報を参照してください。

iSeries サーバー上で実行する Java プログラムのほとんどはアプレットではなくアプリケーションなので、"sandbox" セキュリティー・モデルによる制限を受けません。

注: J2SDK バージョン 1.4 および後続のリリースの場合、JAAS、JCE、JGSS、および JSSE は基本 JDK の一部で、拡張とは見なされません。旧バージョンの JDK の場合、これらのセキュリティ項目は拡張機能です。

## Java セキュリティ・モデル

Java アプレットはどのシステムからでもダウンロードできます。そのため、悪質なアプレットから保護するためのセキュリティ機構が Java 仮想マシンに組み込まれています。Java ランタイム・システムは、Java 仮想マシンがバイトコードをロードするときにそれを検査します。これにより、それらが適正なバイトコードであること、および Java 仮想マシンが Java アプレットに課しているどの制限にも違反しないことが確認されます。

アプレットの場合と同じく、バイトコード・ローダーおよび検査装置はバイトコードが有効であるか、およびデータ・タイプが適切に使用されているかどうかを検査します。それらはさらに、レジスターおよびメモリーが正しくアクセスされているか、およびスタックがオーバーフローするまたはアンダーフローしていないかどうかを検査します。これらの検査によって、Java 仮想マシンがシステムの保全性を妨げることなくクラスを実行できることが保証されます。

Java アプレットは、実行可能な操作、メモリーへのアクセス方法、および Java 仮想マシンを使用する方法に関して制限を受けます。その制限は、Java アプレットが基礎となるオペレーティング・システムまたはシステム上のデータにアクセスすることを防ぎます。これは、“sandbox” セキュリティ・モデルと呼ばれます。Java アプレットが自分のサンドボックス (砂箱) 内でのみ「遊ぶ」ことができるからです。

“sandbox” セキュリティ・モデルは、クラス・ローダー、クラス・ファイル・ペリファイヤー、および `java.lang.SecurityManager` クラスの組み合わせで実現されています。

セキュリティの詳細については、Security by Sun Microsystems, Inc. 資料と、SSL によるアプリケーションの保護 を参照してください。

## Java Cryptography Extension

Java Cryptography Extension (JCE) 1.2 は、Java 2 Software Development Kit (J2SDK), Standard Edition の標準拡張機能です。iSeries サーバー上の JCE 実装は、Sun Microsystems, Inc. の実装と互換性があります。この資料では、iSeries 実装に固有の側面について扱います。

この情報を理解するには、JCE 拡張機能の一般資料に精通している必要があります。JCE 拡張機能の詳細については、Sun JCE の資料 (英語) を参照してください。

1 IBM JCE プロバイダーは、以下のアルゴリズムをサポートしています。

表 3. JDK 1.3 および JDK 1.4.2 でサポートされるアルゴリズム

| JDK バージョン | 署名アルゴリズム   | 暗号アルゴリズム   |
|-----------|--|--|
| 1.3       | SHA1withDSA<br>SHA1withRSA<br>MD5withRSA<br>MD2withRSA | Blowfish<br>AES<br>DES<br>Triple DES<br>PBEWithMD2AndDES<br>PBEWithMD2AndTripleDES<br>PBEWithMD2AndRC2<br>PBEWithMD5AndDES<br>PBEWithMD5AndTripleDES<br>PBEWithMD5AndRC2<br>PBEWithSHA1AndDES<br>PBEWithSHA1AndTripleDES<br>PBEWithSHA1AndRC2<br>PBEWithSHAAnd40BitRC2<br>PBEWithSHAAnd128BitRC2<br>PBEWithSHAAnd40BitRC4<br>PBEWithSHAAnd128BitRC4<br>PBEWithSHAAnd2KeyTripleDES<br>PBEWithSHAAnd3KeyTripleDES<br>Mars<br>RC2<br>RC4<br>RSA<br>Seal |

表3. JDK 1.3 および JDK 1.4.2 でサポートされるアルゴリズム (続き)

| JDK バージョン | 署名アルゴリズム   | 暗号アルゴリズム   |
|-----------|--|--|
| 1.4.2     | SHA1withDSA<br>SHA1withRSA<br>MD5withRSA<br>MD2withRSA | Blowfish<br>AES<br>DES<br>Triple DES<br>PBEWithMD2AndDES<br>PBEWithMD2AndTripleDES<br>PBEWithMD2AndRC2<br>PBEWithMD5AndDES<br>PBEWithMD5AndTripleDES<br>PBEWithMD5AndRC2<br>PBEWithSHA1AndDES<br>PBEWithSHA1AndTripleDES<br>PBEWithSHA1AndRC2<br>PBEWithSHAAnd40BitRC2<br>PBEWithSHAAnd128BitRC2<br>PBEWithSHAAnd40BitRC4<br>PBEWithSHAAnd128BitRC4<br>PBEWithSHAAnd2KeyTripleDES<br>PBEWithSHAAnd3KeyTripleDES<br>Mars<br>RC2<br>RC4<br>RSA<br>Seal |

表4. JDK 1.3 および JDK 1.4.2 でサポートされるアルゴリズム (続き)

| JDK バージョン | メッセージ確認コード (MAC)               | メッセージ要約  | 鍵合意アルゴリズム     |
|-----------|--------------------------------|--|---------------|
| 1.3       | HmacSHA1<br>HmacMD2<br>HmacMD5 | MD2<br>MD5<br>SHA-1                                  | DiffieHellman |
| 1.4.2     | HmacSHA1<br>HmacMD2<br>HmacMD5 | MD2<br>MD5<br>SHA-1<br>SHA-256<br>SHA-384<br>SHA-512 | DiffieHellman |

さらに、IBM JCE プロバイダーは乱数発生ルーチンも備えています。

IBM JCE で Java 1.3 を使用する場合は、/QIBM/ProdData/OS400/Java400/jdk/lib/security/java.security ファイルを編集します。変更が必要なファイル中のセクションは、以下のとおりです。

```
#
# To use the IBMJCE security provider, you need to:
# 1) Install an IBM Cryptographic Access Provider Product
# 2) Uncomment the third provider entry that follows.
#
# List of providers and their preference orders:
```

```
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
#security.provider.3=com.ibm.crypto.provider.IBMJCE
```

IBMJCEFIPS JCE プロバイダーもあります。このプロバイダーは検証済みであり、連邦情報処理標準 (FIPS) 140-2 の『Security Requirements for Cryptographic Modules』に準拠していることが確認されています。

IBMJCEFIPS JCE プロバイダーは、以下のアルゴリズムをサポートしています。

表 5. IBMJCEFIPS JCE プロバイダーによりサポートされるアルゴリズム

| 署名アルゴリズム                   | 暗号アルゴリズム                | メッセージ確認コード | メッセージ要約                                       |
|----------------------------|-------------------------|------------|---|
| SHA1withDSA<br>SHA1withRSA | AES<br>TripleDES<br>RSA | HmacSHA1   | MD5<br>SHA-1<br>SHA-256<br>SHA-384<br>SHA-512 |

IBMJCEFIPS JCE プロバイダーは、乱数生成用の IBMSecureRandom アルゴリズムもサポートしています。

IBMJCEFIPS を使用するには、以下のコマンドを発行して、拡張ディレクトリーにシンボリック・リンクを追加する必要があります。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjcefps.jar')
NEWLNK(< your extension directory >)
```

さらに、プロバイダーをプロバイダーのリストに追加する必要がありますが、これは項目を `java.security` ファイルに追加するか (例: `security.provider.4=com.ibm.crypto.fips.provider.IBMJCEFIPS`)、または `Security.addProvider()` メソッドを使用して実行します。

## Java Secure Socket Extension

Java Secure Socket Extension (JSSE) は、Secure Sockets Layer (SSL) プロトコルの Java 実装です。JSSE は、SSL と Transport Layer Security (TLS) プロトコルを使用して、クライアントとサーバーが TCP/IP を介して安全な通信を行えるようにします。

JSSE は以下の機能を提供します。

- データを暗号化する
- リモート・ユーザー ID を認証する
- リモート・システム名を認証する
- クライアント/サーバー認証を実行する
- メッセージ保全性を保持する

Java 2 Software Development Kit Standard Edition (J2SDK) バージョン 1.4 および後続のリリースに統合され、JSSE は SSL だけの場合より多くの機能性を提供します。

注: この情報は、現在では J2SDK バージョン 1.4 および後続のリリースに組み込まれて出荷されている JSSE のバージョンにのみ関係します。以前のバージョンの JSSE については、Sun Java Web サイトにある Java Secure Socket Extension を参照してください。



## SSL (JSSE バージョン 1.0.8) を使用する

SSL は、サーバーおよびクライアントを認証してプライバシーおよびデータ保全性を備えることを可能にします。すべての SSL 通信は、サーバーとクライアントとの間の「ハンドシェイク」から始まります。ハンドシェイクの際、SSL はクライアントとサーバーが互いに通信するために使用する暗号の組を取り交わします。この暗号の組は、SSL で使用可能な種々のセキュリティ機能の組み合わせです。SSL は J2SDK バージョン 1.3 でのみ使用可能です。Secure Sockets Layer (SSL) の Java 実装である Java Secure Socket Extension (JSSE バージョン 1.0.8) を使用して、Java アプリケーションのセキュリティを強化することができます。

SSL は、以下の方法によりアプリケーションのセキュリティを向上させます。

- 暗号化により通信データを保護する。
- リモート・ユーザー ID を認証する。
- リモート・システム名を認証する。

注: SSL では、デジタル証明書を使用して、Java アプリケーションのソケット通信を暗号化します。デジタル証明書は、保護システム、ユーザー、およびアプリケーションを識別するためのインターネット標準です。IBM デジタル証明書マネージャーを使用すると、デジタル証明書を制御できます。詳しくは、IBM デジタル認証マネージャーを参照してください。

SSL を使用して Java アプリケーションの保護を向上させるには、以下のようになります。

- iSeries サーバーで SSL をサポートできるように準備する。
- 以下のようにして、SSL を使用するように Java アプリケーションを設計する。
  - ソケット・ファクトリーをまだ使用していない場合は、ソケット・ファクトリーを使用するように Java ソケット・コードを変更する。
  - SSL を使用するように Java コードを変更する。
- 以下のようにして、デジタル証明書を使用して Java アプリケーションの保護を向上させる。
  1. 使用するデジタル証明書のタイプを選択します。
  2. アプリケーション実行時にデジタル証明書を使用します。

QsyRegisterAppForCertUse API を使用して、ご使用の Java アプリケーションを保護アプリケーションとして登録することもできます。詳しくは、QsyRegisterAppForCertUseを参照してください。

Java バージョンの SSL について詳しくは、Java Secure Socket Extension を参照してください。

### iSeries サーバーで Secure Sockets Layer をサポートできるように準備する:

システムで Secure Sockets Layer (SSL) を使用できるように準備するには、ライセンス・プログラム (デジタル証明書マネージャー の LP) をインストールする必要があります。

以下のデジタル証明書マネージャー の LP をインストールする必要があります。

- 5722-SS1 i5/OS - デジタル証明書マネージャー

また、システム上のデジタル証明書にアクセスできるか作成できることを確認する必要もあります。

iSeries デジタル証明書の管理とインターネットについて詳しくは、

DCM の計画を参照してください。

### ソケット・ファクトリーを使用するように Java コードを変更する:

既存のコードで Secure Sockets Layer (SSL) を使用するには、まず最初にソケット・ファクトリーを使用するようにコードを変更しなければなりません。

ソケット・ファクトリーを使用するようにコードを変更するには、以下のステップを実行します。

1. 以下の行をご使用のプログラムに追加して、SocketFactory クラスをインポートします。

```
import javax.net.*;
```

2. SocketFactory オブジェクトのインスタンスを宣言する行を追加します。以下に例を示します。

```
SocketFactory socketFactory
```

3. SocketFactory インスタンスを、メソッド SocketFactory.getDefault() と同等の値に設定して、初期設定します。以下に例を示します。

```
socketFactory = SocketFactory.getDefault();
```

SocketFactory の宣言全体は、以下のようになるはずですが、

```
SocketFactory socketFactory = SocketFactory.getDefault();
```

4. 既存のソケットを初期設定します。宣言するソケットごとに、ソケット・ファクトリー上の SocketFactory メソッド createSocket(host,port) を呼び出します。

この時点で、ソケットの宣言は以下のようになるはずですが、

```
Socket s = socketFactory.createSocket(host,port);
```

ここで、

- *s* は、作成するソケットです。
- *socketFactory* は、ステップ 2 で作成した SocketFactory です。
- *host* は、ホスト・サーバーの名前を表すストリング変数です。
- *port* は、ソケット接続のポート番号を表す整数変数です。

上記のステップをすべて完了すると、コードでソケット・ファクトリーが使用されます。コードにこれ以外の変更を加える必要はありません。呼び出されるメソッドとソケットの構文は、依然としてすべて稼働します。

ソケット・ファクトリーを使用するようにサーバー・プログラムを変換する例については、例: サーバーのソケット・ファクトリーを使用するように Java コードを変更するを参照してください。

ソケット・ファクトリーを使用するようにクライアント・プログラムを変換する例については、例: クライアントのソケット・ファクトリーを使用するように Java コードを変更するを参照してください。

### 例: サーバーのソケット・ファクトリーを使用するように Java コードを変更する:

以下の例は、simpleSocketServer という単純なソケット・クラスを変更し、ソケット・ファクトリーを使用してすべてのソケットを作成できるようにする方法を示しています。1 つ目の例は、ソケット・ファクトリーのない simpleSocketServer クラスを示しています。2 つ目の例は、ソケット・ファクトリーのある simpleSocketServer クラスを示しています。2 つ目の例では、simpleSocketServer が factorySocketServer に名前変更されています。

#### 例 1: ソケット・ファクトリーのないソケット・サーバー・プログラム

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

/* File simpleSocketServer.java*/

import java.net.*;
import java.io.*;

public class simpleSocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        ServerSocket serverSocket =
            new ServerSocket(serverPort);

        // a real server would handle more than just one client like this...

        Socket s = serverSocket.accept();
        BufferedInputStream is = new BufferedInputStream(s.getInputStream());
        BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it...

        byte buffer[] = new byte[4096];

        int bytesRead;

        // read until "eof" returned
        while ((bytesRead = is.read(buffer)) > 0) {
            os.write(buffer, 0, bytesRead); // write it back
            os.flush(); // flush the output buffer
        }

        s.close();
        serverSocket.close();
    } // end main()
} // end class definition

```

## 例 2: ソケット・ファクトリーのある単純なソケット・サーバー・プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/* File factorySocketServer.java */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
    }
}

```

```

else
    serverPort = new Integer(args[0]).intValue();

System.out.println("Establishing server socket at port " + serverPort);

// Change the original simpleSocketServer to use a
// ServerSocketFactory to create server sockets.
ServerSocketFactory serverSocketFactory =
    ServerSocketFactory.getDefault();
// Now have the factory create the server socket. This is the last
// change from the original program.
ServerSocket serverSocket =
    serverSocketFactory.createServerSocket(serverPort);

// a real server would handle more than just one client like this...

Socket s = serverSocket.accept();
BufferedInputStream is = new BufferedInputStream(s.getInputStream());
BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

// This server just echoes back what you send it...

byte buffer[] = new byte[4096];

int bytesRead;

while ((bytesRead = is.read(buffer)) > 0) {
    os.write(buffer, 0, bytesRead);
    os.flush();
}

s.close();
serverSocket.close();
}
}

```

背景情報については、ソケット・ファクトリーを使用するための Java コードの変更を参照してください。

#### **例: クライアントのソケット・ファクトリーを使用するように Java コードを変更する:**

以下の例は、`simpleSocketClient` という単純なソケット・クラスを変更し、ソケット・ファクトリーを使用してすべてのソケットを作成できるようにする方法を示しています。1 つ目の例は、ソケット・ファクトリーのない `simpleSocketClient` クラスを示しています。2 つ目の例は、ソケット・ファクトリーのある `simpleSocketClient` クラスを示しています。2 つ目の例では、`simpleSocketClient` が `factorySocketClient` に名前変更されています。

#### **例 1: ソケット・ファクトリーのないソケット・クライアント・プログラム**

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

/* Simple Socket Client Program */

import java.net.*;
import java.io.*;

public class simpleSocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {

```

```

        System.out.println("java simpleSocketClient serverHost serverPort");
        System.out.println("serverPort defaults to 3000 if not specified.");
        return;
    }
    if (args.length == 2)
        serverPort = new Integer(args[1]).intValue();

    System.out.println("Connecting to host " + args[0] + " at port " +
        serverPort);

    // Create the socket and connect to the server.
    Socket s = new Socket(args[0], serverPort);
    .
    .
    .

    // The rest of the program continues on from here.

```

## 例 2: ソケット・ファクトリーのある単純なソケット・クライアント・プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/* Simple Socket Factory Client Program */

// Notice that javax.net.* is imported to pick up the SocketFactory class.
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java factorySocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
            return;
        }
        if (args.length == 2)
            serverPort = new Integer(args[1]).intValue();

        System.out.println("Connecting to host " + args[0] + " at port " +
            serverPort);

        // Change the original simpleSocketClient program to create a
        // SocketFactory and then use the socket factory to create sockets.

        SocketFactory socketFactory = SocketFactory.getDefault();

        // Now the factory creates the socket. This is the last change
        // to the original simpleSocketClient program.

        Socket s = socketFactory.createSocket(args[0], serverPort);
        .
        .
        .

        // The rest of the program continues on from here.

```

背景情報については、ソケット・ファクトリーを使用するための Java コードの変更を参照してください。

**Secure Sockets Layer** を使用するように Java コードを変更する:

すでにコード中でソケット・ファクトリーを使用してソケットを作成している場合は、ご使用のプログラムに Secure Sockets Layer (SSL) サポートを追加できます。まだコード中でソケット・ファクトリーを使用していない場合は、ソケット・ファクトリーを使用するように Java コードを変更するを参照してください。

SSL を使用するようにコードを変更するには、以下のステップを実行します。

1. javax.net.ssl.\* をインポートして、SSL サポートを追加する。

```
import javax.net.ssl.*;
```

2. SSLSocketFactory を使用して SocketFactory を初期設定することにより、SocketFactory を宣言する。

```
SocketFactory newSF = SSLSocketFactory.getDefault();
```

3. 新しい SocketFactory を使用して、以前の SocketFactory の場合と同じ方法でソケットを初期設定する。

```
Socket s = newSF.createSocket(args[0], serverPort);
```

これで、コード中で SSL サポートが使用されるようになりました。コードにこれ以外の変更を加える必要はありません。

コードの例は、例: Secure Sockets Layer を使用するように Java クライアントを変更すると、例: Secure Sockets Layer を使用するように Java サーバーを変更するを参照してください。

#### **例: Secure Sockets Layer を使用するように Java サーバーを変更する:**

以下の例は、factorySocketServer という 1 つのクラスを変更して、Secure Sockets Layer (SSL) を使用できるようにする方法を示しています。

1 つ目の例は、SSL を使用しない factorySocketServer クラスを示しています。2 つ目の例は、同じクラスで SSL を使用するものを示しており、名前が factorySSLSocketServer に変更されています。

#### **例 1: SSL を使用しない単純な factorySocketServer クラス**

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
/* File factorySocketServer.java */
// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        // Change the original simpleSocketServer to use a
        // ServerSocketFactory to create server sockets.
        ServerSocketFactory serverSocketFactory =
            ServerSocketFactory.getDefault();
```



```

// Now have the factory create the server socket. This is the last
// change from the original program.
ServerSocket serverSocket =
    serverSocketFactory.createServerSocket(serverPort);

// a real server would handle more than just one client like this...

Socket s = serverSocket.accept();
BufferedInputStream is = new BufferedInputStream(s.getInputStream());
BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

// This server just echoes back what you send it.

byte buffer[] = new byte[4096];

int bytesRead;

while ((bytesRead = is.read(buffer)) > 0) {
    os.write(buffer, 0, bytesRead);
    os.flush();
}

s.close();
serverSocket.close();
}
}

```

## 例 2: SSL を使用する単純な factorySocketServer クラス

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/* File factorySocketServer.java */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        // Change the original simpleSocketServer to use a
        // ServerSocketFactory to create server sockets.
        ServerSocketFactory serverSocketFactory =
            ServerSocketFactory.getDefault();
        // Now have the factory create the server socket. This is the last
        // change from the original program.
        ServerSocket serverSocket =
            serverSocketFactory.createServerSocket(serverPort);

        // a real server would handle more than just one client like this...

        Socket s = serverSocket.accept();
        BufferedInputStream is = new BufferedInputStream(s.getInputStream());
        BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());
    }
}

```

```

// This server just echoes back what you send it.

byte buffer[] = new byte[4096];

int bytesRead;

while ((bytesRead = is.read(buffer)) > 0) {
    os.write(buffer, 0, bytesRead);
    os.flush();
}

s.close();
serverSocket.close();
}
}

```

背景情報については、Secure Sockets Layer を使用するように Java コードを変更するを参照してください。

**例: Secure Sockets Layer を使用するように Java クライアントを変更する:**

以下の例は、factorySocketClient という 1 つのクラスを変更して、Secure Sockets Layer (SSL) を使用できるようにする方法を示しています。1 つ目の例は、SSL を使用しない factorySocketClient クラスを示しています。2 つ目の例は、同じクラスで SSL を使用するものを示しており、名前が factorySSLSocketClient に変更されています。

**例 1: SSL を使用しない単純な factorySocketClient クラス**

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

/* Simple Socket Factory Client Program */

import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java factorySocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
            return;
        }
        if (args.length == 2)
            serverPort = new Integer(args[1]).intValue();

        System.out.println("Connecting to host " + args[0] + " at port " +
            serverPort);

        SocketFactory socketFactory = SocketFactory.getDefault();

        Socket s = socketFactory.createSocket(args[0], serverPort);
        .
        .
        .

        // The rest of the program continues on from here.
    }
}

```

**例 2: SSL を使用する単純な factorySocketClient クラス**

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Notice that we import javax.net.ssl.* to pick up SSL support
import javax.net.ssl.*;
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySSLSocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java factorySSLSocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
            return;
        }
        if (args.length == 2)
            serverPort = new Integer(args[1]).intValue();

        System.out.println("Connecting to host " + args[0] + " at port " +
            serverPort);

        // Change this to create an SSLSocketFactory instead of a SocketFactory.
        SocketFactory socketFactory = SSLSocketFactory.getDefault();

        // We do not need to change anything else.
        // That's the beauty of using factories!
        Socket s = socketFactory.createSocket(args[0], serverPort);
        .
        .
        .

        // The rest of the program continues on from here.
    }
}
```

背景情報については、Secure Sockets Layer を使用するように Java コードを変更するを参照してください。

### 使用するデジタル証明書の選択:

どのデジタル証明書を使用するか決める際には、複数の要素を考慮する必要があります。ご使用のシステムのデフォルト証明書を使用することもできますし、別の証明書を指定して使用することもできます。

以下の場合には、システムのデフォルト証明書を使用することもできます。

- ご使用の Java アプリケーションに特定のセキュリティー要件がない。
- ご使用の Java アプリケーションに必要なセキュリティーの種類が分からない。
- システムのデフォルト証明書が、ご使用の Java アプリケーションのセキュリティー要件を満たしている。

注: システムのデフォルト証明書を使用することに決めた場合は、システム管理者に問い合わせ、デフォルトのシステム証明書が作成されていることを確認してください。デジタル証明書の管理について詳しくは、DCM の計画を参照してください。

システムのデフォルト証明書を使用しない場合は、別の使用する証明書を選択する必要があります。2種類の証明書を選択できます。

- **ユーザー証明書。** この証明書は、アプリケーションのユーザーを識別します。
- **システム証明書。** この証明書は、アプリケーションが実行されているシステムを識別します。

以下の場合、ユーザー証明書が必要です。

- アプリケーションがクライアント・アプリケーションとして実行されている。
- どのユーザーがアプリケーションを使って作業しているかを識別する証明書が必要である。

以下の場合、システム証明書が必要です。

- アプリケーションがサーバー・アプリケーションとして実行されている。
- アプリケーションが実行されているシステムを識別する証明書が必要である。

必要な種類の証明書を判別し終えたならば、アクセス可能な証明書コンテナの中から該当するデジタル証明書を選択できます。

### Java アプリケーション実行時にデジタル証明書を使用する:

Secure Sockets Layer (SSL) を使用するには、デジタル証明書を使用して Java アプリケーションを実行する必要があります。

使用するデジタル証明書を指定するには、以下のプロパティを使用してください。

- `os400.certificateContainer`
- `os400.certificateLabel`

たとえば、デジタル証明書 MYCERTIFICATE を使用して Java アプリケーション MyClass.class を実行したい場合に、MYCERTIFICATE がデジタル証明書コンテナ YOURDCC 中にあると、java コマンドは以下ようになります。

```
java -Dos400.certificateContainer=YOURDCC  
-Dos400.certificateLabel=MYCERTIFICATE MyClass
```

使用するデジタル証明書をまだ決めていない場合は、使用するデジタル証明書の選択を参照してください。システムのデフォルト証明書を使用するように決めることもできます。この証明書は、システムのデフォルトの証明書コンテナ中に保管されています。

システムのデフォルトのデジタル証明書を使用するには、証明書や証明書コンテナをどこにも指定する必要はありません。ご使用の Java アプリケーションで自動的にシステムのデフォルトのデジタル証明書が使用されます。

iSeries デジタル証明書の管理とインターネットについては、DCM の計画を参照してください。

### デジタル証明書と `-os400.certificateLabel` プロパティ

デジタル証明書は、保護システム、ユーザー、およびアプリケーションを識別するためのインターネット標準です。デジタル証明書は、デジタル証明書コンテナ中に保管されています。デジタル証明書コンテナのデフォルトの証明書を使用したい場合は、証明書のラベルを指定する必要はありません。特定のデジタル証明書を使用したい場合は、以下のプロパティを使用して java コマンド中に証明書のラベルを指定しなければなりません。

```
os400.certificateLabel=
```

たとえば、MYCERTIFICATE という名前の証明書を使用したい場合は、以下のような java コマンドを入力します。

```
java -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

この例では、Java アプリケーション MyClass により証明書 MYCERTIFICATE が使用されます。MYCERTIFICATE が MyClass に使用されるためには、システムのデフォルトの証明書コンテナー中になければなりません。

## デジタル証明書コンテナーと `-os400.certificateContainer` プロパティ

デジタル証明書コンテナーにはデジタル証明書が保管されています。iSeries のシステム・デフォルト証明書コンテナーを使用したい場合は、証明書コンテナーを指定する必要はありません。特定のデジタル証明書コンテナーを使用するには、以下のプロパティを使用して `java` コマンド中にデジタル証明書コンテナーを指定する必要があります。

```
os400.certificateContainer=
```

たとえば、MYDCC という名前の、使用したいデジタル証明書を含む証明書コンテナーを使用したい場合は、以下のような `java` コマンドを入力します。

```
java -Dos400.certificateContainer=MYDCC MyClass
```

この例では、MyClass.class という名前の Java アプリケーションが、MYDCC という名前のデジタル証明書コンテナー中にあるデフォルトのデジタル証明書を使用して、システム上で実行されます。アプリケーション中に作成したソケットによって、MYDCC 中のデフォルト証明書が使用されて、自己識別が行われ、すべての通信保護が行われます。

デジタル証明書コンテナー中のデジタル証明書 MYCERTIFICATE を使用したい場合は、以下のような `java` コマンドを入力します。

```
java -Dos400.certificateContainer=MYDCC  
-Dos400.certificateLabel=MYCERTIFICATE MyClass
```

## Java Secure Socket Extension の使用

JSSE は、SSL と TLS の両方の基礎となるメカニズムを要約するフレームワークと似ています。基礎となっているプロトコルの複雑さと特色を要約することによって、JSSE はプログラマーが安全で暗号化された通信を使用できるようにすると同時に、セキュリティのぜい弱性を最小限に抑えます。この情報は、J2SDK バージョン 1.4 を実行する iSeries サーバー上で JSSE を使用する場合にのみ当てはまります。Java Secure Socket Extension (JSSE) は、Secure Sockets Layer (SSL) プロトコルと Transport Layer Security (TLS) プロトコルの両方を使用して、クライアントとサーバーの間に安全で暗号化された通信を提供します。

IBM JSSE インプリメンテーションは IBM JSSE と呼ばれています。IBM JSSE には、ネイティブ iSeries JSSE プロバイダーと純粋な Java JSSE プロバイダーが組み込まれています。

### JSSE をサポートするよう iSeries サーバーを構成する:

IBM JSSE を使用するように iSeries サーバーを構成する。このトピックには、ソフトウェア要件、JSSE プロバイダーの変更方法、および必要なセキュリティ・プロパティとシステム・プロパティに関する情報もあります。

ご使用の iSeries サーバー上で Java 2 Software Development Kit (J2SDK) バージョン 1.4 または後続のバージョンを使用する場合、JSSE はすでに構成済みです。デフォルト構成は、ネイティブ iSeries JSSE プロバイダーを使用します。

## JSSE プロバイダーの変更

ネイティブ iSeries JSSE プロバイダーの代わりに純粋な Java JSSE プロバイダーを使用するよう JSSE を構成することができます。いくつかの特定の JSSE セキュリティー・プロパティーと Java システム・プロパティーを変更することによって、この 2 つのプロバイダーの間の切り替えができます。詳しくは、以下のトピックを参照してください。

- JSSE プロバイダー
- JSSE セキュリティー・プロパティー
- Java システム・プロパティー

## セキュリティ・マネージャー

Java セキュリティー・マネージャーを使用可能にして JSSE アプリケーションを実行している場合、使用可能なネットワーク許可を設定する必要がある可能性があります。詳しくは、Permissions in the Java 2 SDK の SSL Permission の説明を参照してください。

### JSSE プロバイダー:

IBM JSSE には、ネイティブ iSeries JSSE プロバイダーと純粋な 2 つの Java JSSE プロバイダーが組み込まれています。どのプロバイダーを選択して使用するかは、アプリケーションの必要に応じて異なります。

3 つのプロバイダーはすべて JSSE インターフェースの仕様に準拠します。これらは双方向通信が可能であったり、任意の他の SSL または TLS インプリメンテーション (非 Java インプリメンテーションでも可) と通信することができます。

## ピュア Java JSSE プロバイダー

ピュア Java JSSE プロバイダーは、以下のフィーチャーを備えています。

- デジタル証明書を制御および構成するためのあらゆるタイプの KeyStore オブジェクト (たとえば、JKS および PKCS12 など) を処理します。
- 複数のインプリメンテーションの JSSE コンポーネントを組み合わせ一緒に使用することができます。

IBMJSSE はピュア Java 実装のプロバイダー名です。このプロバイダー名は、適切なケースを使用して `java.security.Security.getProvider()` メソッドか、あるいは、いくつかの JSSE クラス用の種々の `getInstance()` メソッドに渡す必要があります。

## ピュア Java JSSE FIPS 140-2 プロバイダー

ピュア Java JSSE FIPS 140-2 プロバイダーは、以下のフィーチャーを備えています。

- 暗号モジュール用の連邦情報処理標準 (FIPS) 140-2 を使用してコンパイルします。
- デジタル証明書を制御および構成するためのあらゆるタイプの KeyStore オブジェクトを処理します。

注: ピュア Java JSSE FIPS 140-2 プロバイダーは、任意の他の実装のコンポーネントがその実装にプラグインされることを許可しません。

IBMJSSEFIPS はピュア Java JSSE FIPS 140-2 実装のプロバイダー名です。このプロバイダー名は、適切なケースを使用して `java.security.Security.getProvider()` メソッドか、あるいは、いくつかの JSSE クラス用の種々の `getInstance()` メソッドに渡す必要があります。



## ネイティブ iSeries JSSE プロバイダー

ネイティブ iSeries JSSE プロバイダーは、以下のフィーチャーを備えています。

- ネイティブ iSeries SSL サポートを使用します。
- デジタル証明書を構成、制御するために、デジタル証明書マネージャーの使用を許可します。これは、固有な iSeries タイプの KeyStore (IbmISeriesKeyStore) を介して提供されます。
- 最善のパフォーマンスを提供します。
- 複数のインプリメンテーションの JSSE コンポーネントを組み合わせることで一緒に使用することができます。ただし、最良のパフォーマンスを得るには、JSSE ネイティブ iSeries コンポーネントのみを使用してください。

IbmISeriesSslProvider は、ネイティブ iSeries 実装の名前です。このプロバイダー名は、適切なケースを使用して `java.security.Security.getProvider()` メソッドか、あるいは、いくつかの JSSE クラス用の種々の `getInstance()` メソッドに渡す必要があります。

## デフォルト JSSE プロバイダーの変更

セキュリティー・プロパティーに適切な変更を加えることによって、デフォルトの JSSE プロバイダーを変更することができます。詳しくは、以下のトピックを参照してください。

- JSSE セキュリティー・プロパティー

JSSE プロバイダーを変更した後は、システム・プロパティーが、新しいプロバイダーが必要とするデジタル証明書情報 (鍵ストア) 用の適切な構成を指定していることを確認します。詳しくは、以下のトピックを参照してください。

- Java システム・プロパティー

### JSSE セキュリティー・プロパティー:

Java 仮想マシン (JVM) は、多数の重要なセキュリティー・プロパティーを使用します。これらのセキュリティー・プロパティーは、Java マスター・セキュリティー・プロパティー・ファイルを編集することによって設定されます。

- 1 `java.security` というこのファイルは、通常は iSeries サーバー上の `/QIBM/ProdData/Java400/jdk15/lib/security`
- 1 ディレクトリにあります。

以下のリストは、JSSE を使用するための、関連するいくつかのセキュリティー・プロパティーを示しています。この説明は、`java.security` ファイルを編集するためのガイドとして使用してください。

### `security.provider.<integer>`

使用する JSSE プロバイダー。これは静的に暗号プロバイダー・クラスの登録も行います。以下の例のように、異なる JSSE プロバイダーを正確に指定してください。

```
security.provider.5=com.ibm.as400.ibmonly.net.ssl.Provider
security.provider.6=com.ibm.jsse.IBMJSSEProvider
security.provider.7=com.ibm.fips.jsse.IBMJSSEFIPSPProvider
```

### `ssl.KeyManagerFactory.algorithm`

デフォルトの `KeyManagerFactory` アルゴリズムを指定します。ネイティブ iSeries JSSE プロバイダーの場合は、以下を使用します。

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

純粋な Java JSSE プロバイダーの場合は、以下を使用します。

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

詳しくは、`javax.net.ssl.KeyManagerFactory` の javadoc を参照してください。

### **ssl.TrustManagerFactory.algorithm**

デフォルトの `TrustManagerFactory` アルゴリズムを指定します。ネイティブ iSeries JSSE プロバイダーの場合は、以下を使用します。

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

純粋な Java JSSE プロバイダーの場合は、以下を使用します。

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

詳しくは、`javax.net.ssl.TrustManagerFactory` の javadoc を参照してください。

### **ssl.SocketFactory.provider**

デフォルトの SSL ソケット・ファクトリーを指定します。ネイティブ iSeries JSSE プロバイダーの場合は、以下を使用します。

```
ssl.SocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLSocketFactoryImpl
```

純粋な Java JSSE プロバイダーの場合は、以下を使用します。

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
```

詳しくは、`javax.net.ssl.SSLSocketFactory` の javadoc を参照してください。

### **ssl.ServerSocketFactory.provider**

デフォルトの SSL サーバー・ソケット・ファクトリーを指定します。ネイティブ iSeries JSSE プロバイダーの場合は、以下を使用します。

```
ssl.ServerSocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLServerSocketFactoryImpl
```

純粋な Java JSSE プロバイダーの場合は、以下を使用します。

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

詳しくは、`javax.net.ssl.SSLServerSocketFactory` の javadoc を参照してください。

### **JSSE Java システム・プロパティ:**

アプリケーションで JSSE を使用するには、デフォルトの `SSLContext` オブジェクトが構成の確認を行うために必要な、いくつかのシステム・プロパティを指定する必要があります。いくつかのプロパティは両方のプロバイダーに適用され、その他はネイティブ iSeries プロバイダーにだけ適用されます。

ネイティブ iSeries JSSE プロバイダーを使用するとき、プロパティを指定しない場合、`os400.certificateContainer` はデフォルトの `*SYSTEM` になりますが、それは、JSSE がシステム証明書ストア内のデフォルトのエントリーを使用することを意味します。

### **両方にプロバイダーに作用するプロパティ**

以下のプロパティは両方の JSSE プロバイダーに適用されます。それぞれの説明では、該当する場合には、デフォルトのプロパティも示しています。

### **javax.net.ssl.trustStore**

デフォルトの TrustManager が使用する KeyStore オブジェクトが入っているファイルの名前。デフォルト値は jssecacerts または (jssecacerts が存在しない場合は) cacerts です。

### **javax.net.ssl.trustStoreType**

デフォルトの TrustManager が使用する KeyStore オブジェクトのタイプ。デフォルト値は KeyStore.getDefaultType メソッドによって戻される値です。

### **javax.net.ssl.trustStorePassword**

デフォルトの TrustManager が使用する KeyStore オブジェクトのパスワード。

### **javax.net.ssl.keyStore**

デフォルトの KeyManager が使用する KeyStore オブジェクトが入っているファイルの名前。

### **javax.net.ssl.keyStoreType**

デフォルトの KeyManager が使用する KeyStore オブジェクトのタイプ。デフォルト値は KeyStore.getDefaultType メソッドによって戻される値です。

### **javax.net.ssl.keyStorePassword**

デフォルトの KeyManager が使用する KeyStore オブジェクトのパスワード。

## **iSeries のネイティブ JSSE プロバイダーにのみ作用するプロパティ**

以下のプロパティは、ネイティブ iSeries JSSE プロバイダーにのみ適用されます。

### **os400.secureApplication**

アプリケーション ID。 JSSE は、以下のいずれかのプロパティが指定されていない場合にのみ、このプロパティを使用します。

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.net.ssl.trustStoreType

### **os400.certificateContainer**

使用する鍵リングの名前。 JSSE は、以下のいずれかのプロパティが指定されていない場合にのみ、このプロパティを使用します。

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.keyStoreType
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword

- javax.ssl.net.trustStoreType
- os400.secureApplication

#### os400.certificateLabel

使用する鍵リング・ラベル。JSSE は、以下のいずれかのプロパティーが指定されていない場合にのみ、このプロパティーを使用します。

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword
- javax.ssl.net.trustStoreType
- os400.secureApplication

### 追加情報

システム・プロパティーについて詳しくは、以下のトピックを参照してください。

- 15 ページの『Java システム・プロパティーのリスト』
- Sun Java Web サイトの System Properties。

#### ネイティブ iSeries JSSE プロバイダーを使用する:

ネイティブ iSeries JSSE プロバイダーは、JSSE クラスおよびインターフェースの一式を備えています。これには JSSE KeyStore クラスおよび SSLConfiguration クラスの実装が含まれています。

ネイティブ iSeries プロバイダーを効果的に使用するには、このトピックの情報を使い、さらに 296 ページの『SSLConfiguration の Javadoc 情報』も参照してください。

### SSLContext.getInstance メソッドのプロトコル値

以下の表では、ネイティブ iSeries JSSE プロバイダーの SSLContext.getInstance メソッドのプロトコル値を示し、それを説明しています。

| プロトコル値  | サポートされている SSL プロトコル                     |
|---------|---|
| SSL     | SSL バージョン 2、SSL バージョン 3、および TLS バージョン 1 |
| SSLv2   | SSL バージョン 2                             |
| SSLv3   | SSL バージョン 3                             |
| TLS     | SSL バージョン 2、SSL バージョン 3、および TLS バージョン 1 |
| TLSv1   | TLS バージョン 1                             |
| SSL_TLS | SSL バージョン 2、SSL バージョン 3、および TLS バージョン 1 |

### ネイティブ iSeries KeyStore 実装

ネイティブ iSeries プロバイダーは、IbmIseriesKeyStore タイプの KeyStore クラスの実装を備えています。鍵ストアの実装は、デジタル証明書マネージャー・サポートのまわりのラッパーを提供します。鍵ス

トアの内容は、特定のアプリケーション ID または鍵リング・ファイル、パスワード、およびラベルに応じて異なります。JSSE はデジタル証明書マネージャーから鍵ストア・エントリーをロードします。エントリーをロードするため、JSSE はアプリケーションが最初に鍵ストア・エントリーまたは鍵ストア情報にアクセスしようとするときに、適切なアプリケーション ID または鍵リング情報を使用します。鍵ストアは変更できず、構成の変更はすべてデジタル証明書マネージャーを使用して行わなければなりません。

デジタル証明書マネージャーの使用法については、以下のトピックを参照してください。

デジタル証明書マネージャー

## ネイティブ iSeries プロバイダーを使用する際の推奨事項

以下は、ネイティブ iSeries プロバイダーをできるだけ効率的に実行するための推奨事項です。

- ネイティブ iSeries JSSE プロバイダーが機能するためには、JSSE アプリケーションがネイティブ実装のコンポーネントのみを使用する必要があります。たとえば、ネイティブ iSeries JSSE 対応アプリケーションは、純粋な Java JSSE プロバイダーを使用して作成された X509KeyManager オブジェクトを使用して、ネイティブ iSeries JSSE プロバイダーを使用して作成された SSLContext オブジェクトを正常に初期化することはできません。
- さらに、ネイティブ iSeries プロバイダー内の X509KeyManager および X509TrustManager の実装は、IbmISeriesKeyStore オブジェクトか com.ibm.as400.SSLConfiguration オブジェクトのいずれかを使用して初期化しなければなりません。

注: 将来のリリースでは上記の推奨事項が変更され、ネイティブ iSeries JSSE プロバイダーが非ネイティブ・コンポーネント (たとえば、JKS KeyStore や IbmX509 TrustManagerFactory) へのプラグインを許可するようになる可能性があります。

### SSLConfiguration の Javadoc 情報:

com.ibm.as400

クラス SSLConfiguration

java.lang.Object

|

+--com.ibm.as400.SSLConfiguration

インプリメントされているすべてのインターフェース:

java.lang.Cloneable, javax.net.ssl.ManagerFactoryParameters

```
public final class SSLConfiguration
extends java.lang.Object
implements javax.net.ssl.ManagerFactoryParameters, java.lang.Cloneable
```

このクラスは、ネイティブ iSeries JSSE インプリメンテーションによって必要とされる構成の仕様を提供します。

ネイティブ iSeries JSSE インプリメンテーションは、タイプが "IbmISeriesKeyStore" の KeyStore オブジェクトを使用すると、最も効率的に作動します。このタイプの KeyStore オブジェクトには、デジタル証明書マネージャー (DCM) に登録されたアプリケーション ID かまたは鍵リング・ファイル (デジタル証明書コンテナ) に基づいた、キー項目および信頼できる証明書項目が含まれます。また、このタイプの KeyStore オブジェクトを使用して、"IbmISeriesSslProvider" Provider から X509KeyManger および X509TrustManager オブジェクトを初期化することができます。さらに、X509KeyManager および

X509TrustManager オブジェクトを使用して、"IbmISeriesSslProvider" から SSLContext オブジェクトを初期化することができます。次に SSLContext オブジェクトは、KeyStore オブジェクトに指定された構成情報に基づいて、ネイティブ iSeries JSSE インプリメンテーションへのアクセスを提供します。  
 "IbmISeriesKeyStore" KeyStore のロードが実行されるたびに、アプリケーション ID または鍵リング・ファイルに指定された現行の構成に基づいて、KeyStore が初期化されます。

また、このクラスを使用して、任意の有効なタイプの KeyStore オブジェクトを生成することもできます。KeyStore は、アプリケーション ID または鍵リング・ファイルに指定された現行の構成に基づいて初期化されます。アプリケーション ID または鍵リング・ファイルによって指定された構成に何らかの変更を加えた場合、その変更を反映するために、KeyStore オブジェクトを再生成する必要があります。  
 "IbmISeriesKeyStore" 以外のタイプの KeyStore を正常に作成できるようにするには、鍵リング・パスワードを (アプリケーション ID を使用する場合は、\*SYSTEM 証明書ストアに) 指定する必要があることに注意してください。作成される "IbmISeriesKeyStore" タイプの KeyStore 用の秘密鍵に正常にアクセスできるようにするには、鍵リング・パスワードを指定する必要があります。

**対象:** SDK 1.4 以降

**参照:** KeyStore, X509KeyManager, X509TrustManager, SSLContext

## コンストラクターの要約

**SSLConfiguration()** 新規の SSLConfiguration を作成します。詳しくは、298 ページの『コンストラクターの詳細』を参照してください。

表 6. メソッドの要約

|                        |   |
|------------------------|---|
| void                   | 301 ページの『clear』() すべての get メソッドがヌルを戻すように、オブジェクト内のすべての情報を消去します。                                  |
| java.lang.Object       | 303 ページの『clone』() この SSL 構成の新規コピーを生成します。  |
| boolean                | 302 ページの『equals』(java.lang.Objectobj) 他の何らかのオブジェクトがこのオブジェクトに「相当する」かどうかを示します。                    |
| protected void         | 301 ページの『finalize』() ガーベッジ・コレクションが、オブジェクトの参照がこれ以上ないと判断するときに、ガーベッジ・コレクターによってオブジェクト上で呼び出されます。     |
| java.lang.String       | 300 ページの『getApplicationId』() アプリケーション ID を戻します。   |
| java.lang.String       | 301 ページの『getKeyringLabel』() 鍵リング・ラベルを戻します。  |
| java.lang.String       | 300 ページの『getKeyringName』() 鍵リング名を戻します。  |
| char[]                 | 301 ページの『getKeyringPassword』() 鍵リング・パスワードを戻します。   |
| java.security.KeyStore | 303 ページの『getKeyStore』(char[]password) 与えられたパスワードを使用して、タイプが "IbmISeriesKeyStore" の鍵ストアを戻します。     |
| java.security.KeyStore | 303 ページの『getKeyStore』(java.lang.Stringtype, char[]password) 与えられたパスワードを使用して、要求されたタイプの鍵ストアを戻します。 |
| int                    | 302 ページの『hashCode』() オブジェクトのハッシュ・コード値を戻します。   |
| static void            | (java.lang.String[]args) SSLConfiguration 関数を実行します。   |
| void                   | (java.lang.String[]args, java.io.PrintStreamout) SSLConfiguration 関数を実行します。                     |
| void                   | 302 ページの『setApplicationId』(java.lang.StringapplicationId) アプリケーション ID を設定します。                   |



表 6. メソッドの要約 (続き)

|      |  |
|------|--|
| void | 302 ページの『setApplicationId』(java.lang.String applicationId, char[] password) アプリケーション ID および鍵リング・パスワードを設定します。 |
| void | 301 ページの『setKeyring』(java.lang.String name, java.lang.String label, char[] password) 鍵リング情報を設定します。           |

-----

クラス `java.lang.Object` から継承されるメソッド

`getClass, notify, notifyAll, toString, wait, wait, wait`

-----

## コンストラクターの詳細

### SSLConfiguration

```
public SSLConfiguration()
```

新規の `SSLConfiguration` を作成します。アプリケーション ID および鍵リング情報はデフォルト値に初期設定されます。

アプリケーション ID のデフォルト値は、"`os400.secureApplication`" プロパティーに指定された値です。

鍵リング情報のデフォルト値は、"`os400.secureApplication`" プロパティーが指定されている場合は `null` です。"`os400.secureApplication`" プロパティーが指定されていない場合、鍵リング名のデフォルト値は、"`os400.certificateContainer`" プロパティーに指定された値です。"`os400.secureApplication`" プロパティーが指定されていない場合、鍵リング・ラベルは "`os400.certificateLabel`" プロパティーに値に初期設定されます。"`os400.secureApplication`" プロパティーと "`os400.certificateContainer`" プロパティーのどちらも設定されていない場合、鍵リング名は "`*SYSTEM`" に初期設定されます。

-----

## メソッドの詳細

-----

### main

```
public static void main(java.lang.String[] args)
```

`SSLConfiguration` 関数を実行します。実行できるコマンドは、`-help`、`-create`、`-display`、および `-update` の 4 つです。コマンドは、指定される最初のパラメーターでなければなりません。

指定できるオプションは以下のとおりです (任意の順序)。

**`-keystore` *keystore-file-name***

作成、更新、または表示される鍵ストア・ファイルの名前を指定します。このオプションは、すべてのコマンドに必須です。

**`-storepass` *keystore-file-password***

作成、更新、または表示される鍵ストア・ファイルに関連したパスワードを指定します。このオプションは、すべてのコマンドに必須です。

**-storetype keystore-type**

作成、更新、または表示される鍵ストア・ファイルのタイプを指定します。このオプションは、どのコマンドにも指定できます。このオプションが指定されない場合、"IbmISeriesKeyStore" の値が使用されます。

**-appid application-identifier**

作成または更新される鍵ストア・ファイルを初期化するために使用されるアプリケーション ID を指定します。このオプションは、**-create** および **-update** コマンドには任意指定です。 **-appid**、**keyring**、および **-systemdefault** オプションの 1 つだけを指定できます。

**-keyring keyring-file-name**

作成または更新される鍵ストア・ファイルを初期化するために使用される鍵リング・ファイル名を指定します。このオプションは、**-create** および **-update** コマンドには任意指定です。 **-appid**、**keyring**、および **-systemdefault** オプションの 1 つだけを指定できます。

**-keyringpass keyring-file-password**

作成または更新される鍵ストア・ファイルを初期化するために使用される鍵リング・ファイル・パスワードを指定します。このオプションは、**-create** および **-update** コマンドに指定できます。また、鍵ストア・タイプに "IbmISeriesKeyStore" 以外が指定されているときには、このオプションは必須です。このオプションが指定されない場合、隠しておく鍵リング・パスワードが使用されます。

**-keyringlabel keyring-file-label**

作成または更新される鍵ストア・ファイルを初期化するために使用される鍵リング・ラベルを指定します。このオプションは、**-keyring** オプションも指定されている場合に限り指定できます。**keyring** オプションが指定されているときにこのオプションが指定されない場合、鍵リング内のデフォルト・ラベルが使用されます。

**-systemdefault**

作成または更新される鍵ストア・ファイルを初期化するために使用されるシステム・デフォルト値を指定します。このオプションは、**-create** および **-update** コマンドには任意指定です。 **-appid**、**keyring**、および **-systemdefault** オプションの 1 つだけを指定できます。

**-v** 冗長出力が作成されることを指定します。このオプションは、どのコマンドにも指定できます。

ヘルプ・コマンドは、パラメーターをこのメソッドに指定するための使用法情報を表示します。ヘルプ機能呼び出すためのパラメーターは、以下のように指定されます。

**-help**

作成コマンドは、新規の鍵ストア・ファイルを指定します。作成コマンドには 3 つのバリエーションがあります。1 つめのバリエーションは、特定のアプリケーション ID に基づいて鍵ストアを作成します。2 つめのバリエーションは、鍵リングの名前、ラベル、およびパスワードに基づいて鍵ストアを作成します。3 つめのバリエーションは、システム・デフォルト構成に基づいて鍵ストアを作成します。

特定のアプリケーション ID に基づいて鍵ストアを作成するには、**-appid** オプションを指定する必要があります。次のパラメーターは、名前が "keystore.file"、パスワードが "keypass" であるタイプ "IbmISeriesKeyStore" の鍵ストア・ファイルを作成します。これはアプリケーション ID "APPID" に基づいて初期設定されます。

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore  
-appid APPID
```

特定の鍵リング・ファイルに基づいて鍵ストアを作成するには、**-keyring** オプションを指定する必要があります。また、**-keyringpass** および **keyringlabel** オプションを指定することもできます。次のパラメーター

は、名前が "keystore.file"、パスワードが "keypass" であるタイプ "IbmISeriesKeyStore" の鍵ストア・ファイルを作成します。これは、名前が "keyring.file"、鍵リング・パスワードが "ringpass"、鍵リング・ラベルが "keylabel" の鍵リング・ファイルに基づいて初期設定されます。

```
-create -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
        -keyring keyring.file -keyringpass ringpass -keyringlabel keylabel
```

システム・デフォルト構成に基づいて鍵ストアを作成するには、`-systemdefault` オプションを指定する必要があります。次のパラメーターは、名前が "keystore.file"、パスワードが "keypass" であるタイプ "IbmISeriesKeyStore" の鍵ストア・ファイルを作成します。これはシステム・デフォルト構成に基づいて初期設定されます。

```
-create -keystore keystore.file -storepass keypass -systemdefault
```

更新コマンドは、タイプが "IbmISeriesKeyStore" の既存の鍵ストア・ファイルを更新します。更新コマンドには、作成コマンドのバリエーションと同一の 3 つのバリエーションがあります。更新コマンドのオプションは、作成コマンドに使用したオプションと同一です。表示コマンドは、既存の鍵ストア・ファイルに指定された構成を表示します。次のパラメーターは、名前が "keystore.file"、パスワードが "keypass" であるタイプ "IbmISeriesKeyStore" の鍵ストア・ファイルで指定された構成を表示します。

```
-display -keystore keystore.file -storepass keypass -storetype IbmISeriesKeyStore
```

#### パラメーター:

args - コマンド行引数

---

## run

```
public void run(java.lang.String[] args,
                java.io.PrintStream out)
```

SSLConfiguration 関数を実行します。このメソッドのパラメーターおよび機能は main() メソッドと同一です。

#### パラメーター:

args - コマンド引数

out - 結果が書き込まれる出力ストリーム

参照: `com.ibm.as400.SSLConfiguration.main()`

---

## getApplicationId

```
public java.lang.String getApplicationId()
```

アプリケーション ID を戻します。

#### 戻されるもの:

アプリケーション ID。

---

## getKeyringName

```
public java.lang.String getKeyringName()
```

鍵リング名を戻します。

戻されるもの:  
    鍵リング名。

---

## getKeyringLabel

```
public java.lang.String getKeyringLabel()
```

鍵リング・ラベルを戻します。

戻されるもの:  
    鍵リング・ラベル。

---

## getKeyringPassword

```
public final char[] getKeyringPassword()
```

鍵リング・パスワードを戻します。

戻されるもの:  
    鍵リング・パスワード。

---

## finalize

```
protected void finalize()  
    throws java.lang.Throwable
```

ガーベッジ・コレクションが、オブジェクトの参照がこれ以上ないと判断するときに、ガーベッジ・コレクターによってオブジェクト上で呼び出されます。

オーバーライド:  
    クラス `java.lang.Object` の `finalize`

スロー:  
    `java.lang.Throwable` - このメソッドによって出される例外。

---

## clear

```
public void clear()
```

すべての `get` メソッドがヌルを戻すように、オブジェクト内のすべての情報を消去します。

---

## setKeyring

```
public void setKeyring(java.lang.Stringname,  
                      java.lang.Stringlabel,  
                      char[]password)
```

鍵リング情報を設定します。

パラメーター:  
    name - 鍵リング名

label - 鍵リング・ラベル。または、デフォルトの鍵リング項目が使用される場合はヌル。  
password - 鍵リング・パスワード。または隠しておくパスワードが使用される場合はヌル。

---

## setApplicationId

```
public void setApplicationId(java.lang.String applicationId)
```

アプリケーション ID を設定します。

パラメーター:

applicationId - アプリケーション ID。

---

## setApplicationId

```
public void setApplicationId(java.lang.String applicationId,  
char[] password)
```

アプリケーション ID および鍵リング・パスワードを設定します。鍵リング・パスワードを指定すると、作成される鍵ストアが秘密鍵にアクセスできるようになります。

パラメーター:

applicationId - アプリケーション ID。

password - 鍵リング・パスワード。

---

## equals

```
public boolean equals(java.lang.Object obj)
```

他の何らかのオブジェクトがこのオブジェクトに「相当する」かどうかを示します。

オーバーライド:

クラス java.lang.Object の equals

パラメーター:

obj - 比較されるオブジェクト

戻されるもの:

オブジェクトが同じ構成情報を指定するかどうかを示す標識

---

## hashCode

```
public int hashCode()
```

オブジェクトのハッシュ・コード値を戻します。

オーバーライド:

クラス java.lang.Object の hashCode

戻されるもの:

このオブジェクトのハッシュ・コード値。

---

## clone

```
public java.lang.Object clone()
```

この SSL 構成の新規コピーを生成します。この SSL 構成のコンポーネントに対するこれ以降の変更は新規コピーに影響を与えません。逆の場合も同じです。

### オーバーライド:

クラス `java.lang.Object` の `clone`

### 戻されるもの:

この SSL 構成のコピー

---

## getKeyStore

```
public java.security.KeyStore getKeyStore(char[] password)
                                throws java.security.KeyStoreException
```

与えられたパスワードを使用して、タイプが "IbmISeriesKeyStore" の鍵ストアを戻します。鍵ストアは、オブジェクトに現在保管されている構成情報に基づいて初期設定されます。

### パラメーター:

`password` - 鍵ストアの初期設定に使用されます

### 戻されるもの:

オブジェクトに現在保管されている構成情報に基づいて初期設定される `KeyStore` 鍵ストア

### スロー:

`java.security.KeyStoreException` - 鍵ストアを作成できなかった場合

---

## getKeyStore

```
public java.security.KeyStore getKeyStore(java.lang.Stringtype,
                                           char[] password)
                                throws java.security.KeyStoreException
```

与えられたパスワードを使用して、要求されたタイプの鍵ストアを戻します。鍵ストアは、オブジェクトに現在保管されている構成情報に基づいて初期設定されます。

### パラメーター:

`type` - 戻される鍵ストアのタイプ

`password` - 鍵ストアの初期設定に使用されます

### 戻されるもの:

オブジェクトに現在保管されている構成情報に基づいて初期設定される `KeyStore` 鍵ストア

### スロー:

`java.security.KeyStoreException` - 鍵ストアを作成できなかった場合

### 例: IBM Java Secure Sockets Extension:

JSSE の例では、クライアントおよびサーバーがネイティブ `iSeries JSSE` プロバイダーを使用して、安全な通信を可能にするコンテキストを作成する方法を示しています。



注: いずれの例でも、java.security ファイルの指定するプロパティにかかわらず、ネイティブ iSeries JSSE プロバイダーを使用します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

『例: SSLContext オブジェクトを使用する SSL クライアント』

このクライアント・プログラムの例では、"MY\_CLIENT\_APP" アプリケーション ID を使用するために初期化を行う、SSLContext オブジェクトを使用します。このプログラムでは、java.security ファイルにおける指定の有無にかかわらず、ネイティブ iSeries インプリメンテーションを使用します。

306 ページの『例: SSLContext オブジェクトを使用する SSL サーバー』

以下のサーバー・プログラムは、過去に作成された鍵ストア・ファイルによって初期化を行う、SSLContext オブジェクトを使用します。鍵ストア・ファイルの名前は /home/keystore.file であり、鍵ストア・パスワードは password です。

このサンプル・プログラムは、IbmSeriesKeyStore オブジェクトを作成するために鍵ストア・ファイルを必要とします。鍵ストア・オブジェクトはアプリケーション ID として MY\_SERVER\_APP を指定しなければなりません。

鍵ストア・ファイルを作成するために、以下のコマンドのいずれかを使用することができます。

- Qshell コマンド・プロンプトから:

```
java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
-storepass password -appid MY_SERVER_APP
```

Qshell で Java コマンドを使用することについては、iSeries Information Center 内の Qshell を参照してください。

- iSeries コマンド・プロンプトから:

```
RUNJAVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
'/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
```

**例: SSLContext オブジェクトを使用する SSL クライアント:**

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////
//
// This example client program utilizes an SSLContext object, which it initializes
// to use the "MY_CLIENT_APP" application ID.
//
// The example uses the native iSeries JSSE provider, regardless of the
// properties specified by the java.security file.
//
// Command syntax:
//   java -Djava.version=1.4 SslClient
//
// Note that "-Djava.version=1.4" is unnecessary when you have configured
// J2SDK version 1. to be used by default.
//
////////////////////////////////////
```

```
import java.io.*;
import javax.net.ssl.*;
```

```
/**
```

```

* SSL Client Program.
*/
public class SslClient {

    /**
     * SslClient main method.
     *
     * @param args the command line arguments (not used)
     */
    public static void main(String args[]) {
        /*
         * Set up to catch any exceptions thrown.
         */
        try {
            /*
             * Initialize an SSLConfiguration object to specify an application
             * ID. "MY_CLIENT_APP" must be registered and configured
             * correctly with the Digital Certificate Manager (DCM).
             */
            SSLConfiguration config = new SSLConfiguration();
            config.setApplicationId("MY_CLIENT_APP");
            /*
             * Get a KeyStore object from the SSLConfiguration object.
             */
            Char[] password = "password".toCharArray();
            KeyStore ks = config.getKeyStore(password);
            /*
             * Allocate and initialize a KeyManagerFactory.
             */
            KeyManagerFactory kmf =
                KeyManagerFactory.getInstance("IbmISeriesX509");
            KmF.init(ks, password);
            /*
             * Allocate and initialize a TrustManagerFactory.
             */
            TrustManagerFactory tmf =
                TrustManagerFactory.getInstance("IbmISeriesX509");
            tmf.init(ks);
            /*
             * Allocate and initialize an SSLContext.
             */
            SSLContext c =
                SSLContext.getInstance("SSL", "quot");
            C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            /*
             * Get the an SSLSocketFactory from the SSLContext.
             */
            SSLSocketFactory sf = c.getSocketFactory();
            /*
             * Create an SSLSocket.
             *
             * Change the hard-coded IP address to the IP address or host name
             * of the server.
             */
            SSLSocket s = (SSLSocket) sf.createSocket("1.1.1.1", 13333);
            /*
             * Send a message to the server using the secure session.
             */
            String sent = "Test of java SSL write";
            OutputStream os = s.getOutputStream();
            os.write(sent.getBytes());
            /*
             * Write results to screen.
             */
            System.out.println("Wrote " + sent.length() + " bytes...");
            System.out.println(sent);
            /*

```

```

        * Receive a message from the server using the secure session.
        */
        InputStream is = s.getInputStream();
        byte[] buffer = new byte[1024];
        int bytesRead = is.read(buffer);
        if (bytesRead == -1)
            throw new IOException("Unexpected End-of-file Received");
        String received = new String(buffer, 0, bytesRead);
        /*
        * Write results to screen.
        */
        System.out.println("Read " + received.length() + " bytes...");
        System.out.println(received);
    } catch (Exception e) {
        System.out.println("Unexpected exception caught: " +
            e.getMessage());
        e.printStackTrace();
    }
}
}
}

```

## リンク集

コードのサンプルに関する特記事項

### 例: *SSLContext* オブジェクトを使用する SSL サーバー:

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

////////////////////////////////////
//
// The following server program utilizes an SSLContext object that it
// initializes with a previously created keystore file.
//
// The keystore file has the following name and keystore password:
//   File name: /home/keystore.file
//   Password:  password
//
// The example program needs the keystore file in order to create an
// IbmISeriesKeyStore object. The KeyStore object must specify MY_SERVER_APP as
// the application identifier.
//
// To create the keystore file, you can use the following Qshell command:
//
//   java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
//   -storepass password -appid MY_SERVER_APP
//
// Command syntax:
//   java -Djava.version=1.4 JavaSslServer
//
// Note that "-Djava.version=1.4" is unnecessary when you have configured
// J2SDK version 1. to be used by default.
//
////////////////////////////////////

import java.io.*;
import javax.net.ssl.*;

/**
 * Java SSL Server Program using Application ID.
 */
public class JavaSslServer {

    /**
     * JavaSslServer main method.
     *

```

```

* @param args the command line arguments (not used)
*/
public static void main(String args[]) {
    /*
    * Set up to catch any exceptions thrown.
    */
    try {
        /*
        * Allocate and initialize a KeyStore object.
        */
        Char[] password = "password".toCharArray();
        KeyStore ks = KeyStore.getInstance("IbmISeriesKeyStore");
        FileInputStream fis = new FileInputStream("/home/keystore.file");
        Ks.load(fis, password);
        /*
        * Allocate and initialize a KeyManagerFactory.
        */
        KeyManagerFactory kmf =
            KeyManagerFactory.getInstance("IbmISeriesX509");
        Kmf.init(ks, password);
        /*
        * Allocate and initialize a TrustManagerFactory.
        */
        TrustManagerFactory tmf =
            TrustManagerFactory.getInstance("IbmISeriesX509");
        tmf.init(ks);
        /*
        * Allocate and initialize an SSLContext.
        */
        SSLContext c =
            SSLContext.getInstance("SSL", "IbmISeriesSslProvider");
        C.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
        /*
        * Get the an SSLServerSocketFactory from the SSLContext.
        */
        SSLServerSocketFactory sf = c.getSSLServerSocketFactory();
        /*
        * Create an SSLServerSocket.
        */
        SSLServerSocket ss =
            (SSLServerSocket) sf.createServerSocket(13333);
        /*
        * Perform an accept() to create an SSLSocket.
        */
        SSLSocket s = (SSLSocket) ss.accept();
        /*
        * Receive a message from the client using the secure session.
        */
        InputStream is = s.getInputStream();
        byte[] buffer = new byte[1024];
        int bytesRead = is.read(buffer);
        if (bytesRead == -1)
            throw new IOException("Unexpected End-of-file Received");
        String received = new String(buffer, 0, bytesRead);
        /*
        * Write results to screen.
        */
        System.out.println("Read " + received.length() + " bytes...");
        System.out.println(received);
        /*
        * Echo the message back to the client using the secure session.
        */
        OutputStream os = s.getOutputStream();
        os.write(received.getBytes());
        /*
        * Write results to screen.
        */
    }
}

```

```

        System.out.println("Wrote " + received.length() + " bytes...");
        System.out.println(received);
    } catch (Exception e) {
        System.out.println("Unexpected exception caught: " +
            e.getMessage());
        e.printStackTrace();
    }
}
}

```

## リンク集

コードのサンプルに関する特記事項

## Java Authentication and Authorization Service

Java Authentication and Authorization Service (JAAS) は、Java 2 Software Development Kit (J2SDK), Standard Edition の標準拡張機能です。J2SDK が提供するアクセス制御は、コードの発生元や署名者に基づくものです (コード・ソースをベースにしたアクセス制御)。ただし、コードの実行者に基づく追加のアクセス制御を実施する機能に欠けています。JAAS が提供するフレームワークには、このサポートが Java 2 セキュリティ・モデルに対して追加されています。

JAAS API は、J2SDK バージョン 1.3 の拡張機能として、IBM および Sun Microsystems, Inc. によって使用されています。IBM および Sun は、特定のユーザーや身元を現行の Java スレッドに関連付けられるようにするため、この拡張機能を導入しています。これを行うには、`javax.security.auth.Subject` メソッドを使用します。オプションとして、基礎となるオペレーティング・システムのスレッドで `com.ibm.security.auth.ThreadSubject` メソッドを使用することもできます。

注: 注: J2SDK バージョン 1.4 および後続のバージョンでは、JAAS は拡張機能ではなくなり、基本 SDK の一部になりました。

iSeries サーバー上の JAAS 実装は、Sun Microsystems, Inc. の実装と互換性があります。この資料では、iSeries 実装に固有の側面について扱います。ここでは、JAAS 拡張機能の一般資料に精通していることを前提とします。この情報と iSeries の情報を利用しやすくするために、以下のリンクが用意されています。

- 310 ページの『Java 認証・承認サービス (JAAS) 1.0』では、ソフトウェア開発における JAAS API の使用についての情報が提供されます。
- JAAS LoginModule Developer's Guide では、JAAS の認証に関する面に焦点を合わせています。
- JAAS API Specificationには、JAAS に関する Javadoc の情報が示されています。

### 関連情報

iSeries サーバー固有の JAAS Javadoc

## JAAS (Java Authentication and Authorization Service) 用に iSeries サーバーを準備して構成する

Java Authentication and Authorization Service (JAAS) を使用するには、ソフトウェア要件を満たして、iSeries サーバーを構成しなければなりません。

### iSeries サーバー上で JAAS 1.0 を実行するためのソフトウェア要件

以下のライセンス・プログラムをインストールしてください。

- Java 2 SDK、バージョン 1.4 (J2SDK) またはそれ以上

- IBM Toolbox for Java (mod 4) ライセンス・プログラム (5722-JC1) は、OS スレッド身元を変更するために必要です。これには、iSeries の OS スレッド身元の変更をサポートするために必要な ProfileTokenCredential クラスと、固有の実装クラスが含まれています。

## システムを構成する

システムを構成して JAAS を使用できるようにするには、以下のステップに従ってください。

1. J2SDK 1.3 で、jaas13.jar ファイル用の拡張ディレクトリーへのシンボリック・リンクを追加します。これにより、拡張クラス・ローダーは、この JAR ファイルをロードします。リンクを追加するには、iSeries コマンド入力行で以下のコマンドを (すべて 1 行で) 実行します。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
```

注: J2SDK 1.4 およびそれ以降では、拡張ディレクトリーへのシンボリック・リンクを追加する必要はありません。JAAS はこのバージョンでの基本 SDK の一部です。

2. デフォルトの login.config ファイルは `{java.home}/lib/security` にあります。これは、`com.ibm.as400.security.auth.login.BasicAuthenticationLoginModule` を呼び出すものです。この login.config ファイルは、単一用途の ProfileTokenCredential を認証済みサブジェクトに付加します。別のオプションを指定した独自の login.config ファイルを使用するには、アプリケーションの起動時に以下のシステム・プロパティーを含めることができます。

```
-Djava.security.auth.login.config=your login.config file
```

3. jt400Native.jar ファイル用の拡張ディレクトリーへのシンボリック・リンクを追加します。これにより、拡張クラス・ローダーがこのファイルをロードできるようになります。jaas13.jar ファイルは、IBM Toolbox for Java の一部である信任状実装クラスでこの JAR ファイルを必要とします。また、このファイルを CLASSPATH に含めれば、アプリケーション・クラス・ローダーもこのファイルをロードできます。このファイルがクラスパス・ディレクトリーからロードされる場合は、拡張ディレクトリーへのシンボリック・リンクを追加しないでください。

jt400Native.jar ファイルから /QIBM/ProdData/Java400/jdk14/lib/ext ディレクトリーへのシンボリック・リンクを作成すると、サーバー上のすべての J2SDK 1.4 ユーザーは、このバージョンの jt400Native.jar を使用します。さまざまなユーザーが、さまざまなバージョンの IBM Toolbox for Java クラスを要求する場合、この方法は望ましくありません。別の選択肢として、前述のように、アプリケーションの CLASSPATH に jt400Native.jar を入れることができます。また、独自のディレクトリーへのシンボリック・リンクを追加してから、アプリケーションの起動時に java.ext.dirs システム・プロパティーを指定してそのディレクトリーを拡張ディレクトリーのクラスパスに含める、という選択肢もあります。

jt400Native.jar ファイルを /QIBM/ProdData/Java400/jdk13/lib/ext ディレクトリーにリンクするには、iSeries コマンド入力行で以下のコマンドを実行してリンクを追加します。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')
```

jt400Native.jar ファイルを /QIBM/ProdData/Java400/jdk14/lib/ext ディレクトリーにリンクするには、iSeries コマンド入力行で以下のコマンドを実行してリンクを追加します。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')
```

jt400Native.jar を独自のディレクトリーにリンクするには、次のようにします。

- a. リンクを追加するには、iSeries コマンド入力行で以下のコマンドを実行します。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('your extension directory/jt400Native.jar')
```



b. java プログラムを呼び出すときは、以下のパターンを使用します。

```
java -Djava.ext.dirs=your extension directory:default  
extension directories
```

注: iSeries 信任状クラスについての詳細は、IBM Toolbox for Java を参照してください。「**セキュリティー・クラス**」をクリックします。次に「**認証サービス**」をクリックしてマウスを移動します。

「**ProfileTokenCredential**」クラスをクリックします。最後に「**Package**」をクリックします。

4. Java 2 ポリシー・ファイルを更新して、IBM Toolbox for Java JAR ファイルの実際の位置への適切なアクセスを認可します。これらのファイルから拡張ディレクトリーへのシンボリック・リンクが作成され、\${java.home}/lib/security/java.policy ファイル内でそのディレクトリーに java.security.AllPermission が認可されていても、与信は JAR ファイルの実際の位置に基づいて行われます。

IBM Toolbox for Java で信任状クラスを正しく使用するには、アプリケーションの Java 2 ポリシー・ファイルに以下の行を追加します。

```
grant codeBase "file:/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar"  
{  
    permission javax.security.auth.AuthPermission "modifyThreadIdentity";  
    permission java.lang.RuntimePermission "loadLibrary.*";  
    permission java.lang.RuntimePermission "writeFileDescriptor";  
    permission java.lang.RuntimePermission "readFileDescriptor";  
}
```

これらの許可は、アプリケーションの codeBase にも追加する必要があります。これは、IBM Toolbox for Java JAR ファイルが実行する操作が特権モードでは実行されないためです。

Java 2 ポリシー・ファイルについては、『Java 認証・承認サービス (JAAS) 1.0』を参照してください。

5. iSeries ホスト・サーバーが始動して稼働していることを確認してください。Toolbox 内にある ProfileTokenCredential クラス (jt400Native.jar など) は、認証済みサブジェクトに付加された信任状として使用されます。信任状クラスはホスト・サーバーへのアクセスを必要とします。サーバーが始動して稼働しているかどうかを検査するには、iSeries コマンド・プロンプトで以下のコマンドを入力します。

```
StrHostSVR *all  
StrTcpSvr *DDM
```

サーバーがすでに始動している場合は、何も起こりません。サーバーが始動していない場合は、これらのステップでサーバーが始動されます。

## Java 認証・承認サービス (JAAS) 1.0

この資料は、2000 年 3 月 17 日に最終更新されました。

### 開発者用ガイド

- 概説
- この資料の対象読者
- 関連資料
- 紹介
- コア・クラス
- 共通クラス
  - Subject
  - Principals

- **Credentials**
- **認証クラス**
- **LoginContext**
- **LoginModule**
- **CallbackHandler**
- **Callback**
- **権限クラス**
- **Policy**
- **AuthPermission**
- **PrivateCredentialPermission**

## リファレンス

- インプリメンテーション
- "Hello World"、JAAS スタイル!
- 付録 A: java.security セキュリティー・プロパティー・ファイルでの JAAS 設定
- 付録 B: ログイン構成ファイル
- 付録 C: 権限ポリシー・ファイル

## 概説

Java認証・承認サービス (JAAS) は、Java 2 Software Development Kit、バージョン 1.3 の標準拡張機能です。現在、Java 2 は、コード・ソースに基づくアクセス制御 (コードの発信元 およびコードの署名者 に基づいたアクセス制御) を提供しています。ただし、コードの実行者 に基づく追加のアクセス制御を実施する機能に欠けています。JAAS が提供するフレームワークでは、このサポートとともに Java 2 セキュリティー・モデルを拡大しています。

## この資料の対象読者

この資料は、コード・ソース・ベースのセキュリティー・モデルと Subject ベースのセキュリティー・モデルによって制限されたアプリケーションを作成しようとしている、経験を積んだプログラマーを対象としています。

## 関連資料

この資料は、読者が以下の資料をすでに読み終えていることを前提としています。

- Java 2 Software Development Kit API Specification
- JAAS API 仕様
- Security and the Java platform

このガイドの補足は、Sun Microsystems, Inc. が提供する LoginModule Developer's Guide です。

## 紹介

JAAS インフラストラクチャーは、2 つのメイン・コンポーネント、すなわち**認証**コンポーネントと**権限**(承認) コンポーネントに分割できます。JAAS 認証コンポーネントは、Java がアプリケーション、アプレット、Bean、またはサーブレットとして稼働しているかどうかにかかわらず、現在だれがそのコードを処理しているかについて**確実かつ安全に**判別する機能を提供します。JAAS 権限 (承認) コンポーネント

は、そのコード・ソース (Java 2 で実行される) に応じて、また認証された人物に応じて、処理中の Java コードが機密タスクを実行するのを制約することにより、既存の Java 2 セキュリティー・フレームワークを補足します。

JAAS 認証は、プラグ可能形式で実行されます。これにより、Java アプリケーションは、基礎となる認証テクノロジーからの独立を保つことが可能になります。したがって、アプリケーションそのものに変更を加えなくても、新規または更新された認証テクノロジーをアプリケーションの下で接続することができます。

アプリケーションは、

LoginContext

オブジェクトをインスタンス化することにより、認証プロセスを使用可能にすることができます。次に、認証プロセスは認証テクノロジーを判別するために

Configuration

を参照するか、または認証の実行に使用するために

LoginModule

を参照します。標準的な LoginModules はユーザー名およびパスワードを入力するようにプロンプトを出して、それらを確認します。あるいは、音声または指紋のサンプルを読み取って、それらを確認する場合があります。

コードを処理するユーザーが認証されると、JAAS 権限 (承認) コンポーネントが既存の Java 2 アクセス制御モデルとともに機能して、重要なリソースへのアクセスを保護します。

アクセス制御決定がコードの場所およびコードの署名者にのみ基づく (

CodeSource

) Java 2 とは異なり、JAAS アクセス制御決定は処理コードの

CodeSource

と、コードを実行するユーザーの両方、または

Subject

に基づきます。JAAS ポリシーは Java 2 ポリシーを、関係のある Subject ベースの情報で拡張したものに過ぎないことに注意してください。そのため、Java 2 で認識され、理解される許可 (たとえば、

java.io.FilePermission

および

java.net.SocketPermission

) も JAAS によって理解され、認識されます。さらに、JAAS セキュリティー・ポリシーは既存の Java 2 セキュリティー・ポリシーとは物理的に分離しているにもかかわらず、2 つのポリシーは一緒に 1 つの論理ポリシーを形成します。

## コア・クラス

JAAS コア・クラスは、3 つのカテゴリである共通、認証、および権限に分けることができます。

- 共通クラス

- Subject、Principals、Credentials

- 認証クラス
  - LoginContext、LoginModule、CallbackHandler、Callback
- 権限クラス
  - Policy、AuthPermission、PrivateCredentialPermission

## 共通クラス

共通クラスは、JAAS 認証コンポーネントと権限 (承認) コンポーネントの両方に共有されます。

主要な JAAS クラスは

Subject

で、単一のエンティティ (個人など) に関連した情報のグループを表します。これは、エンティティのプリンシパル、公開信任状、および秘密信任状を包含します。

JAAS は、既存の Java 2

`java.security.Principal`

インターフェースを使用して、プリンシパルを表すことに注意してください。また、JAAS は別個の信任状インターフェースまたはクラスを導入しないことにも注意してください。JAAS によって定義される信任状は任意のオブジェクトです。

## Subject

リソースへのアクセスを許可するには、アプリケーションは最初に要求のソースを認証する必要があります。JAAS フレームワークは、要求のソースを表すために、サブジェクトという用語を定義します。サブジェクトは任意のエンティティ (個人またはサービスなど) です。サブジェクトが認証されると、そこに関連した ID、つまりプリンシパルが取り込まれます。サブジェクトは多数のプリンシパルを持つことがあります。たとえば、ある個人が名前プリンシパル ("John Doe") と、それを他のサブジェクトと区別する SSN プリンシパル ("123-45-6789") を持つことがあります。

また、

Subject

がセキュリティ関連の属性を持つこともあります。これを信任状と言います。特殊な保護を必要とする重要な信任状 (秘密暗号鍵など) は、秘密信任状

Set

に保管されます。共有されることが意図されている信任状 (公開鍵証明書または Kerberos チケットなど) は、公開信任状

Set

に保管されます。アクセスして変更する信任状セットが異なれば、それに応じた異なる許可が必要です。

サブジェクトは次のコンストラクターを使用して作成されます。

```
public Subject();  
  
public Subject(boolean readOnly, Set principals,  
               Set pubCredentials, Set privCredentials);
```

最初のコンストラクターは、プリンシパルおよび信任状の空 (非ヌル) のセットを持つサブジェクトを作成します。2 番目のコンストラクターは、プリンシパルおよび信任状の指定されたセットを持つサブジェクトを作成します。また、読み取り専用サブジェクト (不変のプリンシパルおよび信任状セット) を作成できるブル引数を持っています。

これらのコンストラクターを使用しないで、認証済みサブジェクトへの参照を取得する代わりに方法が、`LoginContext` セクションに示されています。

サブジェクトが読み取り専用状態になるようにインスタンス化されなかった場合、次のメソッドを呼び出すことによって、読み取り専用状態に設定することができます。

```
public void setReadOnly();
```

このメソッドを呼び出すには、

```
AuthPermission("setReadOnly")
```

が必要です。読み取り専用状態になったら、プリンシパルまたは信任状を追加または削除しようとすると、`IllegalStateException`

が出されます。

サブジェクトの読み取り専用状態をテストするには、次のメソッドを呼び出すことができます。

```
public boolean isReadOnly();
```

サブジェクトに関連したプリンシパルを検索するには、次の 2 つのメソッドが使用可能です。

```
public Set getPrincipals();  
public Set getPrincipals(Class c);
```

最初のメソッドは、サブジェクトに含まれるすべてのプリンシパルを戻し、2 番目のメソッドは、指定されたクラス `c` のインスタンスまたはクラス `c` のサブクラスのインスタンスであるプリンシパルだけを戻します。サブジェクトに関連したプリンシパルがない場合には、空のセットが戻されます。

サブジェクトに関連した公開信任状を検索するには、次のメソッドが使用可能です。

```
public Set getPublicCredentials();  
public Set getPublicCredentials(Class c);
```

これらのメソッドで観察される動作は、

```
getPrincipals
```

メソッドの動作と同一です。

サブジェクトに関連した秘密信任状を検索するには、次のメソッドが使用可能です。

```
public Set getPrivateCredentials();  
public Set getPrivateCredentials(Class c);
```

これらのメソッドで観察される動作は、

```
getPrincipals
```

および

```
getPublicCredentials
```

メソッドの動作と同一です。

サブジェクトのプリンシパル・セット、公開信任状セット、または秘密信任状セットを変更するか、またはそれに対する操作を行うには、呼び出し元が

```
java.util.Set
```

クラスで定義されたメソッドを使用します。次の例は、このことを示しています。

```
Subject subject;  
Principal principal;  
Object credential;  
  
// add a Principal and credential to the Subject  
subject.getPrincipals().add(principal);  
subject.getPublicCredentials().add(credential);
```

それぞれのセットを変更するには、

```
AuthPermission("modifyPrincipals")
```

、

```
AuthPermission("modifyPublicCredentials")
```

、または

```
AuthPermission("modifyPrivateCredentials")
```

が必要です。また、

```
getPrincipals
```

、

```
getPublicCredentials
```

、および

```
getPrivateCredentials
```

メソッドから戻されるセットだけが、サブジェクトのそれぞれの内部セットによって戻されることに注意してください。そのため、戻されたセットに対する変更は、内部セットにも影響を与えます。

```
getPrincipals(Class c)
```

、

```
getPublicCredentials(Class c)
```

、および

```
getPrivateCredentials(Class c)
```

メソッドから戻されるセットは、サブジェクトのそれぞれの内部セットによって戻されません。メソッドの呼び出しのたびに、新規セットが作成されて戻されます。これらのセットに対する変更は、サブジェクトの内部セットに影響を与えません。次のメソッドは、指定された

```
AccessControlContext
```

に関連したサブジェクトを戻すか、または指定した

```
AccessControlContext
```

に関連したサブジェクトがない場合には、ヌルを戻します。

```
public static Subject getSubject(final AccessControlContext acc);
```



Subject.getSubject

を呼び出すには、

```
AuthPermission("getSubject")
```

が必要です。

また、Subject クラスには、

```
java.lang.Object
```

から継承された次のメソッドが含まれます。

```
public boolean equals(Object o);  
public String toString();  
public int hashCode();
```

特定のサブジェクトとして処理を実行するために、次の静的メソッドを呼び出すことができます。

```
public static Object doAs(final Subject subject,  
                          final java.security.PrivilegedAction action);  
  
public static Object doAs(final Subject subject,  
                          final java.security.PrivilegedExceptionAction action)  
                          throws java.security.PrivilegedActionException;
```

どちらのメソッドも最初に、指定された **subject** を現行のスレッドの

AccessControlContext

に関連付け、次に **action** を処理します。これにより、**action** が **subject** として実行されます。最初のメソッドは実行時例外をスローすることがありますが、通常処理では、このメソッドはその action 引数の run() メソッドから Object を戻します。2 番目のメソッドは同様に動作しますが、その

PrivilegedExceptionAction

run() メソッドからチェック済み例外をスローできる点が異なります。

```
AuthPermission("doAs")
```

は、

```
doAs
```

メソッドを呼び出すために必要です。

最初の

```
doAs
```

メソッドを使用する 2 つの例を示します。クラスが

```
com.ibm.security.Principal
```

で、“BOB” という名前のプリンシパルを持つ

```
Subject
```

が

```
LoginContext
```

"lc" によって認証されていることを想定します。また、SecurityManager がインストールされており、JAAS アクセス制御ポリシーに以下が存在していることも想定しています (JAAS ポリシー・ファイルの詳細は、Policy セクションを参照してください)。

```
// Grant "BOB" permission to read the file "foo.txt"
grant Principal com.ibm.security.Principal "BOB" {
    permission java.io.FilePermission "foo.txt", "read";
};
```

### Subject.doAs の例 1

```
class ExampleAction implements java.security.PrivilegedAction {
    public Object run() {
        java.io.File f = new java.io.File("foo.txt");

        // exists() invokes a security check
        if (f.exists()) {
            System.out.println("File foo.txt exists.");
        }
        return null;
    }
}

public class Example1 {
    public static void main(String[] args) {

        // Authenticate the subject, "BOB".
        // This process is described in the
        // LoginContext section.

        Subject bob;
        ...

        // perform "ExampleAction" as "BOB":
        Subject.doAs(bob, new ExampleAction());
    }
}
```

処理中に、

ExampleAction

では、

f.exists()

を呼び出す際にセキュリティ検査が行われます。ただし、

ExampleAction

は "BOB" として実行しており、JAAS ポリシー (上述) は、必要な

FilePermission

を "BOB" に付与するため、

ExampleAction

はセキュリティ検査をパスします。

例 2 は、例 1 と同じシナリオを使用します。

### Subject.doAs の例 2

```

public class Example2 {
    // Example of using an anonymous action class.
    public static void main(String[] args) {
        // Authenticate the subject, "BOB".
        // This process is described in the
        // LoginContext section.

        Subject bob;
        ...

        // perform "ExampleAction" as "BOB":
        Subject.doAs(bob, new ExampleAction() {
            public Object run() {
                java.io.File f = new java.io.File("foo.txt");
                if (f.exists()) {
                    System.out.println("File foo.txt exists.");
                }
                return null;
            }
        });
    }
}

```

例の `permission grant` ステートメントが間違っていて変更された場合 (正しくない `CodeBase` を追加したり、`Principal` を "MOE" に変更するなど)、どちらの例も

`SecurityException`

をスローします。 `grant` ブロックから `Principal` フィールドを除去してから、それを Java 2 ポリシー・ファイルに移動させると、

`SecurityException`

はスローされません。なぜなら、ここでは許可がより一般的になっているからです (すべてのプリンシパルに使用可能)。

どちらの例も同じ関数を実行するため、一方のコードを他方より優先するには理由が必要です。例 1 は、無名クラスに精通していないプログラマーにとって、読み取りやすいでしょう。また、*action* クラスは、固有の `CodeBase` を持つ個別のファイルに置くことができます。そうすると、`permission grant` はこの情報を使用できます。例 2 はよりコンパクトで、実行される *action* は見つけるのが簡単です。なぜなら、

`doAs`

の呼び出しで実行されるからです。

さらに、次のメソッドも特定のサブジェクトとして処理を実行します。

ただし、

`doAsPrivileged`

メソッドには、提供された *action* および *subject* に基づいてセキュリティ検査があります。提供されたコンテキストは、指定された *subject* および *action* に結びつけられます。ヌル・コンテキスト・オブジェクトは、現行の

`AccessControlContext`

を完全に無視します。

```

public static Object doAsPrivileged(final Subject subject,
    final java.security.PrivilegedAction action,
    final java.security.AccessControlContext acc);

```

```
public static Object doAsPrivileged(final Subject subject,
    final java.security.PrivilegedExceptionAction action,
    final java.security.AccessControlContext acc)
    throws java.security.PrivilegedActionException;
```

doAsPrivileged

メソッドは、次の点で

doAs

メソッドと同様に動作します。すなわち、**subject** はコンテキスト **acc** に関連付けられること、**action** が実行されること、および実行時例外またはチェック済み例外がスローされることです。ただし、

doAsPrivileged

メソッドは、最初に既存のスレッドの

AccessControlContext

を空にしてから、**subject** を提供されたコンテキストに関連付けて、**action** を呼び出します。ヌルの **acc** 引数を指定すると、アクセス制御決定 (**action** の処理中に呼び出される) は **subject** と **action** だけに基づくこととなります。

```
AuthPermission("doAsPrivileged")
```

は、

doAsPrivileged

メソッドを呼び出すために必要です。

## Principals

すでに説明したとおり、プリンシパルはサブジェクトに関連付けることができます。プリンシパルは、サブジェクトの ID を表しており、

```
java.security.Principal
```

および

```
java.io.Serializable
```

インターフェースをインプリメントする必要があります。Subject セクションでは、サブジェクトに関連したプリンシパルを更新する方法を説明しています。

## Credentials

公開信任状クラスと秘密信任状クラスは、コア JAAS クラス・ライブラリーの一部ではありません。そのため、どの java クラスでも信任状を表すことができます。ただし、開発者はそれらの信任状クラスに、信任状に関連した 2 つのインターフェース、つまり Refreshable と Destroyable をインプリメントすることを選択できます。

## Refreshable

このインターフェースは、信任状がそれ自身をリフレッシュする機能を提供します。たとえば、特定の時間制限付きの有効期間を指定した信任状は、その信任状が有効である期間を呼び出し元がリフレッシュできるようにするためにこのインターフェースをインプリメントします。インターフェースには次の 2 つの抽象メソッドがあります。

```
boolean isCurrent();
```

信任状が現行、つまり有効かどうかを判別します。

```
void refresh() throws RefreshFailedException;
```

信任状の妥当性を更新または拡張します。このメソッドのインプリメンテーションでは、

```
AuthPermission("refreshCredential")
```

セキュリティー検査を実行して、呼び出し元が信任状をリフレッシュする許可を持つようにします。

## Destroyable

このインターフェースは、信任状内の内容を破棄する機能を提供します。インターフェースには次の 2 つの抽象メソッドがあります。

```
boolean isDestroyed();
```

信任状が破棄されているかどうかを判別します。

```
void destroy() throws DestroyFailedException;
```

この信任状に関連した情報を破棄して消去します。これ以降に、この信任状に対して特定のメソッドを呼び出すと、

```
IllegalStateException
```

がスローされます。このメソッドのインプリメンテーションでは、

```
AuthPermission("destroyCredential")
```

セキュリティー検査を実行して、呼び出し元が信任状を破棄する許可を持つようにします。

## 認証クラス

Subject

を認証するために、以下のステップが実行されます。

1. アプリケーションは、

```
LoginContext
```

をインスタンス化します。

2. LoginContext

は構成を調べて、そのアプリケーション用に構成されたすべての LoginModules をロードします。

3. アプリケーションは LoginContext の login メソッドを呼び出します。

4. login メソッドは、ロードされたすべての LoginModules を呼び出します。それぞれの

```
LoginModule
```

は、

```
Subject
```

を認証しようとしています。成功すると、LoginModules は関係のあるプリンシパルおよび信任状を

```
Subject
```

に関連付けます。

## 5. LoginContext

は認証状況をアプリケーションに戻します。

## 6. 認証が成功すると、アプリケーションは認証済みの

Subject

を

LoginContext

から取得します。

## LoginContext

LoginContext

クラスは、サブジェクトを認証するために使用される基本メソッドを提供し、さらに基礎となる認証テクノロジーから独立してアプリケーションを開発する方法を提供します。

LoginContext

は構成

Configuration

を調べて、特定のアプリケーション用に構成された認証サービス、すなわち `LoginModules` を判別します。したがって、アプリケーションそのものに変更を加えなくても、そのアプリケーションの下で様々な `LoginModules` を接続することができます。

LoginContext

は、選択できる以下の 4 つのコンストラクターを提供します。

```
public LoginContext(String name) throws LoginException;

public LoginContext(String name, Subject subject) throws LoginException;

public LoginContext(String name, CallbackHandler callbackHandler)
    throws LoginException

public LoginContext(String name, Subject subject,
    CallbackHandler callbackHandler) throws LoginException
```

すべてのコンストラクターは共通のパラメーター、**name** を共有します。この引数は、ログイン構成に索引を付けるために、

LoginContext

によって使用されます。入力パラメーターとして

Subject

を取らないコンストラクターは、新規の

Subject

をインスタンス化します。すべてのコンストラクターで、ヌル入力は許可されません。呼び出し元は、`AuthPermission("createLoginContext")`

に

LoginContext



をインスタンス化することを求めます。

実際の認証は、次のメソッドの呼び出しとともに行われます。

```
public void login() throws LoginException;
```

`login` が呼び出されると、構成された `LoginModules` のそれぞれの `login` メソッドがすべて、認証を実行するために呼び出されます。認証が成功すると、認証済みの

`Subject`

(この時点で、プリンシパル、公開信任状、および秘密信任状を保有可能です) を、次のメソッドを使用して取得することができます。

```
public Subject getSubject();
```

`Subject`

をログアウトし、その認証済みのプリンシパルおよび信任状を除去するために、次のメソッドが提供されています。

```
public void logout() throws LoginException;
```

アプリケーション内の次のコードの断片は、“`moduleFoo`” という名前の構成項目を持つ構成ファイルにアクセスした後で、“`bob`” と呼ばれるサブジェクトを認証します。

```
Subject bob = new Subject();
LoginContext lc = new LoginContext("moduleFoo", bob);
try {
    lc.login();
    System.out.println("authentication successful");
} catch (LoginException le) {
    System.out.println("authentication unsuccessful"+le.printStackTrace());
}
```

アプリケーション内の次のコードの断片は、「名前のない」サブジェクトを認証してから、`getSubject` メソッドを使用してそれを取得します。

```
LoginContext lc = new LoginContext("moduleFoo");
try {
    lc.login();
    System.out.println("authentication successful");
} catch (LoginException le) {
    System.out.println("authentication unsuccessful"+le.printStackTrace());
}
Subject subject = lc.getSubject();
```

認証が失敗すると、`getSubject` はヌルを戻します。また、

`Subject.getSubject`

の場合に存在した、これを行うために必要な

`AuthPermission("getSubject")`

はありません。

## LoginModule

`LoginModule` インターフェースは、アプリケーションの下で接続できる様々な種類の認証テクノロジーをインプリメントする機能を開発者に提供します。たとえば、

`LoginModule`

の 1 つのタイプでは、ユーザー名/パスワード・ベースの形式の認証を実行できます。

LoginModule Developer's Guide は、LoginModules をインプリメントするための段階的な説明を開発者に与える詳しい資料です。

LoginModule

をインスタンス化するためには、

LoginContext

は各

LoginModule

が、引数を取らない public コンストラクターを提供することを期待します。次に、

LoginModule

を関連情報とともにインスタンス化するために、

LoginContext

は、LoginModule の

initialize

メソッドを呼び出します。提供された *subject* は、ヌル以外であることが保証されます。

```
void initialize(Subject subject, CallbackHandler callbackHandler,  
               Map sharedState, Map options);
```

次のメソッドは、認証プロセスを開始します。

```
boolean login() throws LoginException;
```

メソッド・インプリメンテーションの例では、ユーザーにユーザー名とパスワードを入力するように促し、その後 NIS または LDAP などの命名サービスに保管されたデータと照らして情報を検査することができます。代替のインプリメンテーションでは、スマート・カードおよびバイOMETリック認証デバイスとのインターフェースをとり、単にユーザー情報を基礎となるオペレーティング・システムから抽出します。これは、JAAS 認証プロセスの **フェーズ 1** と見なされます。

次のメソッドは、認証プロセスを完了し、終了させます。

```
boolean commit() throws LoginException;
```

認証プロセスの **フェーズ 1** が成功した場合、このメソッドは **フェーズ 2** を継続します。それは、プリンシパル、公開信任状、および秘密信任状をサブジェクトに関連付けることです。 **フェーズ 1** が失敗した場合、 *commit* メソッドは以前に保管された認証状態 (ユーザー名およびパスワードなど) を除去します。

次のメソッドは、 **フェーズ 1** が成功しなかった場合に認証プロセスを停止します。

```
boolean abort() throws LoginException;
```

このメソッドの標準的インプリメンテーションでは、以前に保管された認証状態 (ユーザー名またはパスワードなど) をクリーンアップします。次のメソッドは、サブジェクトをログアウトします。

```
boolean logout() throws LoginException;
```

このメソッドは、元々

Subject

に関連付けられているプリンシパルおよび信任状を

```
commit
```

操作中に除去します。信任状は除去時に破棄されます。

## CallbackHandler

場合によっては、LoginModule は、認証情報を取得するためにユーザーと通信しなければなりません。LoginModules はこのために CallbackHandler を使用します。アプリケーションは CallbackHandler インターフェースをインプリメントし、それを LoginContext に渡します。そして、基礎となる LoginModules に直接転送します。LoginModules は、ユーザーからの入力 (パスワードまたはスマート・カード・ピン番号など) を収集することと、ユーザーに情報 (状況情報など) を提供することの両方のために、CallbackHandler を使用します。アプリケーションが CallbackHandler を指定できるようにすることにより、基礎となる LoginModules は、アプリケーションがユーザーと対話する様々な方法からは独立したままでいられます。たとえば、GUI アプリケーション用の CallbackHandler をインプリメントすると、ユーザーからの入力を送信請求するためにウィンドウが表示されます。他方、非 GUI ツール用の CallbackHandler をインプリメントすると、ユーザーはコマンド行から直接入力するように促されます。

```
CallbackHandler
```

は、以下をインプリメントするための 1 つのメソッドとのインターフェースです。

```
void handle(Callback[] callbacks)
throws java.io.IOException, UnsupportedCallbackException;
```

## Callback

javax.security.auth.callback パッケージには、Callback インターフェースといくつかのインプリメンテーションが含まれています。LoginModules は、Callback の配列を CallbackHandler の handle メソッドに直接渡します。

使用法の詳細については、各種の Callback API を調べてください。

## 権限クラス

```
Subject
```

の認証が成功すると、Subject.doAs または Subject.doAsPrivileged メソッドを呼び出すことにより、きめ細かいアクセス制御をその

```
Subject
```

に対して設定することができます。その

```
Subject
```

に付与される許可は、JAAS

```
Policy
```

内で構成されます。

## Policy

これは、システム規模の JAAS アクセス制御を表すための抽象クラスです。デフォルトとして、JAAS はファイル・ベースのサブクラス・インプリメンテーション、PolicyFile を提供します。それぞれの

```
Policy
```

サブクラスは、以下のメソッドをインプリメントする必要があります。

```
public abstract java.security.PermissionCollection getPermissions
    (Subject subject,
     java.security.CodeSource cs);
public abstract void refresh();
```

getPermissions

メソッドは、指定された

Subject

および

CodeSource

に付与された許可を戻します。

refresh

メソッドは、ランタイム

Policy

を、それが永続ストア (たとえば、ファイルまたはデータベース) から最後にロードされて以降に加えられた変更で更新します。

refresh

メソッドは、

AuthPermission("refreshPolicy")

を必要とします。

次のメソッドは、現行のランタイム

Policy

オブジェクトを取得し、呼び出し元に

AuthPermission("getPolicy")

を持つことを求めるセキュリティ検査で保護されています。

```
public static Policy getPolicy();
```

次のコードの例は、

Policy

オブジェクトを照会して、指定された

Subject

および

CodeSource

に付与された許可のセットを調べる方法を示しています。

```
policy = Policy.getPolicy();
PermissionCollection perms = policy.getPermissions(subject, codeSource);
```

新規の

Policy

オブジェクトを Java ランタイムに設定するには、

Policy.setPolicy

メソッドが使用されます。このメソッドは、呼び出し元が

AuthPermission("setPolicy")

を持っていることを必要とします。

```
public static void setPolicy(Policy policy);
```

### ポリシー・ファイルのサンプル項目:

これらの例は、デフォルトの PolicyFile インプリメンテーションにのみ関係があります。

Policy

内の各項目は、**grant** 項目として表されます。各 **grant** 項目は、codebase/code-signers/Principals トリプレットを指定すると同時に、そのトリプレットに付与される許可を指定します。特に、指定された *codebase* からダウンロードされ、指定された *code signers* によって署名されたコードに、許可が付与されます。このことは、そのコードを実行する

Subject

が、指定されたすべての *Principals* をその

Principal

セットに持っている限り行われます。

Subject

が実行中のコードに関連付けられる様子については、Subject.doAs の例 を参照してください。

```
grant CodeBase ["URL"],
    Signedby ["signers"],
    Principal [Principal_Class] "Principal_Name",
    Principal ... {
    permission Permission_Class ["Target_Name"]
                                   [, "Permission_Actions"]
                                   [, signedBy "SignerName"];
};

// example grant entry
grant CodeBase "http://griffin.ibm.com", Signedby "davis",
    Principal com.ibm.security.auth.NTUserPrincipal "kent" {
    permission java.io.FilePermission "c:/kent/files/*", "read, write";
};
```

### Principal 情報が JAAS

Policy

grant 項目で指定されていない場合、構文解析例外がスローされます。ただし、通常の Java 2 コード・ソース・ベースのポリシー・ファイルにすでに存在している (したがって、Principal 情報を持たない) grant 項目は、依然として有効です。そのような場合、Principal 情報は '\*' (grant 項目はすべてのプリンシパルに適用される) であることが暗示されます。

grant 項目の CodeBase および Signedby コンポーネントは、JAAS

Policy

ではオプションです。それらが存在しない場合、どんなコードベースでもマッチングし、さらにどんな署名者 (符号なしコードを含む) でもマッチングします。

上記の例では、**grant** 項目は、"http://griffin.ibm.com" からダウンロードされ、"davis" によって署名されて、NT ユーザー "kent" として実行しているコードが、1 つの

Permission

を持つことを指定します。この

Permission

は、処理コードがディレクトリー "c:\kent\files" にあるファイルの読み取り/書き込みを行うことを許可します。

複数のプリンシパルが 1 つの **grant** 項目内にリストされることがあります。コードを実行している現行の Subject

は、その

Principal

セット内の指定されたすべてのプリンシパルが項目の許可を付与されるようにする必要があります。

```
grant Principal com.ibm.security.auth.NTUserPrincipal "kent",
    Principal com.ibm.security.auth.NTSidGroupPrincipal "S-1-1-0" {
    permission java.io.FilePermission "c:/user/kent/", "read, write";
    permission java.net.SocketPermission "griffin.ibm.com", "connect";
};
```

この項目は、NT グループ識別番号が "S-1-1-0" の NT ユーザー "kent" として実行する任意のコードに、"c:\user\kent" にあるファイルの読み取りおよび書き込みを行う許可と、"griffin.ibm.com" に対するソケット接続を行う許可の両方を付与します。

## AuthPermission

このクラスは、JAAS に必要な基本的な許可をカプセル化します。AuthPermission には、名前 (「ターゲット名」とも言われる) が含まれますが、アクション・リストは含まれません。名前付きの許可はあってもなくてもかまいません。(

Permission

クラスから) 継承されたメソッドに加えて、

AuthPermission

には、次の 2 つの public コンストラクターがあります。

```
public AuthPermission(String name);
public AuthPermission(String name, String actions);
```

最初のコンストラクターは、指定された名前を持つ新規の AuthPermission を作成します。2 番目のコンストラクターも、指定された名前を持つ新規の AuthPermission オブジェクトを作成しますが、追加の actions 引数 (現在は未使用で、ヌルになっている) を持っています。このコンストラクターは、新規の Permission オブジェクトをインスタンス化するために、

Policy

オブジェクト用にのみ存在します。たいていのコードの場合、最初のコンストラクターが適切です。



AuthPermission オブジェクトは、Policy、Subject、LoginContext、および Configuration オブジェクトへのアクセスを保護するために使用されます。サポートされる有効な名前のリストについては、AuthPermission Javadoc を参照してください。

## PrivateCredentialPermission

このクラスは、Subject の秘密信任状へのアクセスを保護し、1 つの public コンストラクターを提供します。

```
public PrivateCredentialPermission(String name, String actions);
```

このクラスの詳細については、PrivateCredentialPermission Javadoc を参照してください。

## インプリメンテーション

注: 付録 A には、ここで説明する静的プロパティを含む、サンプルの **java.security** ファイルが記載されています。

JAAS プロバイダーおよびポリシー・ファイルにはデフォルト値が存在するため、ユーザーは JAAS をインプリメントするためにそれらのリストを静的に (java.security ファイル内で) リストすることも、動的に (コマンド行 **-D** オプション) リストすることも必要ありません。また、デフォルト構成およびポリシー・ファイル・プロバイダーは、ユーザー開発のプロバイダーによって置き換えられることがあります。そのため、このセクションでは、JAAS デフォルト・プロバイダーおよびポリシー・ファイルとともに、代替のプロバイダーを使用可能にするプロパティの説明を試みます。

ここで要約される事柄よりも詳しい情報については、Default Policy File API および Default Configuration File API をお読みください。

### 認証プロバイダー

認証プロバイダー、または構成クラスは、

```
login.configuration.provider=[class]
```

とともに *java.security* ファイルで静的に設定されます。このプロバイダーは、Configuration

オブジェクトを作成します。

以下に例を示します。

```
login.configuration.provider=com.foo.Config
```

セキュリティー・プロパティ

```
login.configuration.provider
```

が *java.security* にない場合、JAAS はそれを次のデフォルト値に設定します。

```
com.ibm.security.auth.login.ConfigFile
```

Configuration

が作成される前にセキュリティー・マネージャーが設定される場合、

```
AuthPermission("getLoginConfiguration")
```

が付与される必要があります。

構成プロバイダーをコマンド行で動的に設定する方法はありません。

## 認証構成ファイル

認証構成ファイルは、*java.security* で

```
login.config.url.n=[URL]
```

とともに静的に設定されます。ここで、*n* は 1 から始まる連続する整数です。フォーマットは、Java セキュリティー・ポリシー・ファイル (*policy.url.n=[URL]*) のフォーマットと同一です。

セキュリティ・プロパティー

```
policy.allowSystemProperty
```

が *java.security* で "true" に設定される場合、ユーザーは次のプロパティーとともに **-D** オプションを使用して、コマンド行でポリシー・ファイルを動的に設定することができます。

```
java.security.auth.login.config
```

。値はパスまたは URL です。例 (NT の場合):

```
... -Djava.security.auth.login.config=c:%config_policy%login.config ...  
または  
... -Djava.security.auth.login.config=file:c:/config_policy/login.config ...
```

注: コマンド行で二重等号 (==) を使用すると、ユーザーは見つかったその他のすべてのポリシー・ファイルをオーバーライドすることができます。

構成ファイルが静的または動的に見つからない場合、JAAS は構成ファイルを次のデフォルト位置からロードしようとします。

```
${user.home}%.java.login.config
```

ここで、*\${user.home}* はシステムに依存する位置です。

## 権限プロバイダー

権限プロバイダー、または JAAS Policy クラスは、

```
auth.policy.provider=[class]
```

とともに *java.security* ファイルで静的に設定されます。このプロバイダーは、JAAS サブジェクト・ベースの

Policy

オブジェクトを作成します。

以下に例を示します。

```
auth.policy.provider=com.foo.Policy
```

セキュリティ・プロパティー

```
auth.policy.provider
```

が *java.security* にない場合、JAAS はそれを次のデフォルト値に設定します。

```
com.ibm.security.auth.PolicyFile
```

。

## Configuration

が作成される前にセキュリティー・マネージャーが設定される場合、

```
AuthPermission("getPolicy")
```

が付与される必要があります。

権限プロバイダーをコマンド行で動的に設定する方法はありません。

### 権限ポリシー・ファイル

権限ポリシー・ファイルは、`java.security` で

```
auth.policy.url.n=[URL]
```

とともに静的に設定されます。ここで、`n` は 1 から始まる連続する整数です。フォーマットは、Java セキュリティー・ポリシー・ファイル (`policy.url.n=[URL]`) のフォーマットと同一です。

セキュリティー・プロパティー

```
policy.allowSystemProperty
```

が `java.security` で "true" に設定される場合、ユーザーは次のプロパティーとともに **-D** オプションを使用して、コマンド行でポリシー・ファイルを動的に設定することができます。

```
java.security.auth.policy
```

。値はパスまたは URL です。例 (NT の場合):

```
... -Djava.security.auth.policy=c:%auth_policy%java.auth.policy ...  
または  
... -Djava.security.auth.policy=file:c:/auth_policy/java.auth.policy ...
```

注: コマンド行で二重等号 (==) を使用すると、ユーザーは見つかったその他のすべてのポリシー・ファイルをオーバーライドすることができます。

権限ポリシーのロード元にするデフォルト位置はありません。

## "Hello World"、JAAS スタイル!

黒っぽいサングラスをかけ、お気に入りのフェドラー帽をかぶり、手にはアルト・サックスを取り ...

**JAAS-y** を始める 때가 来ました! これがもう一つの "Hello World!" プログラムです。このセクションでは、JAAS インストールをテストするためのプログラムを使用可能にします。

**インストール:** JAAS がインストールされていることが想定されています。たとえば、JAAS JAR ファイルがご使用の Development Kit の `extensions` ディレクトリーにコピーされています。

**ファイルの取得:** `HelloWorld.tar` をご使用のテスト・ディレクトリーにダウンロードします。"`jar xvf HelloWorld.tar`" を使用して、それを解凍します。

テスト・ディレクトリーの内容を検査します。

**ソース・ファイル:**

- `HWLoginModule.java`
- `HWPrincipal.java`
- `HelloWorld.java`

## クラス・ファイル

- ソース・ファイルは、`classes` ディレクトリーにプリコンパイルされています。

## ポリシー・ファイル

- `jaas.config`
- `java2.policy`
- `jaas.policy`

ソース・ファイルのコンパイル: 3 つのソース・ファイル、***HWLoginModule.java***、***HWPrincipal.java***、および ***HelloWorld.java*** はすでにコンパイルされているため、コンパイルする必要はありません。

ソース・ファイルが変更される場合、それらが保管されているテスト・ディレクトリーに変更して、次のように入力します。

```
javac -d .\classes *.java
```

クラスパスに `classes` ディレクトリー (`.\classes`) を追加して、クラスをコンパイルできるようにする必要があります。

注:

`HWLoginModule`

および

`HWPrincipal`

は、

`com.ibm.security`

パッケージにあり、コンパイル時に適切なディレクトリー (`>test_dir<.\classes\com\ibm\security`) に作成されます。

ポリシー・ファイルの調査: 構成ファイル ***jaas.config*** には次の 1 つの項目が含まれています。

```
helloWorld {  
    com.ibm.security.HWLoginModule required debug=true;  
};
```

1 つの

`LoginModule`

だけがテスト・ケースに提供されます。 ***HelloWorld*** アプリケーションを処理するときには、

`LoginModuleControlFlag`

(`required`、`requisite`、`sufficient`、`optional`) を変えたり、デバッグ・フラグを削除したりして、試してみてください。テストで利用できる `LoginModule` が他にもある場合、この構成を自由に変えて複数の `LoginModule` で試してみることもできます。

`HWLoginModule`

について、簡単に説明します。

Java 2 ポリシー・ファイル、***java2.policy*** には、1 つの許可ブロックが含まれています。

```
grant {
    permission javax.security.auth.AuthPermission "createLoginContext";
    permission javax.security.auth.AuthPermission "modifyPrincipals";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

**HelloWorld** アプリケーションは、(1) LoginContext オブジェクトを作成し、(2) 認証された Subject

のプリンシパルを変更し、(3)

Subject

クラスの doAsPrivileged メソッドを呼び出すため、3 つの許可が必要です。

JAAS ポリシー・ファイル **jaas.policy** にも、1 つの許可ブロックが含まれています。

```
grant Principal com.ibm.security.HWPrincipal "bob" {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "foo.txt", "read";
};
```

3 つの許可は、最初に **bob** という名前の

HWPrincipal

に認可されます。認証された

Subject

に追加される実際のプリンシパルは、ログイン・プロセス (さらに後) で使用されるユーザー名です。

以下に **HelloWorld** のアクション・コードを示します。3 つのシステム呼び出し (許可が必要である理由) が**ボールド体**で示されています。

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("Your java.home property: "
            +System.getProperty("java.home"));

        System.out.println("Your user.home property: "
            +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("foo.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        System.out.println("Oh, by the way ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignore
        }
        System.out.println("Hello World!");
        return null;
    }
}, null);
```

**HelloWorld** プログラムを実行するときは、さまざまなユーザー名を使用し、それに応じて **jaas.policy** を変更できます。 **java2.policy** を変更する必要はありません。また、テスト・ディレクトリー内に **foo.txt** というファイルを作成して、最後のシステム呼び出しをテストします。

## ソース・ファイルの調査:

LoginModule、  
HWLoginModule

は、正しいパスワード (大文字小文字の区別がある) を入力したユーザーを認証します: **Go JAAS**。

**HelloWorld** アプリケーションでは、ユーザーは 3 回までパスワードの入力を試みることができます。 **Go JAAS** が正しく入力されると、ユーザー名と同じ名前を持つ

HWPrincipal

オブジェクトが、認証された

Subject

に追加されます。

Principal クラスの

HWPrincipal

は、入力されたユーザー名に基づくプリンシパルを表します。この名前は、認証されたサブジェクトに許可を与える際に重要になります。

メイン・アプリケーションの

HelloWorld

は、**helloWorld** という名前の構成項目に基づいて、

LoginContext

をまず作成します。構成ファイルについてはすでに説明されています。ユーザー入力を検索するためにコールバックが使用されます。このプロセスを調べるには、**HelloWorld.java** ファイルにある

MyCallbackHandler

クラスを参照してください。

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

ユーザーがユーザー名とパスワードを (最大 3 回) 入力し、**Go JAAS** がパスワードとして入力されると、サブジェクトは認証されます (

HWLoginModule

によってサブジェクトに

HWPrincipal

が追加されます)。

すでに説明したとおり、以後の作業は、認証されたサブジェクトとして実行されます。

## HelloWorld テストの実行



**HelloWorld** プログラムを実行するには、最初にテスト・ディレクトリーに変更します。構成およびポリシー・ファイルを読み込む必要があります。正しいプロパティーについてインプリメンテーションを参照して、`java.security` がコマンド行のいずれかに設定します。後者のメソッドについて、これから説明します。

以下のコマンドは、明確にするために複数の行に分割されています。1つの連続するコマンドとして入力してください。

```
java -Djava.security.manager=  
-Djava.security.auth.login.config=.%jaas.config  
-Djava.security.policy=.%java2.policy  
-Djava.security.auth.policy=.%jaas.policy  
HelloWorld
```

注: 各ユーザーのテスト・ディレクトリー標準パスは異なるため、ポリシー・ファイルに "`¥filename`" を使用することが必要です。希望する場合は、"`.`" をテスト・ディレクトリーへのパスの代わりにしてください。たとえば、テスト・ディレクトリーが "`c:¥test¥hello`" の場合、最初のファイルは次のように変更されます。

```
-Djava.security.auth.login.config=c:¥test¥hello¥jaas.config
```

ポリシー・ファイルが見つからない場合は、

```
SecurityException
```

が表示されます。見つかった場合は、`java.home` および `user.home` プロパティーに関する情報が表示されます。また、テスト・ディレクトリーに `foo.txt` というファイルがあるかどうかを検査されます。最後に、いたるところに "Hello World" というメッセージが表示されます。

## >HelloWorld を楽しむ

好きなだけ **HelloWorld** を再実行してください。入力したユーザー名/パスワードを変え、構成ファイル項目を変更し、ポリシー・ファイル許可を変更し、さらに追加の `LoginModules` を `helloWorld` 構成項目に追加する (積み重ねる) ことまで、すでに提案されました。さらに、`codebase` フィールドをポリシー・ファイルに追加することができます。

最後に、セキュリティー・マネージャーなしでプログラムを実行して、問題発生時における動作を調べてみてください。

## 付録 A: java.security セキュリティー・プロパティー・ファイル・ファイルでの JAAS 設定

以下に示すのは、すべての Java 2 インストールに現れる

```
java.security
```

ファイルです。このファイルは、Java 2 ランタイムの

```
lib/security
```

(Windows では

```
lib¥security
```

) ディレクトリーに現れます。したがって、Java 2 ランタイムが

```
jdk1.3
```

というディレクトリーにインストールされている場合、ファイルは以下のようになります。

- jdk1.3/lib/security/java.security

(Unix)

- jdk1.3\lib\security\java.security

(Windows)

JAAS は 4 つの新規のプロパティを

java.security

に追加します。

- 認証プロパティ

—  
login.configuration.provider

—  
login.policy.url.n

- 権限プロパティ

—  
auth.policy.provider

—  
auth.policy.url.n

新規の JAAS プロパティは、このファイルの末尾に置かれます。

```
#
# This is the "master security properties file".
#
# In this file, various security properties are set for use by
# java.security classes. This is where users can statically register
# Cryptography Package Providers ("providers" for short). The term
# "provider" refers to a package or set of packages that supply a
# concrete implementation of a subset of the cryptography aspects of
# the Java Security API. A provider may, for example, implement one or
# more digital signature algorithms or message digest algorithms.
#
# Each provider must implement a subclass of the Provider class.
# To register a provider in this master security properties file,
# specify the Provider subclass name and priority in the format
#
#   security.provider.n=className
#
# This declares a provider, and specifies its preference
# order n. The preference order is the order in which providers are
# searched for requested algorithms (when no specific provider is
# requested). The order is 1-based; 1 is the most preferred, followed
# by 2, and so on.
#
# className must specify the subclass of the Provider class whose
# constructor sets the values of various properties that are required
# for the Java Security API to look up the algorithms or other
# facilities implemented by the provider.
#
# There must be at least one provider specification in java.security.
# There is a default provider that comes standard with the JDK. It
```

```

# is called the "SUN" provider, and its Provider subclass
# named Sun appears in the sun.security.provider package. Thus, the
# "SUN" provider is registered via the following:
#
#   security.provider.1=sun.security.provider.Sun
#
# (The number 1 is used for the default provider.)
#
# Note: Statically registered Provider subclasses are instantiated
# when the system is initialized. Providers can be dynamically
# registered instead by calls to either the addProvider or
# insertProviderAt method in the Security class.

#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun

#
# Class to instantiate as the system Policy. This is the name of the class
# that will be used as the Policy object.
#
policy.provider=sun.security.provider.PolicyFile
# The default is to have a single system-wide policy file,
# and a policy file in the user's home directory.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy

# whether or not we expand properties in the policy file
# if this is set to false, properties (${...}) will not be expanded in policy
# files.
policy.expandProperties=true

# whether or not we allow an extra policy to be passed on the command line
# with -Djava.security.policy=somefile. Comment out this line to disable
# this feature.
policy.allowSystemProperty=true

# whether or not we look into the IdentityScope for trusted Identities
# when encountering a 1.1 signed JAR file. If the identity is found
# and is trusted, we grant it AllPermission.
policy.ignoreIdentityScope=false

#
# Default keystore type.
#
keystore.type=jks

#
# Class to instantiate as the system scope:
#
system.scope=sun.security.provider.IdentityDatabase

#####
#
# Java Authentication and Authorization Service (JAAS)
# properties and policy files:
#

# Class to instantiate as the system Configuration for authentication.
# This is the name of the class that will be used as the Authentication
# Configuration object.
#
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
# The default is to have a system-wide login configuration file found in
# the user's home directory. For multiple files, the format is similar to
# that of CodeSource-base policy files above, that is policy.url.n

```

```
login.config.url.1=file:${user.home}/.java.login.config

# Class to instantiate as the system Principal-based Authorization Policy.
# This is the name of the class that will be used as the Authorization
# Policy object.
#
auth.policy.provider=com.ibm.security.auth.PolicyFile
# The default is to have a system-wide Principal-based policy file found in
# the user's home directory. For multiple files, the format is similar to
# that of CodeSource-base policy files above, that is policy.url.n and
# auth.policy.url.n
auth.policy.url.1=file:${user.home}/.java.auth.policy
```

## 付録 B: ログイン構成ファイル

ログイン構成ファイルには、以下の形式の 1 つ以上の

```
LoginContext
```

アプリケーション名が含まれています。

```
Application {
    LoginModule Flag ModuleOptions;
    > more LoginModule entries <
    LoginModule Flag ModuleOptions;
};
```

ログイン構成ファイルは、

```
java.security
```

ファイルにある

```
login.config.url.n
```

セキュリティ・プロパティを使用して配置されます。このプロパティおよび

```
java.security
```

ファイルの位置については、付録 A を参照してください。

*Flag* の値は、認証が回を重ねて進むときの全部的な動作を制御します。以下は、*Flag* の有効な値と、それぞれの意味の説明を表しています。

### 1. Required。

```
LoginModule
```

は正常に実行される必要があります。成功しても失敗しても、認証は

```
LoginModule
```

リストの処理を引き続き先に進めます。

### 2. Requisite。

```
LoginModule
```

は正常に実行される必要があります。成功すると、認証は

```
LoginModule
```

リストの処理を継続します。失敗すると、制御はただちにアプリケーションに戻ります (認証は

```
LoginModule
```

リストの処理を先に進めません)。

### 3. **Sufficient**。

LoginModule

は正常に実行される必要がありません。成功すると、制御はただちにアプリケーションに戻ります (認証は

LoginModule

リストの処理を先に進めません)。失敗すると、認証は

LoginModule

リストの処理を継続します。

### 4. **Optional**。

LoginModule

は正常に実行される必要がありません。成功しても失敗しても、認証は

LoginModule

リストの処理を引き続き先に進めます。

全体の認証は、すべての *Required* および *Requisite* の LoginModules が正常に実行される場合に限り成功します。

*Sufficient* の

LoginModule

が構成されて成功する場合、全体の認証が成功するには、その *Sufficient* の

LoginModule

の前に *Required* および *Requisite* の LoginModules だけが正常に実行される必要があります。 *Required* または *Requisite* の LoginModules がアプリケーション用に構成されていない場合、少なくとも 1 つの *Sufficient* または *Optional* の

LoginModule

が正常に実行される必要があります。

### サンプル構成ファイル:

```
/* Sample Configuration File */
```

```
Login1 {  
    com.ibm.security.auth.module.SampleLoginModule required debug=true;  
};
```

```
Login2 {  
    com.ibm.security.auth.module.SampleLoginModule required;  
    com.ibm.security.auth.module.NTLoginModule sufficient;  
    ibm.loginModules.SmartCard requisite debug=true;  
    ibm.loginModules.Kerberos optional debug=true;  
};
```

注: フラグには大文字小文字の区別がありません。 *REQUISITE* = *requisite* = *Requisite* です。

**Login1** は、クラス

com.ibm.security.auth.module.SampleLoginModule

のインスタンスである 1 つの LoginModule のみ持っています。したがって、**Login1** に関連した LoginContext

は、そのただ 1 つのモジュールが正常に認証を行う場合に限り、正常に認証されます。 *Required* フラグは、この例では意味がありません。複数のモジュールが存在するときに、フラグ値は認証に関係のある影響を与えます。

**Login2** は、表を使って説明する方が簡単です。

| Login2 認証状況     |            |      |      |      |      |      |      |      |      |
|-----------------|------------|------|------|------|------|------|------|------|------|
| サンプル・ログイン・モジュール | required   | pass | pass | pass | pass | fail | fail | fail | fail |
| NT ログイン・モジュール   | sufficient | pass | fail | fail | fail | pass | fail | fail | fail |
| スマート・カード        | requisite  | *    | pass | pass | fail | *    | pass | pass | fail |
| Kerberos        | optional   | *    | pass | fail | *    | *    | pass | fail | *    |
| 全体の認証           |            | pass | pass | pass | fail | fail | fail | fail | fail |

\* = 以前の *REQUISITE* モジュールが失敗したか、または以前の *SUFFICIENT* モジュールが成功したため、アプリケーションに制御が戻ったことによる、意味のない値。

## 付録 C: 権限ポリシー・ファイル

上記のプリンシパル・ベースの JAAS ポリシーの grant ブロックの例では十分でない場合、ここにさらに説明があります。

```
// SAMPLE JAAS POLICY FILE: java.auth.policy
```

```
// The following permissions are granted to Principal 'Pooh' and all codesource:
```

```
grant Principal com.ibm.security.Principal "Pooh" {
    permission javax.security.auth.AuthPermission "setPolicy";
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "c:/foo/jaas.txt", "read";
};
```

```
// The following permissions are granted to Principal 'Pooh' AND 'Eyeore'
// and CodeSource signedBy "DrSecure":
```

```
grant signedBy "DrSecure"
    Principal com.ibm.security.Principal "Pooh",
    Principal com.ibm.security.Principal "Eyeore" {
    permission javax.security.auth.AuthPermission "modifyPublicCredentials";
    permission javax.security.auth.AuthPermission "modifyPrivateCredentials";
    permission java.net.SocketPermission "us.ibm.com", "connect,accept,resolve";
    permission java.net.SocketPermission "griffin.ibm.com", "accept";
};
```

```
// The following permissions are granted to Principal 'Pooh' AND 'Eyeore' AND
// 'Piglet' and CodeSource from the c:%jaas directory signed by "kent" and "bruce":
```

```
grant codeBase "file:c:/jaas/*",
    signedBy "kent, bruce",
    Principal com.ibm.security.Principal "Pooh",
    Principal com.ibm.security.Principal "Eyeore",
    Principal com.ibm.security.Principal "Piglet" {
    permission javax.security.auth.AuthPermission "getSubject";
    permission java.security.SecurityPermission "printIdentity";
    permission java.net.SocketPermission "guapo.ibm.com", "accept";
};
```

## JAAS (Java Authentication and Authorization Service) のサンプル

このトピックには、iSeries サーバー上での Java Authentication and Authorization Service (JAAS) サンプルが記載されています。

2 つの JAAS サンプルとして、HelloWorld および SampleThreadSubjectLogin があります。説明とソース・コードについては、以下をクリックしてください。

**iSeries サーバー上の JAAS (Java Authentication and Authorization Service) で HelloWorld をコンパイルして実行する:**

ここでは、Java Authentication and Authorization Service (JAAS) 用の **HelloWorld** を iSeries サーバー上でコンパイルして実行する方法について考えます。

ここでの情報は、API Developers Guide の『**HelloWorld**』のセクションを置き換えるものとなります。ソース・コード、ポリシー、および構成ファイルは、『API Developers Guide』のものと同じです。ただし、iSeries サーバー固有の面もいくつかあります。

1.

独自のテスト・ディレクトリーに以下のソース・ファイルを置いてください。

- HWLoginModule.java
- HWPrincipal.java
- HelloWorld.java

これらのソース・ファイルを、`.classes` ディレクトリーにコンパイルする必要があります。

使用する HTML ブラウザーの形式にフォーマットされた、これらのファイルのソース・コードを見るには、HTML 形式の HelloWorldを参照してください。

2.

これら 3 つのソース・ファイル、HWLoginModule.java、HWPrincipal.java、HelloWorld.java をコンパイルする必要があります。iSeries コマンド入力行で、以下のコマンドを (それぞれを 1 行で) 実行してください。

a.

```
strqsh
```

b.

```
cd yourTestDir
```

c.



```
javac -J-Djava.version=1.3
      -classpath /qibm/proddata/os400/java400/ext/jaas13.jar:.
      -d ./classes *.java
```

`yourTestDir` は、サンプル・ファイルを入れるために作成したディレクトリーです。クラスパスに `classes` ディレクトリー (`¥classes`) を追加して、クラスをコンパイルできるようにする必要があります。

注: `HWLoginModule` および `HWPrincipal` は `com.ibm.security` パッケージにあり、コンパイル時に適切なディレクトリー (`¥classes¥com¥ibm¥security`) に作成されます。

3.

`iSeries` コマンド入力行で、以下のコマンドを (それぞれを 1 行で) 実行してください。

a.

```
strqsh
```

b. `cd yourTestDir`

`yourTestDir` は、サンプル・ファイルを入れるために作成したディレクトリーです。クラスパスに `classes` ディレクトリー (`¥classes`) を追加して、クラスをコンパイルできるようにする必要があります。

c. 独自のテスト・ディレクトリーに以下のソース・ファイルを置いてください。

- `jaas.config`
- `java2.policy`
- `jaas.policy`

d.

```
java -Djava.security.manager=
     -Djava.security.auth.login.config=./jaas.config
     -Djava.security.policy=./java2.policy
     -Djava.security.auth.policy=./jaas.policy
     -Djava.version=1.3
     -classpath ./classes
     HelloWorld
```

e. ユーザー名を要求するプロンプトが出されたら、**bob** と入力してください。セキュリティー・マネージャーとともに実行している場合、成功させるには、すべてのアクセス許可についてユーザー **bob** を入力しなければなりません。パスワードを要求するプロンプトが出されたら、**Go JAAS** と入力してください。大文字小文字の区別があり、スペースも必要です。

**詳細: JAAS (Java Authentication and Authorization Service) 用の HelloWorld の動作:** この文書では、Java Authentication and Authorization Service (JAAS) 用の **HelloWorld** の動作について詳しく説明します。ここでの情報は、API Developers Guide の『**HelloWorld**』のセクションを置き換えるものとなります。ソース・コード、ポリシー、および構成ファイルは、『API Developers Guide』のものと同じです。ただし、`iSeries` サーバー固有の面もいくつかあります。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

### 構成ファイルとポリシー・ファイル

構成ファイル `jaas.config` には、次の 1 つの項目が含まれています。

```
helloWorld {
    com.ibm.security.HWLoginModule required debug=true;
};
```

このテスト・ケースでは、1 つの LoginModule しか組み込まれません。 HelloWorld アプリケーションを実行するときは、LoginModuleControlFlag (required、requisite、sufficient、optional) を変えたり、デバッグ・フラグを削除したりすることができます。テストで利用できる LoginModule が他にもある場合、この構成を変えて複数の LoginModule で試してみることもできます。

Java 2 ポリシー・ファイル **java2.policy** には、1 つの許可ブロックが含まれています。

```
grant {
    permission javax.security.auth.AuthPermission "createLoginContext";
    permission javax.security.auth.AuthPermission "modifyPrincipals";
    permission javax.security.auth.AuthPermission "doAsPrivileged";
};
```

これら 3 つの許可は、HelloWorld アプリケーションが以下のことを行うために必要です。

1. LoginContext オブジェクトを作成する。
2. 認証された Subject の Principal を変更する。
3. Subject クラスの doAsPrivileged メソッドを呼び出す。

JAAS ポリシー・ファイル **jaas.policy** にも、1 つの許可ブロックが含まれています。

```
grant Principal com.ibm.security.HWPrincipal "bob" {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "foo.txt", "read";
};
```

これらの 3 つの許可は、最初に "bob" という名前の HWPrincipal に認可されます。認証された Subject に加えられる実際の Principal は、ログイン・プロセス時に使用されたユーザー名です。

以下に HelloWorld のアクション・コードを示します。3 つのシステム呼び出し (許可が必要である理由) がボールド体で示されています。

```
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("\nYour java.home property: "
            +System.getProperty("java.home"));

        System.out.println("\nYour user.home property: "
            +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        System.out.println("\nOh, by the way ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignore
        }
        System.out.println("\n\nHello World!\n");
        return null;
    }
}, null);
```

HelloWorld プログラムを実行するときは、さまざまなユーザー名を使用し、それに応じて jaas.policy を変更できます。java2.policy を変更する必要はないはずですが。また、テスト・ディレクトリー内に foo.txt というファイルを作成すれば、最後のシステム呼び出しをテストして、そのファイルに適切なアクセス水準が認可されているかを確認できます。

## HelloWorld ソース・ファイルを調べる

LoginModule クラスの HWLoginModule は、以下の正しいパスワード (スペースを含めた大文字小文字の区別がある) を入力したユーザーを認証するだけです。

### • Go JAAS

セキュリティー・マネージャーとともに実行している場合、成功させるには、すべてのアクセス許可についてユーザー 'bob' を入力しなければなりません。

HelloWorld アプリケーションでは、ユーザーは 3 回までパスワードの入力を試みることができます。Go JAAS が正しく入力されると、ユーザー名と同じ名前を持つ HWPrincipal オブジェクトが、認証された Subject に追加されます。

Principal クラスの HWPrincipal は、入力されたユーザー名に基づく Principal を表します。この名前は、認証された Subject に許可を与える際に重要になります。

メイン・アプリケーションの HelloWorld は、helloWorld という名前の構成項目に基づいて、LoginContext をまず作成します。ユーザー入力を検索するためにコールバックが使用されます。このプロセスを調べるには、HelloWorld.java ファイルにある MyCallbackHandler クラスを参照してください。以下はソース・コードの抜粋です。

```
LoginContext lc = null;
try {
    lc = new LoginContext("helloWorld", new MyCallbackHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}
```

ユーザーがユーザー名とパスワードを (最大 3 回) 入力し、Go JAAS がパスワードとして入力されると、Subject は認証されます (HWLoginModule によって Subject に HWPrincipal が追加されます)。以後の作業は、認証された Subject として実行されます。

ポリシー・ファイルが見つからない場合は、**SecurityException** が出されます。見つかった場合は、java.home および user.home プロパティに関する情報が表示されます。また、テスト・ディレクトリーに foo.txt というファイルがあるかどうかを検査されます。最後に、いたるところに "Hello World" というメッセージが表示されます。

## HelloWorld を楽しむ

好きなだけ HelloWorld を実行してください。以下を試してみることができるでしょう。

- 入力するユーザー名やパスワードを変えてみる
- 構成ファイルの項目を変更する
- ポリシー・ファイルの許可を変更する
- helloWorld 構成項目に LoginModule を追加する
- ポリシー・ファイルにコード・ベース・フィールドを追加する
- セキュリティー・マネージャーなしでプログラムを実行して、問題発生時における動作を調べる

**Java Authentication and Authorization Service SampleThreadSubjectLogin についての説明:**

## ソース・ファイル

SampleThreadSubjectLogin.java ソース・ファイルは、独自のテスト・ディレクトリーに置いてください。このソース・ファイルは、.classes ディレクトリーにコンパイルする必要があります。

使用する HTML ブラウザーの形式にフォーマットされた、このファイルのソース・コードを見るには、例 : JAAS SampleThreadSubjectLoginを参照してください。

## ポリシー・ファイル

JAAS ポリシー・ファイルの一般情報については、API Developers Guideを参照してください。SampleThreadSubjectLogin サンプルに特有のポリシー・ファイルは以下のとおりです。

- threadLogin.config
- threadJava2.policy
- threadJaas.policy

この例をコンパイルして実行する方法については、SampleThreadSubjectLogin.java の最初にある注記を参照してください。

### リンク集

SampleThreadSubjectLogin.java

例: JAAS SampleThreadSubjectLogin

API Developers Guide

この資料は、2000 年 3 月 17 日に最終更新されました。

threadLogin.config

threadJava2.policy

threadJaas.policy

SampleThreadSubjectLogin.java

## IBM Java Generic Security Service (JGSS)

Java Generic Security Service (JGSS) は、認証およびセキュア・メッセージング用の汎用インターフェースを提供します。このインターフェースで、秘密鍵、公開鍵、または他のセキュリティー・テクノロジーをベースにした各種のメカニズムを使用することができます。

基礎となるセキュリティー・メカニズムの複雑さや特性を標準インターフェースで一般化することにより、JGSS はセキュア・ネットワーク・アプリケーションの開発に以下の利点を提供します。

- 単一の抽象インターフェースを利用するアプリケーションを開発することができる。
- 変更を加えることなく、セキュリティー・メカニズムの異なるアプリケーションを使用することができる。

JGSS は Generic Security Service Application Programming Interface (GSS-API) 用の Java バインディングを定義しますが、その API は Internet Engineering Task Force (IETF) によって標準化され、X/Open Group によって採用されている 暗号 API です。

IBM JGSS インプリメンテーションは IBM JGSS と呼ばれています。IBM JGSS は基礎となるデフォルト・セキュリティー・システムとして Kerberos V5 を使用する GSS-API フレームワークのインプリメンテーションです。それはまた、Kerberos 信任状を作成し使用するための Java Authentication and

Authorization Service (JAAS) ログイン・モジュールとしても機能します。さらに、それらの信任状を使用する場合、JGSS に JAAS 権限を実行させることもできます。

IBM JGSS には、ネイティブ iSeries JGSS プロバイダー、Java JGSS プロバイダー、および Kerberos 信任状管理ツール (kinit、ktab、および klist) の Java バージョンが組み込まれています。

注: ネイティブ iSeries JGSS プロバイダーは、ネイティブ iSeries Network Authentication Services (NAS) ライブラリーを使用します。ネイティブ・プロバイダーを使用する場合、ネイティブ iSeries Kerberos ユーティリティーを使用しなければなりません。詳しくは、JGSS プロバイダー を参照してください。

J2SDK Security enhancement from Sun Microsystems, Inc.

Internet Engineering Task Force (IETF) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1

IETF RFC 2853 Generic Security Service API Version 2: Java Bindings

The X/Open Group GSS-API Extensions for DCE

## JGSS の概念

JGSS の操作は、Generic Security Service Application Programming Interface (GSS-API) によって標準化されているように、4 つの異なる段階で構成されています。

この段階は次のとおりです。

1. プリンシパル用信任状の収集。
2. 対等プリンシパル通信間のセキュリティ・コンテキストの作成および設定。
3. 対等間のセキュア・メッセージの交換。
4. リソースのクリーンアップおよびリリース。

さらに、JGSS は Java Cryptographic Architecture を活用し、異なるセキュリティ・メカニズムのシームレスなプラグを可能にします。

以下のリンクから、これら重要な JGSS の概念についての高水準の説明を読むことができます。

- プリンシパルおよび信任状
- コンテキストの設定
- メッセージ保護および交換
- リソースのクリーンアップおよび解放
- セキュリティ・メカニズム

### プリンシパルおよび信任状:

アプリケーションが対等機能と JGSS セキュア通信を行うための ID をプリンシパルと呼びます。プリンシパルは、実ユーザー、または自動サービスの場合もあります。プリンシパルはそのメカニズムのもとで、ID を証明するものとしてセキュリティ・メカニズム特定信任状を獲得します。

たとえば、Kerberos メカニズムを使用する場合、プリンシパルの信任状は Kerberos 鍵配布センター (KDC) によって発行されるチケット許可チケットの形式をとります。マルチ・メカニズム環境では、GSS-API 信任状は複数の信任状エレメントを含むことができ、各エレメントは 1 つの基礎となるメカニズム信任状を表します。

GSS-API 規格では、プリンシパルが信任状を獲得する方法は規定されておらず、GSS-API インプリメンテーションは信任状の獲得手段を提供しないのが普通です。プリンシパルは GSS-API を使用する前に信任状を得ます。GSS-API はプリンシパルのために信任状を得るセキュリティー・メカニズムを単に照会するにすぎません。

IBM JGSS には、Java 版の Kerberos 信任状管理ツール 347 ページの『com.ibm.security.krb5.internal.tools Class Kinit』、349 ページの『com.ibm.security.krb5.internal.tools Class Ktab』、および

『com.ibm.security.krb5.internal.tools Class Klist』が組み込まれています。さらに、IBM JGSS は、JAAS を使用するオプションの Kerberos ログイン・インターフェースを提供することによって標準 GSS-API を強化します。純粹の Java JGSS プロバイダーはオプションのログイン・インターフェースをサポートしていますが、ネイティブ iSeries プロバイダーはサポートしていません。詳しくは、以下のトピックを参照してください。

- Kerberos 信任状の取得
- JGSS プロバイダー

#### ***com.ibm.security.krb5.internal.tools Class Klist:***

このクラスは、信任状キャッシュおよびキー・タブにある項目をリストするためのコマンド行ツールとして実行できます。

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Klist
```

```
public class Klist
extends java.lang.Object
```

このクラスは、信任状キャッシュおよびキー・タブにある項目をリストするためのコマンド行ツールとして実行できます。

### **コンストラクターの要約**

Klist()

### **メソッドの要約**

|             |  |
|-------------|--|
| static void | main(java.lang.String[] args)<br>コマンド行で呼び出すことができるメインプログラム。 |
|-------------|--|

クラス `java.lang.Object` から継承されるメソッド

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

### **コンストラクターの詳細**

**Klist**

```
public Klist()
```

## メソッドの詳細

### main

```
public static void main(java.lang.String[] args)
```

コマンド行で呼び出すことができるメインプログラム。

使用法: `java com.ibm.security.krb5.tools.Klist [[-c] [-f] [-e] [-a]] [-k [-t] [-K]] [name]`

信任状キャッシュに使用可能なオプション:

- **-f** は信任状フラグを示します
- **-e** は暗号化タイプを示します
- **-a** はアドレス・リストを表示します

キー・タブに使用可能なオプション:

- **-t** はキー・タブ項目のタイム・スタンプを示します
- **-K** はキー・タブ項目の DES キーを示します

### *com.ibm.security.krb5.internal.tools Class Kinit:*

Kerberos v5 チケットを取得するための Kinit ツール。

```
java.lang.Object  
|  
+--com.ibm.security.krb5.internal.tools.Kinit
```

```
public class Kinit  
extends java.lang.Object
```

Kerberos v5 チケットを取得するための Kinit ツール。

## コンストラクターの要約

```
Kinit(java.lang.String[] args)
```

新規の Kinit オブジェクトを構成します。

## メソッドの要約

|             |  |
|-------------|--|
| static void | <code>main(java.lang.String[] args)</code><br>メイン・メソッドは、チケット要求のためのユーザー・コマンド行入力を受け入れるために使用されます。 |
|-------------|--|

クラス `java.lang.Object` から継承されるメソッド

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

## コンストラクターの詳細

### Kinit



```
public Kinit(java.lang.String[] args)
    throws java.io.IOException,
           RealmException,
           KrbException
```

新規の Kinit オブジェクトを構成します。

#### パラメーター:

args - チケット要求オプションの配列。使用可能なオプションは、-f、-F、-p、-P、-c、-k、principal、password です。

#### スロー:

java.io.IOException - 入出力エラーが発生した場合。  
RealmException - レalmをインスタンス化できなかった場合。  
KrbException - Kerberos 操作中にエラーが発生した場合。

## メソッドの詳細

### main

```
public static void main(java.lang.String[] args)
```

メイン・メソッドは、チケット要求のためのユーザー・コマンド行入力を受け入れるために使用されます。

使用法: java com.ibm.security.krb5.tools.Kinit [-f] [-F] [-p] [-P] [-k] [-c cache name] [principal] [password]

- -f 転送可能
- -F 転送不可
- -p プロキシ可能
- -P プロキシ不可
- -c キャッシュ名 (すなわち、FILE:d:¥temp¥mykrb5cc)
- -k キー・タブを使用する
- -t キー・タブ・ファイル名
- principal プリンシパル名 (すなわち、qwedf qwedf@IBM.COM)
- password プリンシパルの Kerberos パスワード

java com.ibm.security.krb5.tools.Kinit -help を使用して、ヘルプ・メニューを立ち上げます。

現在、ファイル・ベースの信任状キャッシュのみサポートされています。デフォルトでは、KDC から取得したチケットを保管するために、krb5cc\_{user.name} という名前のキャッシュ・ファイルが {user.home} ディレクトリに生成されます。たとえば、Windows NT では、c:¥winnt¥profiles¥qwedf¥krb5cc\_qwedf になります。ここで、qwedf は {user.name} で、c:¥winnt¥profile¥qwedf は {user.home} です。{user.home} は、Kerberos によって Java システム・プロパティー "user.home" から取得されます。時として、{user.home} がヌルである場合 (ほとんどない)、キャッシュ・ファイルはプログラムが実行している現行ディレクトリに保管されます。{user.name} は、オペレーティング・システムのログイン・ユーザー名です。これは、ユーザーのプリンシパル名と異なっている可能性があります。1 人のユーザーが複数のプリンシパル名を使用できますが、信任状キャッシュの 1 次プリンシパルにできるのは 1 つだけです。これは 1 つのキャッシュ・ファイルが 1 つの特定のユーザー・プリンシパル用のチケットしか保管できないことを意味します。ユーザーが次の Kinit でプリンシパル名を交換する場合、デフォルトで、新規のチケット用に生

成されるキャッシュ・ファイルが古いキャッシュ・ファイルを上書きします。上書きされないようにするには、新規のチケットを要求するときに、別のディレクトリーまたは別のキャッシュ・ファイルを指定する必要があります。

### キャッシュ・ファイルの場所

ユーザー固有のキャッシュ・ファイルの名前および場所を指定するには、いくつかの方法があります。Kerberos が検索する順序は以下に示すとおりです。

1. **-c** オプション。 `java com.ibm.security.krb5.tools.Kinit -c FILE:<ユーザー固有のディレクトリーおよびファイル名>`。 "FILE:" は、信任状キャッシュのタイプを識別するための接頭部です。デフォルトはファイル・ベースのタイプです。
2. 実行時に、`-DKRB5CCNAME=FILE:<ユーザー固有のディレクトリーおよびファイル名>` を使用して、Java システム・プロパティー "KRB5CCNAME" を設定します。
3. 実行時の前に、環境変数 "KRB5CCNAME" をコマンド・プロンプトで設定します。オペレーティング・システムが異なると、環境変数を設定する方法も異なります。たとえば、Windows では `set KRB5CCNAME=FILE:<ユーザー固有のディレクトリーおよびファイル名>` を使用しますが、UNIX では `export KRB5CCNAME=FILE:<ユーザー固有のディレクトリーおよびファイル名>` を使用します。Kerberos では、システム固有のコマンドに依存して環境変数を検索することに注意してください。UNIX で使用されるコマンドは、`"/usr/bin/env"` です。

KRB5CCNAME には大/小文字の区別があり、すべて大文字です。

KRB5CCNAME が上記の説明のとおり設定されていない場合、デフォルトのキャッシュ・ファイルが使用されます。デフォルト・キャッシュは、次の順序で位置指定されます。

1. Unix プラットフォームでは `/tmp/krb5cc_<uid>`。ここで、`<uid>` は Kinit JVM を稼働しているユーザーのユーザー ID です。
2. `<user.home>/krb5cc_<user.name>`。ここで、`<user.home>` および `<user.name>` はそれぞれ Java `user.home` および `user.name` プロパティーです。
3. `<user.home>/krb5cc` (`<user.name>` を JVM から取得できない場合)

### KDC 通信タイムアウト

Kinit は、信任状であるチケット許可チケットを獲得するために鍵配布センター (KDC) と通信します。この通信は、KDC が一定の期間内に応答しない場合にタイムアウトするように設定できます。タイムアウト期間は、Kerberos 構成ファイル内の `libdefaults` スタンザ (すべての KDC に適用可能) または個々の KDC スタンザで (ミリ秒単位で) 設定できます。デフォルトのタイムアウト値は 30 秒です。

#### *com.ibm.security.krb5.internal.tools Class Ktab:*

このクラスは、ユーザーがキー・テーブル内の項目を管理するのを助けるためのコマンド行ツールとして実行できます。使用可能な関数には、`list/add/update/delete` サービス・キーが含まれます。

```
java.lang.Object
|
+--com.ibm.security.krb5.internal.tools.Ktab

public class Ktab
extends java.lang.Object
```

このクラスは、ユーザーがキー・テーブル内の項目を管理するのを助けるためのコマンド行ツールとして実行できます。使用可能な関数には、list/add/update/delete サービス・キーが含まれます。

## コンストラクターの要約

Ktab()

## メソッドの要約

|             |  |
|-------------|--|
| static void | main(java.lang.String[] args)<br>コマンド行で呼び出すことができるメインプログラム。 |
|-------------|--|

クラス `java.lang.Object` から継承されるメソッド

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

## コンストラクターの詳細

**Ktab**

public Ktab()

## メソッドの詳細

**main**

public static void main(java.lang.String[] args)

コマンド行で呼び出すことができるメインプログラム。

使用法: `java com.ibm.security.krb5.tools.Ktab <オプション>`

Ktab に使用可能なオプション:

- **-l** キー・タブ名および項目をリストします
- **-a** *<principal name>*(*<password>*) 項目をキー・タブに追加します
- **-d** *<principal name>* 項目をキー・タブから削除します
- **-k** *<keytab name>* キー・タブ名およびパスに接頭部 `FILE:` を付けて指定します
- **-help** 指示を表示します

## コンテキストの設定:

セキュリティー信任状を獲得した後、2 つの通信対等機能はその信任状を使用してセキュリティー・コンテキストを設定します。2 つの対等機能は 1 つの結合コンテキストを設定しますが、各対等機能はそのコンテキストの自分自身のローカル・コピーを維持します。コンテキストの設定には、開始側ピアが受入側ピアに対して、自分自身を認証することが含まれます。イニシエーターは相互参照の要求を選択することができますが、その場合、アクセプターは自身をイニシエーターに認証します。

コンテキストの設定が完了する際、設定されたコンテキストは 2 つの対等機能間でのその後のセキュア・メッセージ交換を可能にする状態情報 (共有暗号鍵など) を作成します。

## メッセージの保護および交換:

コンテキストが設定されると、2 つの対等機能のセキュア・メッセージ交換が可能になります。メッセージの発信元は、メッセージをエンコードするためのローカル GSS-API インプリメンテーションを呼び出します。これにより、メッセージの保全性を確保することができ、ある場合はメッセージの機密性を確保することができます。その後、アプリケーションは対等機能に結果トークンを送ります。

対等機能のローカル GSS-API インプリメンテーションは、設定されたコンテキストからの情報を以下の方法で使用します。

- メッセージの保全性を検証する
- メッセージを暗号解読する (メッセージが暗号化されている場合)

#### リソースのクリーンアップおよび解放:

リソースを解放するために、JGSS アプリケーションは不要なコンテキストを削除します。JGSS アプリケーションは削除されたコンテキストにアクセスすることはできませんが、メッセージ交換のためにそれを使用すると例外が発生します。

#### セキュリティー・メカニズム:

GSS-API は、1 つ以上の基礎となるセキュリティー・メカニズムの抽象フレームワークから構成されています。どのようにフレームワークが基礎となるセキュリティー・メカニズムと相互作用するかは、インプリメンテーションによって異なります。

そのようなインプリメンテーションは 2 つの一般カテゴリーに分けられます。

- 一方のモノリシックなインプリメンテーションは、フレームワークを単一のメカニズムに強力にバインドします。この種類のインプリメンテーションでは、他のメカニズムや同じメカニズムの別のインプリメンテーションでさえ使用できなくなります。
- もう一方の高度なモジュラー・インプリメンテーションは、使いやすさと柔軟性を提供します。この種類のインプリメンテーションでは、別のセキュリティー・メカニズムとそのインプリメンテーションをフレームワークにシームレスかつ容易に結合することができます。

IBM JGSS は後者のカテゴリーに属します。モジュラー・インプリメンテーションとして、IBM JGSS は Java Cryptographic Architecture (JCA) によって定義されたプロバイダー・フレームワークを活用し、すべての基本メカニズムを (JCA) プロバイダーとして取り扱います。JGSS プロバイダーは JGSS セキュリティー・メカニズムの具体的なインプリメンテーションを提供します。アプリケーションは複数のメカニズムをインスタンス化して使用します。

プロバイダーが複数のメカニズムをサポートするのは可能であり、JGSS は異なるセキュリティー・メカニズムを簡単に使用できるようにします。しかし、GSS-API は、複数のメカニズムが利用可能な場合に、2 つの通信対等機能がメカニズムを選択する手段を提供しません。メカニズムを選択する 1 つの方法は、Simple And Protected GSS-API Negotiating Mechanism (SPNEGO) で開始することです。これは、2 つの対等機能間の実際のメカニズムを折衝する疑似メカニズムです。IBM JGSS には SPNEGO メカニズムは含まれていません。

SPNEGO について詳しくは、Internet Engineering Task Force (IETF) RFC 2478 The Simple and Protected GSS-API Negotiation Mechanism をご覧ください。

### IBM JGSS を使用するように iSeries サーバーを構成する

JGSS を使用するように iSeries サーバーを構成する方法は、ご使用のサーバー上で稼働している Java 2 Software Development Kit (J2SDK) のバージョンによって異なります。

## J2SDK バージョン 1.3 で JGSS を使用するように iSeries サーバーを構成する:

ご使用の iSeries サーバー上で Java 2 Software Development Kit (J2SDK) バージョン 1.3 を使用する場  
合、JGSS を使用できるようにサーバーを準備し構成する必要があります。デフォルトの構成は、純粋の  
Java JGSS プロバイダーを使用します。

### ソフトウェア要件

J2SDK バージョン 1.3 で JGSS を使用するためには、ご使用のサーバーに Java Authentication and  
Authorization Service (JAAS) 1.3 がインストールされていなければなりません。

### JGSS を使用するようにサーバーを構成する

J2SDK バージョン 1.3 で JGSS を使用するようにサーバーを構成するためには、ibmjgssprovider.jar ファ  
イル用の拡張ディレクトリーへのシンボリック・リンクを追加します。ibmjgssprovider.jar ファイルには、  
JGSS クラスおよび純粋の Java JGSS プロバイダーが含まれています。シンボリック・リンクを追加する  
と、拡張クラス・ローダーは ibmjgssprovider.jar ファイルをロードできるようになります。

#### シンボリック・リンクを追加する

シンボリック・リンクを追加するためには、iSeries コマンド行に以下のコマンドを入力し (1 行で)、  
**ENTER** を押してください。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar')  
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/ibmjgssprovider.jar')
```

注: iSeries サーバーのデフォルト Java 1.3 ポリシーは、JGSS に適切な許可を与えます。独自の  
java.policy ファイルを作成する計画をお持ちの場合、ibmjgssprovider.jar を付与するアクセス権につい  
ては JVM アクセス権をご覧ください。

### JGSS プロバイダーの変更

JGSS はデフォルトとして純粋の Java プロバイダーを使用しますが、JGSS を使用するようにサーバーを  
構成した後、ネイティブ iSeries JGSS プロバイダーを使用するように JGSS を構成することができます。  
ネイティブ・プロバイダーを使用するように JGSS を構成した後は、2 つのプロバイダーを容易に切り替  
えることができるようになります。詳しくは、以下のトピックを参照してください。

- JGSS プロバイダー
- ネイティブ iSeries JGSS プロバイダーを使用するように JGSS を構成する

### セキュリティー・マネージャー

使用可能な Java セキュリティー・マネージャーで IBM JGSS アプリケーションを実行する場合、セキュ  
リティー・マネージャーの使用を参照してください。

#### ネイティブ iSeries JGSS プロバイダーを使用するように JGSS を構成する:

IBM JGSS は純粋の Java プロバイダーをデフォルトで使用します。オプションによってネイティブ  
iSeries JGSS プロバイダーを使用します。

別のプロバイダーについて詳しくは、JGSS プロバイダーを参照してください。

### ソフトウェア要件



ネイティブ iSeries JGSS プロバイダーは IBM Toolbox for Java のクラスにアクセスできなければなりません。IBM Toolbox for Java へのアクセス法については、354 ページの『ネイティブ iSeries JGSS プロバイダーが IBM Toolbox for Java にアクセスできるようにする』を参照してください。

ネットワーク認証サービスを構成していることを確認してください。詳しくは、ネットワーク認証サービスを参照してください。

### ネイティブ iSeries JGSS プロバイダーを指定する

J2SDK バージョン 1.3 でネイティブ iSeries JGSS プロバイダーを使用する前に、サーバーが JGSS を使用できるように構成されていることを確認してください。詳しくは、J2SDK バージョン 1.3 で JGSS を使用するように iSeries サーバーを構成するを参照してください。J2SDK バージョン 1.4 または後続のバージョンを使用している場合、JGSS はすでに構成されています。

注: 次の指示では、`{java.home}` は、サーバー上で使用している Java のバージョンの位置へのパスを示しています。たとえば、J2SDK バージョン 1.4 を使用している場合、`{java.home}` は `/QIBM/ProdData/Java400/jdk14` です。コマンド内の `{java.home}` を、Java ホーム・ディレクトリーへの実際のパスに置換することを忘れないでください。

ネイティブ iSeries JGSS プロバイダーを使用するように JGSS を構成するためには、以下のタスクを完了してください。

### シンボリック・リンクを追加する

`ibmjgssiseriesprovider.jar` ファイル用の拡張ディレクトリーへのシンボリック・リンクを追加するためには、iSeries コマンド行に以下のコマンドを入力し (1 行で)、**ENTER** を押してください。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssiseriesprovider.jar')
NEWLNK('${java.home}/lib/ext/ibmjgssiseriesprovider.jar')
```

`ibmjgssiseriesprovider.jar` ファイル用の拡張ディレクトリーへのシンボリック・リンクを追加した後、拡張クラス・ローダーが JAR ファイルをロードします。

### プロバイダーをセキュリティー・プロバイダー・リストに追加する

`java.security` ファイルのセキュリティー・プロバイダー・リストにネイティブ・プロバイダーを追加します。

1. 編集のため、`{java.home}/lib/security/java.security` を開きます。
2. セキュリティー・プロバイダー・リストを検索します。それは、`java.security` ファイルの最初の方に以下のような名前が存在するはずで

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
```

3. オリジナル Java プロバイダーより前になるようにセキュリティー・プロバイダー・リストにネイティブ iSeries JGSS プロバイダーを追加します。言い換えれば、`com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider` を `com.ibm.jgss.IBMJGSSProvider` より小さい番号でリストに追加し、その後、`IBMJGSSProvider` の位置を更新します。以下に例を示します。

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider
security.provider.5=com.ibm.security.jgss.IBMJGSSProvider
```

IBMJGSSiSeriesProvider は 4 番目に、IBMJGSSProvider は 5 番目にそれぞれ入力されていることに注意してください。さらに、セキュリティー・プロバイダー・リストの入力番号が連続しており、入力番号が 1 つずつ増加していることをチェックしてください。

4. java.security file を保管してクローズしてください。

**ネイティブ iSeries JGSS プロバイダーが IBM Toolbox for Java にアクセスできるようにする:** ネイティブ iSeries JGSS プロバイダーは IBM Toolbox for Java のクラスにアクセスできなければなりません。以下のいずれかのオプションを使用して、IBM JGSS が、Toolbox for Java jt400Native.jar ファイルにアクセスできるようにしてください。

- Java 拡張ディレクトリーの jt400Native.jar にシンボリック・リンクを置きます。
- 自分の拡張ディレクトリーの jt400Native.jar にシンボリック・リンクを置き、このディレクトリーをディレクトリー・リストに組み込みます。

注:

- Java 拡張ディレクトリーの jt400Native.jar にシンボリック・リンクを置くと、J2SDK のすべてのユーザーが、このバージョンの jt400Native.jar で実行することになります。さまざまなユーザーが、さまざまなバージョンの IBM Toolbox for Java クラスを要求する場合、この方法は望ましくありません。
- 次の指示では、`{java.home}` は、サーバー上で使用している Java のバージョンの位置へのパスを示しています。たとえば、J2SDK バージョン 1.4 を使用している場合、`{java.home}` は `/QIBM/ProdData/Java400/jdk14` です。コマンド内の `{java.home}` を、Java ホーム・ディレクトリーへの実際のパスに置換することを忘れないでください。

## Java 拡張ディレクトリーにシンボリック・リンクを置く

Java 拡張ディレクトリーの jt400Native.jar ファイルにリンクを置くには、iSeries コマンド行で、次のコマンドを (1 行で) 入力し、**ENTER** キーを押してください。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('${java.home}/lib/ext/jt400Native.jar')
```

## 自分の拡張ディレクトリーにシンボリック・リンクを置く

自分のディレクトリーの jt400Native.jar ファイルにリンクするには、iSeries コマンド行で、次のコマンドを (1 行で) 入力し、**ENTER** キーを押してください。

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
NEWLNK('<your extension directory>/jt400Native.jar')
```

プログラムを呼び出すときには、次の形式の Java コマンドを使用してください。

```
java -Djava.ext.dirs=<your extension directory>:{java.home}/lib/ext:/QIBM/UserData/Java400/ext
```

## J2SDK バージョン 1.4 または後続のバージョンで JGSS を使用するように iSeries サーバーを構成する:

ご使用の iSeries サーバー上で Java 2 Software Development Kit (J2SDK) バージョン 1.4 またはそれ以上を使用する場合、JGSS はすでに構成済みです。デフォルトの構成は、純粹の Java JGSS プロバイダーを使用します。



## JGSS プロバイダーの変更

純粹の Java JGSS プロバイダーの代わりにネイティブ iSeries JGSS プロバイダーを使用するように、JGSS を構成することができます。ネイティブ・プロバイダーを使用するように JGSS を構成した後は、2 つのプロバイダーを容易に切り替えることができるようになります。詳しくは、以下のトピックを参照してください。

- JGSS プロバイダー
- ネイティブ iSeries JGSS プロバイダーを使用するように JGSS を構成する

## セキュリティ・マネージャー

使用可能な Java セキュリティー・マネージャーで JGSS アプリケーションを実行する場合、セキュリティ・マネージャーの使用を参照してください。

### JGSS プロバイダー:

IBM JGSS には、ネイティブ iSeries JGSS プロバイダーおよび純粹の Java JGSS プロバイダーが含まれています。どのプロバイダーを選択して使用するかは、アプリケーションの必要に応じて異なります。

純粹の Java JGSS プロバイダーには、以下の機能があります。

- アプリケーションに対して最高レベルのポータビリティを保証する。
- オプションの JAAS Kerberos ログイン・インターフェースと連動する。
- Java Kerberos 信任状管理ツールとの互換性。

ネイティブ iSeries JGSS プロバイダーには、以下の機能があります。

- ネイティブ iSeries Kerberos ライブラリーの使用。
- Qshell Kerberos 信任状管理ツールとの互換性。
- JGSS アプリケーションの高速実行。

注: 両方の JGSS プロバイダーは GSS-API 仕様に従っているため、互いに互換性があります。言い換えれば、純粹の Java JGSS プロバイダーを使用するアプリケーションは、ネイティブ iSeries JGSS プロバイダーを使用するアプリケーションとの相互運用が可能であるということです。

## JGSS プロバイダーの変更

注: ご使用のサーバーが J2SDK バージョン 1.3 を実行している場合、ネイティブ iSeries JGSS プロバイダーに変更する前に、ご使用のサーバーが JGSS を使用できるように構成されていることを確認してください。詳しくは、以下のトピックを参照してください。

- J2SDK バージョン 1.3 で JGSS を使用するように iSeries サーバーを構成する
- ネイティブ iSeries JGSS プロバイダーを使用するように JGSS を構成する

以下の方法の 1 つを使用して、JGSS プロバイダーを容易に変更することができます。

- `${java.home}/lib/security/java.security` のセキュリティ・プロバイダー・リストを編集する

注: `${java.home}` がご使用のサーバー上で使用している Java のバージョンのロケーションへのパスを示します。たとえば、J2SDK バージョン 1.3 をお使いの場合、`${java.home}` は `/QIBM/ProdData/Java400/jdk13` です。

- `GSSManager.addProviderAtFront()` または `GSSManager.addProviderAtEnd(Specify)` のどちらかを使用して、JGSS アプリケーションの中でプロバイダー名を指定してください。詳しくは、『`GSSManager javadoc`』を参照してください。

### セキュリティー・マネージャーの使用:

使用可能な Java セキュリティー・マネージャーでご使用の JGSS アプリケーションを実行する場合、アプリケーションおよび JGSS に必要なアクセス権があることを確認する必要があります。

**JVM アクセス権:** JGSS が実行するアクセス制御検査に加え、Java 仮想マシン (JVM) は、ファイル、Java プロパティー、パッケージ、およびソケットを含むさまざまなリソースへのアクセス時に、許可検査を実行します。

JVM 許可の使用に関する詳細は、`Permissions in the Java 2 SDK` を参照してください。

次のリストは、JGSS の JAAS 機能を使用する場合、またはセキュリティー・マネージャーで JGSS を使用する場合に必要な許可をリストします。

- `javax.security.auth.AuthPermission "modifyPrincipals"`
- `javax.security.auth.AuthPermission "modifyPrivateCredentials"`
- `javax.security.auth.AuthPermission "getSubject"`
- `javax.security.auth.PrivateCredentialPermission "javax.security.auth.kerberos.KerberosKey javax.security.auth.kerberos.KerberosPrincipal ¥"*¥", "read"`
- `javax.security.auth.PrivateCredentialPermission "javax.security.auth.kerberos.KerberosTicket javax.security.auth.kerberos.KerberosPrincipal ¥"*¥", "read"`
- `java.util.PropertyPermission "com.ibm.security.jgss.debug", "read"`
- `java.util.PropertyPermission "DEBUG", "read"`
- `java.util.PropertyPermission "java.home", "read"`
- `java.util.PropertyPermission "java.security.krb5.conf", "read"`
- `java.util.PropertyPermission "java.security.krb5.kdc", "read"`
- `java.util.PropertyPermission "java.security.krb5.realm", "read"`
- `java.util.PropertyPermission "javax.security.auth.useSubjectCredsOnly", "read"`
- `java.util.PropertyPermission "user.dir", "read"`
- `java.util.PropertyPermission "user.home", "read"`
- `java.lang.RuntimePermission "accessClassInPackage.sun.security.action"`
- `java.security.SecurityPermission "putProviderProperty.IBMJGSSProvider"`

### JAAS 許可検査:

IBM JGSS は、JAAS が使用可能にするプログラムが信任状を使用し、サービスにアクセスするときに、ランタイム許可検査を実行します。Java プロパティー `javax.security.auth.useSubjectCredsOnly` を `false` に設定することによって、このオプションの JAAS 機能を使用不可能にすることができます。さらに、JGSS は、アプリケーションがセキュリティー・マネージャーを使って実行される場合のみ、許可検査を実行します。

JGSS は、現行アクセス制御コンテキストで有効な Java ポリシーに対して、許可検査を実行します。JGSS は、次の特定の許可検査を実行します。

- `javax.security.auth.kerberos.DelegationPermission`

- `javax.security.auth.kerberos.ServicePermission`

## DelegationPermission 検査

`DelegationPermission` により、セキュリティー・ポリシーは、Kerberos のチケット転送およびプロキシーを行う機能の使用を制御できます。これらの機能を使用して、クライアントは、サービスがクライアントの代わりとして動作することを許可できます。

`DelegationPermission` は、次の順序で 2 つの引数を取ります。

1. 従属プリンシパル。クライアントの代わりに、またクライアントの権限の下で動作する、サービス・プリンシパルの名前。
2. クライアントが従属プリンシパルに使用を許可するサービスの名前。

### 例: DelegationPermission 検査の使用

次の例では、`superSecureServer` が従属プリンシパル、`krbtgt/REALM.IBM.COM@REALM.IBM.COM` が `superSecureServer` にクライアントの代わりに使用を許可するサービスです。この場合、サービスはクライアントのチケット許可チケットです。つまり、`superSecureServer` は、クライアントの代わりにどんなサービスのチケットでも入手できるということです。

```
permission javax.security.auth.kerberos.DelegationPermission
    "¥"superSecureServer/host.ibm.com@REALM.IBM.COM¥"
    ¥"krbtgt/REALM.IBM.COM@REALM.IBM.COM¥";
```

この例では、`DelegationPermission` は、`superSecureServer` だけが使用できる鍵配布センター (KDC) から、新しいチケット許可チケットを入手するためのクライアント許可を認可します。クライアントが新しいチケット許可チケットを `superSecureServer` に送信した後、`superSecureServer` にはクライアントの代わりに動作する機能が付与されます。

次の例では、クライアントが、ftp サービスのみに `superSecureServer` がアクセスすることを許可する新しいチケットを入手できるようにします。

```
permission javax.security.auth.kerberos.DelegationPermission
    "¥"superSecureServer/host.ibm.com@REALM.IBM.COM¥"
    ¥"ftp/ftp.ibm.com@REALM.IBM.COM¥";
```

詳細については、Sun Web サイトの [J2SDK documentation](#) の `javax.security.auth.kerberos.DelegationPermission` クラスについての説明を参照してください。

## ServicePermission 検査

`ServicePermission` は、コンテキストの開始および受け入れのための信任状の使用制限を検査します。コンテキスト開始側には、コンテキストを開始する許可がなければなりません。同様に、コンテキストの受入側にも、コンテキストを受け入れる許可が必要です。

### 例: ServicePermission 検査の使用

次の例は、クライアント・サイドが、許可をクライアントに付与することによって、ftp サービスを使ってコンテキストを開始できるようにします。

```
permission javax.security.auth.kerberos.ServicePermission
    "ftp/host.ibm.com@REALM.IBM.COM", "initiate";
```

次の例は、サーバー・サイドが、許可をサーバーに付与することによって、ftp サービスに秘密鍵を使ってコンテキストを開始できるようにします。

```
permission javax.security.auth.kerberos.ServicePermission
    "ftp/host.ibm.com@REALM.IBM.COM", "accept";
```

詳細については、Sun Web サイトの J2SDK documentation の `javax.security.auth.kerberos.ServicePermission` クラスについての説明を参照してください。

## IBM JGSS アプリケーションの実行

IBM Java Generic Security Service (JGSS) API 1.0 は、セキュア・アプリケーションを、さまざまな基礎となるメカニズムの複雑さおよび特殊さから保護します。JGSS は、Java Authentication and Authorization Service (JAAS) および IBM Java Cryptography Extension (JCE) が提供する機能を使用します。

JGSS 機能には、以下のものが含まれます。

- 識別認証
- メッセージの保全性と機密性
- オプションの JAAS Kerberos ログイン・インターフェースおよび許可検査

### Kerberos 信任状の取得および秘密鍵の作成:

GSS-API は、信任状を取得するための方法を定義していません。このため、IBM JGSS Kerberos メカニズムでは、ユーザーが、Kerberos 信任状を取得することが必要です。このトピックでは、Kerberos 証明書を取得して秘密鍵を作成する方法、および Kerberos ログインおよび許可検査を実行するための JAAS の使用方法を学び、Java 仮想マシン (JVM) が必要とする JAAS 許可のリストを学習できます。

信任状は、以下のいずれかの方式を使用して取得できます。

- Kinit および Ktab ツール
- オプションの JAAS Kerberos ログイン・インターフェース

### Kinit および Ktab ツール:

選択された JGSS プロバイダーは、Kerberos 信任状および秘密鍵を取得するのに使用するツールを決定します。

## 純粋な Java JGSS プロバイダーの使用

純粋な Java JGSS プロバイダーを使用している場合、IBM JGSS Kinit および Ktab ツールを使用して信任状と秘密鍵を取得します。Kinit および Ktab ツールは、コマンド行インターフェースを使用し、他のバージョンが提供するオプションに類似したオプションを提供します。

- Kinit ツールを使用して Kerberos 信任状を取得できます。このツールは、Kerberos 配布センター (KDC) と接触し、チケット許可チケット (TGT) を取得します。TGT によって、GSS-API を使用するものも含め、Kerberos 可能サービスにアクセスできます。
- サーバーは、Ktab ツールを使用して秘密鍵を取得できます。JGSS は、サーバー上のキー・テーブル・ファイルに秘密鍵を保管します。詳細は、Ktab Java に関する資料を参照してください。

別の方法として、アプリケーションは、JAAS ログイン・インターフェースを使って TGT と秘密鍵を取得することができます。詳しくは、以下を参照してください。

- 347 ページの『com.ibm.security.krb5.internal.tools Class Kinit』
- 349 ページの『com.ibm.security.krb5.internal.tools Class Ktab』
- JAAS ログイン・インターフェース

## ネイティブ iSeries JGSS プロバイダーの使用

ネイティブ iSeries JGSS プロバイダーを使用している場合、Qshell kinit および klist ユーティリティを使用します。詳しくは、Kerberos 信任状およびキー・テーブル用のユーティリティを参照してください。

### JAAS Kerberos ログイン・インターフェース:

IBM JGSS は、Java Authentication and Authorizaiton Service (JAAS) Kerberos ログイン・インターフェースを備えています。Java プロパティ `javax.security.auth.useSubjectCredsOnly` を `false` に設定することによって、この機能を使用不可にすることができます。

注: 純粋な Java JGSS プロバイダーはログイン・インターフェースを使用できますが、ネイティブ iSeries JGSS プロバイダーは使用できません。

JAAS について詳しくは、Java Authentication and Authorization Service を参照してください。

## JAAS および JVM 許可

セキュリティー・マネージャーを使用している場合、アプリケーションおよび JGSS に必要な JVM と JAAS 許可があるかどうか確認する必要があります。詳しくは、セキュリティー・マネージャーの使用を参照してください。

## JAAS 構成ファイル・オプション

ログイン・インターフェースには、使用されるログイン・モジュールとして、`com.ibm.security.auth.module.Krb5LoginModule` を指定する JAAS 構成ファイルが必要です。以下の表は、`Krb5LoginModule` がサポートするオプションをリストしています。オプションは大文字小文字を区別しないので注意してください。

| オプション名                        | 値  | デフォルト                      | 説明                              |
|-------------------------------|--|----------------------------|---------------------------------|
| <code>principal</code>        | <code>&lt;string&gt;</code>              | なし; プロンプトに従う               | Kerberos プリンシパル名                |
| <code>credsType</code>        | <code>initiator   acceptor   both</code> | <code>initiator</code>     | JGSS 信任状タイプ                     |
| <code>forwardable</code>      | <code>true false</code>                  | <code>false</code>         | 転送可能チケット許可チケット (TGT) を獲得できるかどうか |
| <code>proxiable</code>        | <code>true false</code>                  | <code>false</code>         | プロキシー可能 TGT を獲得できるかどうか          |
| <code>useCcache</code>        | <code>&lt;URL&gt;</code>                 | <code>ccache</code> を使用しない | 指定された信任状キャッシュから TGT を検索する       |
| <code>useKeytab</code>        | <code>&lt;URL&gt;</code>                 | キー・テーブルを使用しない              | 指定されたキー・テーブルから秘密鍵を検索する          |
| <code>useDefaultCcache</code> | <code>true false</code>                  | デフォルト・キャッシュを使用しない          | デフォルトの信任状キャッシュから TGT を検索する      |
| <code>useDefaultKeytab</code> | <code>true false</code>                  | デフォルト・キー・テーブルを使用しない        | 指定されたキー・テーブルから秘密鍵を検索する          |

`Krb5LoginModule` の簡単な例については、JAAS ログイン構成ファイルのサンプルを参照してください。

### オプションの非互換性

プリンシパル名を除く `Krb5LoginModule` オプションの中には、相互に非互換であるもの、つまり一緒に指定できないものがあります。次の表は、ログイン・モジュール・オプションで、互換性のあるものと非互換であるものを示します。

表中の標識は、2 つの関連したオプションの間の関係を表します。

- X = 非互換
- N/A = 適用不可の組み合わせ
- ブランク = 互換

| Krb5LoginModule オプション | credsType initiator | credsType acceptor | credsType both | forward | proxy | use Ccache | use Keytab | useDefault Ccache | useDefault Keytab |
|-----------------------|---------------------|--------------------|----------------|---------|-------|------------|------------|-------------------|-------------------|
| credsType=initiator   |                     | N/A                | N/A            |         |       |            | X          |                   | X                 |
| credsType=acceptor    | N/A                 |                    | N/A            | X       | X     | X          |            | X                 |                   |
| credsType=both        | N/A                 | N/A                |                |         |       |            |            |                   |                   |
| forwardable           |                     | X                  |                |         |       | X          | X          | X                 | X                 |
| proxiabile            |                     | X                  |                |         |       | X          | X          | X                 | X                 |
| useCcache             |                     | X                  |                | X       | X     |            | X          | X                 | X                 |
| useKeytab             | X                   |                    |                | X       | X     | X          |            | X                 | X                 |
| useDefaultCcache      |                     | X                  |                | X       | X     | X          | X          |                   | X                 |
| useDefaultKeytab      | X                   |                    |                | X       | X     | X          | X          | X                 |                   |

## プリンシパル名オプション

プリンシパル名は、他のどのオプションとも組み合わせて指定できます。プリンシパル名を指定しない場合、Krb5LoginModule は、ユーザーにプリンシパル名を指定するというプロンプトを出します。Krb5LoginModule がユーザーにプロンプトを出すかどうかは、指定する他のオプションに依存します。

### サービス・プリンシパル名のフォーマット

サービス・プリンシパル名を指定するには、次のフォーマットの 1 つを使用してください。

- <service\_name> (たとえば、superSecureServer)
- <service\_name>@<host> (たとえば、superSecureServer@myhost)

後者のフォーマットでは、<host> はサービスが常駐するマシンのホスト名です。完全に修飾されたホスト名を使用できます (必ずしも使用する必要はありません)。

注: JAAS は、特定の文字を区切り文字として認識します。JAAS スtring (プリンシパル名など) で次の文字のいずれかを使用する場合、文字を引用符で囲んでください。

- (下線)
- : (コロン)
- / (スラッシュ)
- ¥ (円記号)

## プリンシパル名とパスワードのプロンプトを出す

JAAS 構成ファイルで指定するオプションは、Krb5LoginModule ログインが非対話式か、対話式かを決定します。

- 非対話式ログインは、どんな情報についてもプロンプトを出しません。
- 対話式ログインは、プリンシパル名、パスワード、またはその両方のプロンプトを出します。

### 非対話式ログイン

ログインは、信任状タイプをイニシエーター (credsType=initiator) として指定し、次のアクションのいずれかを実行する場合に非対話式に進行します。

- useCcache オプションを指定する



- useDefaultCcache オプションを true に設定する

また、信任状タイプをアクセプター、または両方 (credsType=acceptor または credsType=both) に指定し、次のアクションのいずれかを実行する場合に非対話式に進行します。

- useKeytab オプションを指定する
- useDefaultKeytab オプションを true に設定する

## 対話式ログイン

他の構成は、Kerberos KDC から TGT を取得できるように、プリンシパル名およびパスワードのプロンプトを出すログイン・モジュールになります。ログイン・モジュールは、プリンシパル・オプションを指定すると、パスワードのみについてプロンプトを出します。

対話式ログインでは、アプリケーションが、ログイン・コンテキストの作成時にコールバック・ハンドラーとして com.ibm.security.auth.callback.Krb5CallbackHandler を指定することが必要です。コールバック・ハンドラーには、入力を促すプロンプトを出す役割があります。

## 信任状タイプ・オプション

信任状タイプをイニシエーターとアクセプターの両方 (credsType=both) にする必要がある場合、Krb5LoginModule は TGT と秘密鍵の両方を取得します。ログイン・モジュールは、TGT を使ってコンテキストと秘密鍵を開始し、コンテキストを受け入れます。JAAS 構成ファイルには、ログイン・モジュールが 2 つのタイプの信任状を獲得するために十分な情報が入っているはずですが、

信任状タイプ、アクセプターおよび両方については、ログイン・モジュールはサービス・プリンシパルを想定します。

### 構成ファイルとポリシー・ファイル:

JGSS および JAAS は、いくつかの構成ファイルおよびポリシー・ファイルに依存しています。これらのファイルを、環境およびアプリケーションに順応するように編集する必要があります。JGSS で JAAS を使用していない場合、JAAS 構成ファイルおよびポリシー・ファイルを無視しても構いません。

注: 次の指示では、`{java.home}` は、サーバー上で使用している Java のバージョンの位置へのパスを示しています。たとえば、J2SDK バージョン 1.4 を使用している場合、`{java.home}` は `/QIBM/ProdData/Java400/jdk14` です。プロパティー設定の `{java.home}` を、Java ホーム・ディレクトリーへの実際のパスに置換することを忘れないでください。

## Kerberos 構成ファイル

IBM JGSS には、Kerberos 構成ファイルが必要です。Kerberos 構成ファイルのデフォルト名および位置は、使用中のオペレーティング・システムに依存しています。JGSS は、デフォルト構成ファイルを次の順序で検索します。

1. Java プロパティー `java.security.krb5.conf` が参照するファイル
2. `{java.home}/lib/security/krb5.conf`
3. Microsoft® Windows プラットフォームの `c:\¥winnt¥krb5.ini`
4. Solaris プラットフォームの `/etc/krb5/krb5.conf`
5. UNIX プラットフォームの `/etc/krb5.conf`



## JAAS 構成ファイル

JAAS ログイン機能の使用には、JAAS 構成ファイルが必要です。以下のプロパティの 1 つを設定することにより、JAAS 構成ファイルを指定できます。

- Java システム・プロパティ `java.security.auth.login.config`
- `${java.home}/lib/security/java.security` ファイルのセキュリティー・プロパティ `login.config.url.<integer>`

詳しくは、Sun Java Authentication and Authorization Service (JAAS) Web サイトを参照してください。

## JAAS ポリシー・ファイル

デフォルト・ポリシー・インプリメンテーションを使用すると、ポリシー・ファイルにアクセス権を記録することによって、JGSS は JAAS アクセス権をエンティティーに付与します。以下のプロパティの 1 つを設定することにより、JAAS ポリシー・ファイルを指定できます。

- Java システム・プロパティ `java.security.policy`
- `${java.home}/lib/security/java.security` ファイルのセキュリティー・プロパティ `property.policy.url.<integer>`

J2SDK バージョン 1.4 および後続のリリースを使用している場合、JAAS への個別のポリシー・ファイルの指定はオプションです。J2SDK バージョン 1.4 およびそれ以降のデフォルト・ポリシー・プロバイダーは、JAAS が必要とするポリシー・ファイル・エントリーをサポートします。

詳しくは、Sun Java Authentication and Authorization Service (JAAS) Web サイトを参照してください。

## Java マスター・セキュリティー・プロパティ・ファイル

Java 仮想マシン (JVM) は、多数の重要なセキュリティー・プロパティを使用します。これらのセキュリティー・プロパティは、Java マスター・セキュリティー・プロパティ・ファイルを編集することによって設定されます。このファイルの名前は `java.security` で、通常 `iSeries` サーバー上の `${java.home}/lib/security` ディレクトリーにあります。

次のリストは、JGSS を使用するためのいくつかの関連セキュリティー・プロパティを説明します。この説明は、`java.security` ファイルを編集するためのガイドとして使用してください。

注: 適用可能な場合は、説明には JGSS サンプルを実行するのに必要な適切な値が含まれます。

**security.provider.<integer>**: 使用する予定の JGSS プロバイダー。これは静的に暗号プロバイダー・クラスの登録も行います。IBM JGSS は、IBM JCE プロバイダーが提供する暗号および他のセキュリティー・サービスを使用します。次の例のとおり、`sun.security.provider.Sun` および `com.ibm.crypto.provider.IBMJCE` パッケージを指定してください。

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

**policy.provider**: システム・ポリシー・ハンドラー・クラス。以下に例を示します。

```
policy.provider=sun.security.provider.PolicyFile
```

**policy.url.<integer>**: ポリシー・ファイルの URL。サンプル・ポリシー・ファイルを使用するには、次のようなエントリーを含めます。

```
policy.url.1=file:/home/user/jgss/config/java.policy
```

**login.configuration.provider**: JAAS ログイン構成ハンドラー・クラス。次に例を示します。

```
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
```

**auth.policy.provider:** JAAS プリンシパルに基づくアクセス制御ポリシー・ハンドラー・クラス。次に例を示します。

```
auth.policy.provider=com.ibm.security.auth.PolicyFile
```

**login.config.url.<integer>:** JAAS ログイン構成ファイルの URL。サンプル構成ファイルを使用するには、次のようなエントリーを含めます。

```
login.config.url.1=file:/home/user/jgss/config/jaas.conf
```

**auth.policy.url.<integer>:** JAAS ポリシー・ファイルの URL。プリンシパルに基づく構成と、CodeSource に基づく構成の両方を JAAS ポリシー・ファイルに含めることができます。サンプル・ポリシー・ファイルを使用するには、次のようなエントリーを含めます。

```
auth.policy.url.1=file:/home/user/jgss/config/jaas.policy
```

## 信任状キャッシュおよびサーバー・キー・テーブル

ユーザー・プリンシパルは、その Kerberos 信任状を信任状キャッシュに保持します。サービス・プリンシパルは、秘密鍵をキー・テーブルに保持します。実行時に、IBM JGSS は次の方法でこれらのキャッシュを見つけます。

### ユーザー信任状キャッシュ

JGSS は、次の順序でユーザー信任状キャッシュを探します。

1. Java プロパティ `KRB5CCNAME` が参照するファイル
2. 環境変数 `KRB5CCNAME` が参照するファイル
3. UNIX システムの `/tmp/krb5cc_<uid>`
4. `${user.home}/krb5cc_${user.name}`
5. `${user.home}/krb5cc` (`${user.name}` が取得できない場合)

### サーバー・キー・テーブル

JGSS は、次の順序でサーバー・キー・テーブル・ファイルを探します。

1. Java プロパティ `KRB5_KTNAME` の値
2. Kerberos 構成ファイルの `libdefaults` スタンザにある `default_keytab_name` エントリー
3. `${user.home}/krb5_keytab`

## IBM JGSS アプリケーションの開発

セキュア・アプリケーションを開発するには、JGSS を使用します。トランスポート・トークンの生成、JGSS オブジェクトの作成、コンテキストの設定などについて学習します。

JGSS アプリケーションを開発するには、高水準 GSS-API 仕様および Java バインディング仕様を熟知している必要があります。IBM JGSS 1.0 は、主にこれらの仕様に基づき、準拠しています。詳細は、以下のリンクを参照してください。

- RFC 2743: Generic Security Service Application Programming Interface Version 2, Update 1
- RFC 2853: Generic Security Service API Version 2: Java Bindings

### IBM JGSS アプリケーション・プログラミング・ステップ:

トランスポート・トークンの使用、必要な JGSS オブジェクトの作成、コンテキストの設定と削除、およびメッセージごとのサービスの使用を含む、JGSS アプリケーションの開発に必要な複数のステップがあります。

JGSS アプリケーションの操作は、Generic Security Service Application Programming Interface (GSS-API) 操作可能モデルに従います。JGSS 操作に重要な概念については、JGSS の概念を参照してください。

## JGSS トランスポート・トークン

重要な JGSS 操作のいくつかは、Java バイト配列の形式でトークンを生成します。1 つの JGSS 対等機能から別の対等機能にトークンを転送するのは、アプリケーションの役割です。JGSS は、アプリケーションがトランスポート・トークンに使用するプロトコルを、何らかの方法で抑制することはありません。アプリケーションは、JGSS トークンを他のアプリケーション (つまり JGSS でない) のデータと一緒にトランスポートすることがあります。しかし、JGSS は JGSS 特有のトークンだけを受け入れ、使用します。

## JGSS アプリケーションの操作順序

JGSS 操作では、以下のリストにある順序で使用する必要がある、特定のプログラミング構成を必要とします。各ステップは、イニシエーターとアクセプターの両方に適用されます。

注: 情報には、高水準 JGSS API の使用を例示し、アプリケーションが `org.ietf.jgss` パッケージをインポートすることを前提とするサンプル・コードの断片が含まれます。多くの高水準 API が過負荷ですが、断片ではこれらのメソッドのうち最もよく使用される形式のみを示します。もちろん、ご自分の必要に最適の API メソッドを使用してください。

### 1. GSSManager の作成

GSSManager のインスタンスは、他の JGSS オブジェクト・インスタンスを作成するためのファクトリーとして動作します。

### 2. GSSName の作成

GSSName は、JGSS プリンシパルの ID を表します。JGSS 操作の中には、ヌル GSSName を指定すると、デフォルト・プリンシパルを見つけて使用できるものがあります。

### 3. GSSCredential の作成

GSSCredential は、プリンシパルのメカニズム特有の信任状を具体化します。

### 4. GSSContext の作成

GSSContext は、コンテキストの設定、および後続するメッセージごとのサービスに使用されます。

### 5. コンテキスト上でオプションのサービスを選択する

アプリケーションは、相互認証などのオプション・サービスを、明示的に要求する必要があります。

### 6. コンテキストの設定

イニシエーターは、それ自体をアクセプターに認証します。しかし、相互認証の要求時には、アクセプターがそれ自体をイニシエーターに認証します。

### 7. メッセージごとのサービスの使用

イニシエーターとアクセプターが、設定されたコンテキストを介してセキュア・メッセージを交換します。

### 8. コンテキストの削除

アプリケーションは、もう必要ないコンテキストを削除します。

### **GSSManager の作成:**

GSSManager 抽象クラスは、JGSS オブジェクトの作成のため、ファクトリーとしての役割を果たします。

GSSManager 抽象クラスは以下を作成します。

- GSSName
- GSSCredential
- GSSContext

また、GSSManager には、サポートされるセキュリティー・メカニズムおよび名前タイプを決定し、JGSS プロバイダーを指定するメソッドもあります。GSSManager getInstance 静的メソッドを使用して、デフォルトの GSSManager のインスタンスを作成してください。

```
GSSManager manager = GSSManager.getInstance();
```

### **GSSName の作成:**

GSSName は、GSS-API プリンシパルの ID を表します。GSSName には、サポートされる基礎となるメカニズムごとに 1 つずつ、プリンシパルのたくさんの表記が含まれることがあります。名前の表記のみを含む GSSName は、メカニズム名 (MN) と呼ばれます。

GSSManager には、ストリングまたはバイトの連続する配列から GSSName を作成するためのいくつかの過負荷メソッドがあります。メソッドは、指定された名前タイプに従ってストリングまたはバイト配列を解釈します。通常、GSSName バイト配列メソッドを使用して、エクスポートされた名前を再構成します。エクスポートされた名前は、通常タイプ GSSName.NT\_EXPORT\_NAME のメカニズム・タイプです。これらのメソッドのいくつかによって、名前の作成に使用するセキュリティー・メカニズムを指定することができます。

### **例: GSSName の使用**

次の基本コードの断片は、GSSName の使用方法を示します。

**注:** Kerberos サービス名ストリングを、<service> または <service@host> のどちらかとして指定します。<service> はサービスの名前、<host> はサービスが実行されるマシンのホスト名です。完全に修飾されたホスト名を使用できます (必ずしも使用する必要はありません)。ストリングの @<host> 部分を省略すると、GSSName はローカル・ホスト名を使用します。

```
// Create GSSName for user foo.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);

// Create a Kerberos V5 mechanism name for user foo.
Oid krb5Mech = new Oid("1.2.840.113554.1.2.2");
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME, krb5Mech);

// Create a mechanism name from a non-mechanism name by using the GSSName
// canonicalize method.
GSSName fooName = manager.createName("foo", GSSName.NT_USER_NAME);
GSSName fooKrb5Name = fooName.canonicalize(krb5Mech);
```

### **GSSCredential の作成:**

GSSCredential には、プリンシパルの代わりにコンテキストを作成するのに必要なすべての暗号情報が含まれており、複数のメカニズム用の信任状情報も入っていることがあります。

GSSManager には、3 つの信任状作成メソッドがあります。メソッドのうち 2 つは、GSSName、信任状の存続時間、信任状入手元の 1 つ以上のメカニズム、および信任状使用タイプのパラメーターを取ります。3 番目のメソッドは、使用タイプだけを取り、他のパラメーターに関してはデフォルト値を使用します。ヌル・メカニズムの指定でも、デフォルトのメカニズムを使用します。メカニズムのヌル配列を指定すると、メソッドが、信任状をメカニズムのデフォルト設定に戻します。

注: IBM JGSS は Kerberos V5 メカニズムのみをサポートするので、これがデフォルトのメカニズムです。

アプリケーションが一度に作成できるのは、3 つの信任状タイプ (*initiate*、*accept*、または *initiate and accept*) のうち 1 つだけです。

- コンテキスト・イニシエーターは、*initiate* 信任状を作成します。
- アクセプターは *accept* 信任状を作成します。
- イニシエーターとしても振る舞うアクセプターは、*initiate and accept* 信任状を作成します。

### 例: 信任状の取得

次の例は、イニシエーターにデフォルト信任状を取得します。

```
GSSCredentials fooCreds = manager.createCredentials(GSSCredential.INITIATE)
```

次の例は、デフォルトの有効期間を持つイニシエーター *foo* に、Kerberos V5 信任状を取得します。

```
GSSCredential fooCreds = manager.createCredential(fooName, GSSCredential.DEFAULT_LIFETIME,  
krb5Mech,GSSCredential.INITIATE);
```

次の例は、すべてデフォルトのアクセプター信任状を取得します。

```
GSSCredential serverCreds = manager.createCredential(null, GSSCredential.DEFAULT_LIFETIME,  
(Oid)null, GSSCredential.ACCEPT);
```

### GSSContext の作成:

IBM JGSS は、GSSManager がコンテキストの作成用に提供する 2 つのメソッドをサポートします。

このメソッドは以下のものです。

- コンテキスト・イニシエーターが使用するメソッド
- アクセプターが使用するメソッド

注: GSSManager は、前にエクスポートされたコンテキストの再作成を含む、コンテキストを作成するための 3 番目のメソッドを提供します。しかし、IBM JGSS Kerberos V5 メカニズムはエクスポートされたコンテキストの使用をサポートしていないので、IBM JGSS はこのメソッドをサポートしません。

アプリケーションは、コンテキストの受け入れ用のイニシエーター・コンテキストを使用したり、コンテキスト開始のためのアクセプター・コンテキストを使用したりすることはできません。サポートされるどちらのコンテキスト作成メソッドにも、入力としての信任状が必要です。信任状の値がヌルである場合、JGSS はデフォルト信任状を使用します。

### 例: GSSContext の使用

次の例では、プリンシパル (*foo*) がホスト (*securityCentral*) 上で対等機能 (*superSecureServer*) を使用してコンテキストを開始する際に使用するコンテキストを作成します。例では、対等機能を *superSecureServer@securityCentral* として指定します。作成されるコンテキストは、デフォルト期間に有効です。

```
GSSName serverName = manager.createName("superSecureServer@securityCentral",
                                         GSSName.NT_HOSTBASED_SERVICE, krb5Mech);
GSSContext fooContext = manager.createContext(serverName, krb5Mech, fooCreds,
                                             GSSCredential.DEFAULT_LIFETIME);
```

次の例は、どんな対等機能によっても開始されるコンテキストを受け入れるための `superSecureServer` のコンテキストを作成します。

```
GSSContext serverAcceptorContext = manager.createContext(serverCreds);
```

アプリケーションが、両方のタイプのコンテキストを作成し、同時に使用できることに注目してください。

### オプションのセキュリティー・サービスの要求:

アプリケーションは、オプションのセキュリティー・サービスをどれでも要求できます。IBM JGSS はいくつかのサービスをサポートしています。

サポートされるオプションのサービスは以下のとおりです。

- 代行
- 相互認証
- 再生検出
- 順不同検出
- 使用可能なメッセージごとの機密性
- 使用可能なメッセージごとの健全性

オプション・サービスを要求するには、アプリケーションは、コンテキスト上の適切な要求メソッドを使って明示的に要求する必要があります。これらのオプション・サービスを要求できるのはイニシエーターだけです。イニシエーターは、コンテキストの設定が始まる前に要求をする必要があります。

オプション・サービスについて詳しくは、Internet Engineering Task Force (IETF) RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1 の『Optional Service Support』を参照してください。

### 例: オプション・サービスの要求

次の例では、コンテキスト (`fooContext`) が相互認証および代行サービスを使用可能にする要求を行います。

```
fooContext.requestMutualAuth(true);
fooContext.requestCredDeleg(true);
```

### コンテキストの設定:

2 つの通信する対等機能は、メッセージごとのサービスを使用する際に用いるセキュリティー・コンテキストを設定することが必要です。

イニシエーターは、そのコンテキスト上で `initSecContext()` を呼び出し、それによってトークンがイニシエーター・アプリケーションに戻されます。イニシエーター・アプリケーションは、コンテキスト・トークンをアクセプター・アプリケーションに移送します。アクセプターはそのコンテキスト上で `acceptSecContext()` を呼び出し、イニシエーターから受け取ったコンテキスト・トークンを指定します。基礎となるメカニズムおよびイニシエーターが選択したオプション・サービスによっては、`acceptSecContext()`



は、アクセプター・アプリケーションがイニシエーター・アプリケーションに転送する必要のあるトークンを作成する場合があります。その後イニシエーター・アプリケーションは、受け取ったトークンを使用して、`initSecContext()` をもう一度呼び出します。

アプリケーションは、`GSSContext.initSecContext()` および `GSSContext.acceptSecContext()` に複数の呼び出しを行うことができます。また、コンテキスト設定中に、複数のトークンを対等機能と交換することもできます。したがって、コンテキスト設定の典型的なメソッドでは、アプリケーションがコンテキストを設定するまでは、ループを使って `GSSContext.initSecContext()` または `GSSContext.acceptSecContext()` を呼び出します。

### 例: コンテキストの設定

次の例では、コンテキスト設定のイニシエーター (foo) 側を例示します。

```
byte array[] inToken = null; // The input token is null for the first call
int inTokenLen = 0;

do {
    byte[] outToken = fooContext.initSecContext(inToken, 0, inTokenLen);

    if (outToken != null) {
        send(outToken); // transport token to acceptor
    }

    if( !fooContext.isEstablished()) {
        inToken = receive(); // receive token from acceptor
        inTokenLen = inToken.length;
    }
} while (!fooContext.isEstablished());
```

次の例では、コンテキスト設定のアクセプター側を例示します。

```
// The acceptor code for establishing context may be the following:
do {
    byte[] inToken = receive(); // receive token from initiator
    byte[] outToken =
        serverAcceptorContext.acceptSecContext(inToken, 0, inToken.length);

    if (outToken != null) {
        send(outToken); // transport token to initiator
    }
} while (!serverAcceptorContext.isEstablished());
```

### メッセージごとのサービスの使用:

セキュリティー・コンテキストの設定後、2 つの通信する対等機能が、設定されたコンテキストを介してセキュア・メッセージを交換します。

どちらの対等機能も、コンテキスト設定時にイニシエーターまたはアクセプターのどちらの役割を果たしたかに関係なく、セキュア・メッセージを発信できます。メッセージをセキュアにするため、IBM JGSS はメッセージを介して、暗号メッセージ保全コード (MIC) を計算します。オプションで、IBM JGSS は、プライバシーを確実にするために、Kerberos V5 メカニズムがメッセージを暗号化するようにできます。

### メッセージの送信

IBM JGSS は、メッセージをセキュアにするための 2 つのメソッドのセット `wrap()` および `getMIC()` を提供します。

#### `wrap()` の使用



wrap メソッドは、次のアクションを実行します。

- MIC の計算
- メッセージの暗号化 (オプション)
- トークンを戻す

呼び出し側アプリケーションは、MessageProp クラスを GSSContext と組み合わせて使用し、暗号化をメッセージに適用するかどうかを指定します。

戻されたトークンには、MIC とメッセージのテキストの両方が含まれます。メッセージのテキストは、暗号文 (暗号化されたメッセージ用) か、オリジナルのプレーン・テキスト (暗号化されていないメッセージ用) のどちらかです。

### getMIC() の使用

getMIC メソッドは、次のアクションを実行しますが、メッセージの暗号化はできません。

- MIC の計算
- トークンを戻す

戻されるトークンには、計算された MIC だけが入っており、オリジナル・メッセージは含まれません。したがって、MIC トークンを対等機能に移送することに加えて、MIC を検査できるように、対等機能に何らかの方法でオリジナル・メッセージが分かるようにする必要があります。

### 例: メッセージごとのサービスを使ってメッセージを送信する

次の例は、1 つの対等機能 (foo) が、別の対等機能 (superSecureServer) に送達するため、メッセージをラップする方法を示します。

```
byte[] message = "Ready to roll!".getBytes();
MessageProp mprop = new MessageProp(true); // foo wants the message encrypted
byte[] wrappedMessage =
    fooContext.wrap(message, 0, message.length, mprop);
send(wrappedMessage); // transfer the wrapped message to superSecureServer

// This is how superSecureServer may obtain a MIC for delivery to foo:
byte[] message = "You bet!".getBytes();
MessageProp mprop = null; // superSecureServer is content with
                          // the default quality of protection

byte[] mic =
    serverAcceptorContext.getMIC(message, 0, message.length, mprop);
send(mic);
// send the MIC to foo. foo also needs the original message to verify the MIC
```

### メッセージの受信

ラップされたメッセージの受信者は、unwrap() を使ってメッセージをデコードします。unwrap メソッドは、次のアクションを実行します。

- メッセージに埋め込まれた暗号 MIC を検査する
- 送信側が MIC の計算に使用したオリジナル・メッセージを戻す

送信側がメッセージを暗号化した場合、unwrap() は MIC の検査前にメッセージの暗号化を解除し、その後オリジナル・プレーン・テキストを戻します。MIC トークンの受信者は、verifyMIC() を使って指定されたメッセージを介して MIC を検査します。

対等アプリケーションは、独自のプロトコルを使って、JGSS コンテキストとメッセージ・トークンを相互に配信します。また、対等アプリケーションは、トークンが MIC または循環メッセージのどちらかを判別するため、プロトコルを定義する必要もあります。たとえば、そのようなプロトコルの一部は、Simple Authentication and Security Layer (SASL) アプリケーションが使用するプロトコルと同じほど単純 (かつ厳格) である場合があります。SASL プロトコルは、常にコンテキスト・アクセプターが、コンテキスト設定に続くメッセージごとの (ラップされる) トークンを送信するための最初の対等機能であることを指定します。

詳細は、Simple Authentication and Security Layer (SASL) を参照してください。

### 例: メッセージごとのサービスを使ってメッセージを受信する

次の例は、対等機能 (superSecureServer) が、別の対等機能 (foo) から受け取ったラップ・トークンをアンラップする方法を示します。

```
MessageProp mprop = new MessageProp(false);

byte[] plaintextFromFoo =
    serverAcceptorContext.unwrap(wrappedTokenFromFoo, 0,
                                wrappedTokenFromFoo.length, mprop);

// superSecureServer can now examine mprop to determine the message properties
// (such as whether the message was encrypted) applied by foo.

// foo verifies the MIC received from superSecureServer:

MessageProp mprop = new MessageProp(false);
fooContext.verifyMIC(micFromFoo, 0, micFromFoo.length, messageFromFoo, 0,
                    messageFromFoo.length, mprop);

// foo can now examine mprop to determine the message properties applied by
// superSecureServer. In particular, it can assert that the message was not
// encrypted since getMIC should never encrypt a message.
```

### コンテキストの削除:

対等機能は、コンテキストが必要なくなると、それを削除します。JGSS 操作では、各対等機能が、一方的にコンテキストを削除する時期を決定し、もう一方の対等機能に通知する必要はありません。

JGSS は、削除コンテキスト・トークンを定義しません。コンテキストを削除するには、対等機能は GSSContext オブジェクトの廃棄メソッドを呼び出し、コンテキストが使用するリソースがあればそれを解放します。廃棄された GSSContext オブジェクトは、アプリケーションによってヌルに設定されない限り、廃棄後もアクセス可能です。しかし、廃棄済み (ただしアクセス可能) コンテキストの使用を試みると、例外がスローされます。

### JGSS アプリケーションで JAAS を使用する:

IBM JGSS には、アプリケーションが JAAS を使って信任状を取得するためのオプションの JAAS ログイン機能が含まれます。JAAS ログイン機能がプリンシパル信任状および秘密鍵を JAAS ログイン・コンテキストの対象オブジェクトに保管した後、JGSS はその対象から信任状を検索できます。

JGSS のデフォルトの振る舞いは、その対象から信任状と秘密鍵を検索することです。Java プロパティー `javax.security.auth.useSubjectCredsOnly` を `false` に設定することによって、この機能を使用不可にすることができます。

注: 純粋な Java JGSS プロバイダーはログイン・インターフェースを使用できますが、ネイティブ iSeries JGSS プロバイダーは使用できません。

JAAS 機能について詳しくは、Kerberos 信任状および秘密鍵の取得を参照してください。

JAAS ログイン機能を使用するには、アプリケーションは次のように JAAS プログラミング・モデルに従う必要があります。

- JAAS ログイン・コンテキストの作成
- JAAS Subject.doAs 構成の範囲内での操作

次のコードの断片は、JAAS. Subject.doAs 構成の範囲内での操作の概念を例示します。

```
static class JGSSOperations implements PrivilegedExceptionAction {
    public JGSSOperations() {}
    public Object run () throws GSSException {
        // JGSS application code goes/runs here
    }
}

public static void main(String args[]) throws Exception {
    // Create a login context that will use the Kerberos
    // callback handler
    // com.ibm.security.auth.callback.Krb5CallbackHandler

    // There must be a JAAS configuration for "JGSSClient"
    LoginContext loginContext =
        new LoginContext("JGSSClient", new Krb5CallabackHandler());
    loginContext.login();

    // Run the entire JGSS application in JAAS privileged mode
    Subject.doAsPrivileged(loginContext.getSubject(),
        new JGSSOperations(), null);
}
```

## デバッグ

JGSS 問題を識別しようとしている場合、JGSS デバッグ機能を使用して、役立つカテゴリー化されたメッセージを生成します。

Java プロパティ `com.ibm.security.jgss.debug` に適切な値を設定することによって、1 つ以上のカテゴリーをオンにすることができます。複数のカテゴリーをアクティブにするには、コンマを使ってカテゴリー名を区切ります。

カテゴリーのデバッグには、次のものが関係します。

| カテゴリー | 説明                    |
|-------|-----------------------|
| help  | デバッグ・カテゴリーのリスト        |
| all   | すべてのカテゴリーのデバッグをオンにする  |
| off   | デバッグを完全にオフにする         |
| app   | アプリケーションのデバッグ (デフォルト) |
| ctx   | コンテキスト操作のデバッグ         |
| cred  | 信任状 (名前を含む) の操作       |
| marsh | トークンのマーシャル            |
| mic   | MIC 操作                |
| prov  | プロバイダー操作              |

| カテゴリー   | 説明           |
|---------|--------------|
| qop     | QOP 操作       |
| unmarsh | トークンのアンマーシャル |
| unwrap  | アンラップ操作      |
| wrap    | ラップ操作        |

## JGSS デバッグ・クラス

JGSS アプリケーションを方針に基づいてデバッグするには、IBM JGSS フレームワークでデバッグ・クラスを使用します。アプリケーションは、デバッグ・クラスを使用しデバッグ・カテゴリーをオン/オフにし、アクティブ・カテゴリーのデバッグ情報を表示することができます。

デフォルトのデバッグ・コンストラクターは、Java プロパティー `com.ibm.security.jgss.debug` を読み取り、アクティブにする (オンにする) カテゴリーを判別します。

### 例: アプリケーション・カテゴリーのデバッグ

次の例は、アプリケーション・カテゴリーでデバッグ情報を要求する方法を示します。

```
import com.ibm.security.jgss.debug;

Debug debug = new Debug(); // Gets categories from Java property

// Lots of work required to set up someBuffer. Test that the
// category is on before setting up for debugging.

if (debug.on(Debug.OPTS_CAT_APPLICATION)) {
    // Fill someBuffer with data.
    debug.out(Debug.OPTS_CAT_APPLICATION, someBuffer);
    // someBuffer may be a byte array or a String.
```

## サンプル: IBM Java Generic Security Service (JGSS)

IBM Java Generic Security Service (JGSS) サンプル・ファイルには、クライアントおよびサーバー・プログラム、構成ファイル、ポリシー・ファイル、および javadoc 参照情報が含まれます。サンプル・プログラムを使って JGSS セットアップをテストし、検証します。

サンプルの HTML バージョンを表示するか、またはサンプル・プログラムの javadoc 情報およびソース・コードをダウンロードすることができます。サンプルをダウンロードすることによって、javadoc 参照情報を表示し、コードを調査し、構成ファイルおよびポリシー・ファイルを編集し、サンプル・プログラムをコンパイルおよび実行することができます。

- サンプルの HTML バージョンを表示する
- サンプル javadoc 情報をダウンロードおよび表示する
- サンプル・プログラムをダウンロードおよび実行する

### サンプル・プログラムの説明

JGSS サンプルには、次の 4 つのプログラムが含まれます。

- 非 JAAS サーバー
- 非 JAAS クライアント
- JAAS 使用可能サーバー
- JAAS 使用可能クライアント

JAAS 使用可能バージョンは、対応する非 JAAS バージョンと完全に相互運用しています。したがって、JAAS 使用可能クライアントを非 JAAS サーバーに対して、または非 JAAS クライアントを JAAS 使用可能サーバーに対して実行することができます。

**注:** 注: サンプルの実行時、構成ファイルおよびポリシー・ファイルの名前、JGSS デバッグ・オプション、およびセキュリティー・マネージャーを含む、1 つ以上のオプションの Java プロパティーを指定することができます。また、JAAS 機能をオンにしたりオフにしたりすることもできます。

サンプルは、1 サーバー構成でも 2 サーバー構成でも実行できます。1 サーバー構成は、1 次サーバーと通信するクライアントから構成されます。2 サーバー構成は、1 次サーバーと 2 次サーバーから構成され、1 次サーバーは 2 次サーバーに対するイニシエーター、またはクライアントとして動作します。

2 サーバー構成を使用すると、クライアントは最初にコンテキストを開始し、ソース・メッセージを 1 次サーバーと交換します。次に、クライアントはその信任状を 1 次サーバーに委任します。次に、クライアントの代わりに、1 次サーバーがこれらの信任状を使用してコンテキストを開始し、ソース・メッセージを 2 次サーバーと交換します。また、1 次サーバーがそれ自体の代わりにクライアントとして動作する 2 サーバー構成を使用することもできます。この場合、1 次サーバーは独自の信任状を使用してコンテキストを開始し、2 次サーバーとメッセージを交換します。

1 次サーバーに対して同時に実行できるクライアントの数は決まっています。クライアントを直接 2 次サーバーに対して実行することは可能ですが、2 次サーバーが代行信任状を使用したり、独自の信任状を使用してイニシエーターとして実行することはできません。

#### **IBM JGSS サンプルの表示:**

IBM Java Generic Security Service (JGSS) サンプル・ファイルには、クライアントおよびサーバー・プログラム、構成ファイル、ポリシー・ファイル、および javadoc 参照情報が含まれます。次のリンクを使って、JGSS サンプルの HTML バージョンを表示してください。

#### **サンプル・プログラムの表示**

次のリンクを使って JGSS サンプル・プログラムの HTML バージョンを表示します。

- 非 JAAS クライアントのサンプル・プログラム
- 非 JAAS サーバーのサンプル・プログラム
- JAAS 対応クライアントのサンプル・プログラム
- JAAS 使用可能サーバー・プログラムのサンプル

#### **構成ファイルおよびポリシー・ファイルのサンプルの表示**

次のリンクを使って JGSS 構成ファイルおよびポリシー・ファイルの HTML バージョンを表示します。

- Kerberos 構成ファイル
- JAAS 構成ファイル
- JAAS ポリシー・ファイル
- Java ポリシー・ファイル

追加情報に関しては、以下のトピックを参照してください。

- サンプル・プログラムの説明
- サンプル・プログラムのダウンロードおよび実行

### サンプル: Kerberos 構成ファイル:

サンプル構成ファイルの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
# -----  
# Kerberos configuration file for running the JGSS sample applications.  
# Modify the entries to suit your environment.  
#-----  
  
[libdefaults]  
default_keytab_name = /QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab  
default_realm       = REALM.IBM.COM  
default_tkt_encypes = des-cbc-crc  
default_tgs_encypes = des-cbc-crc  
default_checksum    = rsa-md5  
kdc_timesync        = 0  
kdc_default_options = 0x40000010  
clockskew           = 300  
check_delegate      = 1  
ccache_type         = 3  
kdc_timeout         = 60000  
  
[realms]  
REALM.IBM.COM = {  
    kdc = kdc.ibm.com:88  
}  
  
[domain_realm]  
.ibm.com = REALM.IBM.COM
```

#### リンク集

IBM JGSS サンプルのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

### サンプル: JAAS ログイン構成ファイル:

サンプル構成ファイルの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
/**  
 * -----  
 * JAAS Login Configuration for the JGSS samples.  
 * -----  
 *  
 * Code example disclaimer  
 * IBM grants you a nonexclusive copyright license to use all programming code  
 * examples from which you can generate similar function tailored to your own  
 * specific needs.  
 * All sample code is provided by IBM for illustrative purposes only.  
 * These examples have not been thoroughly tested under all conditions.  
 * IBM, therefore, cannot guarantee or imply reliability, serviceability, or  
 * function of these programs.  
 * All programs contained herein are provided to you "AS IS" without any  
 * warranties of any kind.  
 * The implied warranties of non-infringement, merchantability and fitness  
 * for a particular purpose are expressly disclaimed.  
 *  
 */
```



```

*
* Supported options:
*   principal=<string>
*   credsType=initiator|acceptor|both (default=initiator)
*   forwardable=true|false (default=false)
*   proxiable=true|false (default=false)
*   useCcache=<URL_string>
*   useKeytab=<URL_string>
*   useDefaultCcache=true|false (default=false)
*   useDefaultKeytab=true|false (default=false)
*   noAddress=true|false (default=false)
*
* Default realm (which is obtained from the Kerberos config file) is
* used if the principal specified does not include a realm component.
*/

```

```

JAASClient {
  com.ibm.security.auth.module.Krb5LoginModule required
  useDefaultCcache=true;
};

```

```

JAASServer {
  com.ibm.security.auth.module.Krb5LoginModule required
  credsType=acceptor useDefaultKeytab=true
  principal=gss_service/myhost.ibm.com@REALM.IBM.COM;
};

```

## リンク集

### IBM JGSS サンプルのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

### サンプル: JAAS ポリシー・ファイル:

サンプル・ポリシー・ファイルの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

// -----
// JAAS policy file for running the JGSS sample applications.
// Modify these permissions to suit your environment.
// Not recommended for use for any purpose other than that stated above.
// In particular, do not use this policy file or its
// contents to protect resources in a production environment.
//
// Code example disclaimer
// IBM grants you a nonexclusive copyright license to use all programming code
// examples from which you can generate similar function tailored to your own
// specific needs.
// All sample code is provided by IBM for illustrative purposes only.
// These examples have not been thoroughly tested under all conditions.
// IBM, therefore, cannot guarantee or imply reliability, serviceability, or
// function of these programs.
// All programs contained herein are provided to you "AS IS" without any
// warranties of any kind.
// The implied warranties of non-infringement, merchantability and fitness
// for a particular purpose are expressly disclaimed.
//
// -----
//-----
// Permissions for client only
//-----

```

```

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "bob@REALM.IBM.COM"
{
    // foo needs to be able to initiate a context with the server
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "initiate";

    // So that foo can delegate his creds to the server
    permission javax.security.auth.kerberos.DelegationPermission
        "¥"gss_service/myhost.ibm.com@REALM.IBM.COM¥" ¥"krbtgt/REALM.IBM.COM@REALM.IBM.COM¥";
};

//-----
// Permissions for the server only
//-----

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service/myhost.ibm.com@REALM.IBM.COM"
{
    // Permission for the server to accept network connections on its host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // Permission for the server to accept JGSS contexts
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service/myhost.ibm.com@REALM.IBM.COM", "accept";

    // The server acts as a client when communicating with the secondary (backup) server
    // This permission allows the server to initiate a context with the secondary server
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "initiate";
};

//-----
// Permissions for the secondary server
//-----

grant CodeBase "file:ibmjgsssample.jar",
    Principal javax.security.auth.kerberos.KerberosPrincipal
        "gss_service2/myhost.ibm.com@REALM.IBM.COM"
{
    // Permission for the secondary server to accept network connections on its host
    permission java.net.SocketPermission "myhost.ibm.com", "accept";

    // Permission for the server to accept JGSS contexts
    permission javax.security.auth.kerberos.ServicePermission
        "gss_service2/myhost.ibm.com@REALM.IBM.COM", "accept";
};

```

## リンク集

### IBM JGSS サンプルのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

### サンプル: Java ポリシー・ファイル:

サンプル・ポリシー・ファイルの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

// -----
// Java policy file for running the JGSS sample applications on
// the iSeries server.

```

```

// Modify these permissions to suit your environment.
// Not recommended for use for any purpose other than that stated above.
// In particular, do not use this policy file or its
// contents to protect resources in a production environment.
//
// Code example disclaimer
// IBM grants you a nonexclusive copyright license to use all programming code
// examples from which you can generate similar function tailored to your own
// specific needs.
// All sample code is provided by IBM for illustrative purposes only.
// These examples have not been thoroughly tested under all conditions.
// IBM, therefore, cannot guarantee or imply reliability, serviceability, or
// function of these programs.
// All programs contained herein are provided to you "AS IS" without any
// warranties of any kind.
// The implied warranties of non-infringement, merchantability and fitness
// for a particular purpose are expressly disclaimed.
//
//-----
grant CodeBase "file:ibmjgsssample.jar" {
    // For Java 1.3
    permission javax.security.auth.AuthPermission "createLoginContext";

    // For Java 1.4
    permission javax.security.auth.AuthPermission "createLoginContext.JAASClient";
    permission javax.security.auth.AuthPermission "createLoginContext.JAASServer";

    permission javax.security.auth.AuthPermission "doAsPrivileged";

    // Permission to request a ticket from the KDC
    permission javax.security.auth.kerberos.ServicePermission
        "krbtgt/REALM.IBM.COM@REALM.IBM.COM", "initiate";

    // Permission to access sun.security.action classes
    permission java.lang.RuntimePermission "accessClassInPackage.sun.security.action";

    // A whole bunch of Java properties are accessed
    permission java.util.PropertyPermission "java.net.preferIPv4Stack", "read";
    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.util.PropertyPermission "DEBUG", "read";
    permission java.util.PropertyPermission "com.ibm.security.jgss.debug", "read";
    permission java.util.PropertyPermission "java.security.krb5.kdc", "read";
    permission java.util.PropertyPermission "java.security.krb5.realm", "read";
    permission java.util.PropertyPermission "java.security.krb5.conf", "read";
    permission java.util.PropertyPermission "javax.security.auth.useSubjectCredsOnly",
        "read,write";

    // Permission to communicate with the Kerberos KDC host
    permission java.net.SocketPermission "kdc.ibm.com", "connect,accept,resolve";

    // I run the samples from my localhost
    permission java.net.SocketPermission "myhost.ibm.com", "accept,connect,resolve";
    permission java.net.SocketPermission "localhost", "listen,accept,connect,resolve";

    // Access to some possible Kerberos config locations
    // Modify the file paths as applicable to your environment
    permission java.io.FilePermission "${user.home}/krb5.ini", "read";
    permission java.io.FilePermission "${java.home}/lib/security/krb5.conf", "read";

    // Access to the Kerberos key table so we can get our server key.
    permission java.io.FilePermission
        "/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab", "read";

```

```
// Access to the user's Kerberos credentials cache.  
permission java.io.FilePermission "${user.home}/krb5cc_${user.name}",  
"read";  
};
```

### サンプル: IBM JGSS サンプルの javadoc 情報のダウンロードおよび表示:

IBM JGSS サンプル・プログラムの文書をダウンロードして表示するには、次のステップに従ってください。

1. javadoc 情報を保管したい既存のディレクトリーを選択 (または新規のディレクトリーを作成) します。
2. そのディレクトリーに javadoc 情報をダウンロードします。
3. jgsssampledoc.zip から、ディレクトリーにファイルを解凍します。
4. ブラウザーを使って index.htm ファイルにアクセスします。

### コードの特記事項情報

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態を提供され、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

### サンプル: サンプル・プログラムのダウンロードおよび実行:

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

サンプルの変更または実行前に、サンプル・プログラムの説明をお読みください。

サンプル・プログラムを実行するには、以下のタスクを実行してください。

1. サンプル・ファイルを iSeries サーバーにダウンロードする
2. サンプル・ファイルの実行を準備する
3. サンプル・プログラムを実行する

サンプルの実行方法について詳しくは、380 ページの『例: 非 JAAS サンプルの実行』を参照してください。

### コードの特記事項情報

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態を提供され、いかなる保証も適用されません。商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任の保証の適用も一切ありません。

### **サンプル: IBM JGSS サンプルのダウンロード:**

このトピックでは、iSeries サーバーへのサンプル javadoc のダウンロードの手順を記載しています。

サンプルの変更または実行前に、サンプル・プログラムの説明をお読みください。

サンプル・ファイルをダウンロードし、iSeries サーバーに保管するには、次のステップに従ってください。

1. iSeries サーバーで、サンプル・プログラム、構成ファイル、およびポリシー・ファイルを保管したい既存のディレクトリーを選択 (または新規のディレクトリーを作成) します。
2. サンプル・プログラムをダウンロード (ibmjgsssample.zip) します。
3. サーバー上のディレクトリーに、ibmjgsssample.zip からファイルを解凍します。

ibmjgsssample.jar の内容の解凍により、次のアクションが実行されます。

- サンプルの .class ファイルを含む ibmjgsssample.jar を選択されたディレクトリーに入れる
- 構成ファイルおよびポリシー・ファイルを含むサブディレクトリー (名前付き config) を作成する
- サンプルの .java ソース・ファイルを含むサブディレクトリー (名前付き src) を作成する

### **関連情報**

関連タスクについて調べる場合、または例を見るには、以下のリンクを参照してください。

- 『サンプル: サンプル・プログラムの実行の準備』
- 380 ページの『サンプル: サンプル・プログラムの実行』
- 380 ページの『例: 非 JAAS サンプルの実行』

### **サンプル: サンプル・プログラムの実行の準備:**

ソース・コードのダウンロード後、サンプル・プログラムを実行する前にいくつかの準備タスクを実行する必要があります。

サンプルの変更または実行前に、サンプル・プログラムの説明をお読みください。

ソース・コードのダウンロード後、次のタスクを実行すると、サンプル・プログラムが実行できるようになります。

- 構成ファイルおよびポリシー・ファイルを、環境に合うように編集します。詳しくは、各構成ファイルおよびポリシー・ファイルにコメントを参照してください。
- java.security ファイルに、iSeries サーバーに合った正しい設定が含まれることを確認します。詳細については、Java マスター・セキュリティー・プロパティー・ファイルを参照してください。
- 変更された Kerberos 構成ファイル (krb5.conf) を、使用中の J2SDK のバージョンに適切な iSeries サーバー上のディレクトリーに入れてください。
  - J2SDK のバージョン 1.3 の場合: /QIBM/ProdData/Java400/jdk13/lib/security
  - J2SDK のバージョン 1.4 の場合: /QIBM/ProdData/Java400/jdk14/lib/security
  - J2SDK のバージョン 1.5 の場合: /QIBM/ProdData/Java400/jdk15/lib/security

### **関連情報**

関連タスクについて調べる場合、または例を見るには、以下のリンクを参照してください。

- 379 ページの『サンプル: IBM JGSS サンプルのダウンロード』
- 『サンプル: サンプル・プログラムの実行』
- 『例: 非 JAAS サンプルの実行』

#### **サンプル: サンプル・プログラムの実行:**

ソース・コードをダウンロードして変更した後、サンプルのうち 1 つを実行することができます。

サンプルの変更または実行前に、サンプル・プログラムの説明をお読みください。

サンプルを実行するには、まずサーバー・プログラムを開始する必要があります。サーバー・プログラムは、クライアント・プログラムの開始前に実行しており、接続を受け入れる準備ができていなければなりません。サーバーは、`listening on port <server_port>` と表示されると、接続を受け入れる準備ができていくことになります。 `<server_port >` は、クライアントの開始時に指定する必要のあるポート番号なので、必ず覚えておくか、書き留めてください。

次のコマンドを使ってサンプル・プログラムを開始します。

```
java [-Dproperty1=value1 ... -DpropertyN=valueN] com.ibm.security.jgss.test.<program> [options]
```

ここで、

- `[-DpropertyN=valueN]` は、構成ファイルおよびポリシー・ファイルの名前、JGSS デバッグ・オプション、およびセキュリティー・マネージャーを含む、1 つ以上のオプションの Java プロパティーです。詳しくは、以下の例、および JGSS アプリケーションの実行を参照してください。
- `<program>` は、実行するサンプル・プログラムを指定する、必須パラメーターです (Client、Server、JAASClient、または JAASServer のいずれか)。
- `[options]` は、実行するサンプル・プログラムのオプション・パラメーターです。サポートされるオプションのリストを表示するには、次のコマンドを使用してください。

```
java com.ibm.security.jgss.test.<program> -?
```

**注:** Java プロパティー `javax.security.auth.useSubjectCredsOnly` を `false` に設定することによって、JGSS 使用可能サンプルの JAAS 機能をオフにすることができます。もちろん、JAAS 使用可能サンプルのデフォルト値では JAAS はオンであり、プロパティー値は `true` です。非 JAAS クライアントおよびサーバー・プログラムは、明示的にプロパティー値を設定しない限り、プロパティーを `false` に設定しません。

#### **関連情報**

関連タスクについて調べる場合、または例を見るには、以下のリンクを参照してください。

- 379 ページの『サンプル: サンプル・プログラムの実行の準備』
- 379 ページの『サンプル: IBM JGSS サンプルのダウンロード』
- 『例: 非 JAAS サンプルの実行』

#### **例: 非 JAAS サンプルの実行:**

サンプルを実行するには、サンプル・ソース・コードをダウンロードして変更する必要があります。詳細は、サンプル・プログラムのダウンロードと実行を参照してください。



## 1 次サーバーの始動

次のコマンドを使用して、ポート 4444 で listen する非 JAAS サーバーを始動します。サーバーはプリンシパル (superSecureServer) として稼働し、2 次サーバー (backupServer) を使用します。サーバーは、アプリケーションおよび信任状デバッグ情報も表示します。

```
java -classpath ibmjgsssample.jar
      -Dcom.ibm.security.jgss.debug="app, cred"
      com.ibm.security.jgss.test.Server -p 4444
      -n superSecureServer -s backupServer
```

この例を正常に実行すると、次のメッセージが表示されます。

```
listening on port 4444
```

## 2 次サーバーの始動

次のコマンドを使用して、ポート 3333 で listen し、プリンシパル backupServer として稼働する、非 JAAS 2 次サーバーを始動します。

```
java -classpath ibmjgsssample.jar
      com.ibm.security.jgss.test.Server -p 3333
      -n backupServer
```

## クライアントの始動

次のコマンドを使用し (1 行に入力し)、JAAS 使用可能クライアント (myClient) を稼働します。クライアントは、ホストの 1 次サーバー (securityCentral) と通信します。クライアントは、デフォルトのセキュリティー・マネージャーを使用可能にした状態で稼働し、config ディレクトリーの JAAS 構成ファイルとポリシー・ファイル、および Java ポリシー・ファイルを使用します。config ディレクトリーの詳細は、IBM JGSS サンプルのダウンロードを参照してください。

```
java -classpath ibmjgsssample.jar
      -Djava.security.manager
      -Djava.security.auth.login.config=config/jaas.conf
      -Djava.security.policy=config/java.policy
      -Djava.security.auth.policy=config/jaas.policy
      com.ibm.security.jgss.test.JAASClient -n myClient
      -s superSecureServer -h securityCentral:4444
```

### リンク集

サンプル・プログラムのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

IBM JGSS サンプルのダウンロード

このトピックでは、iSeries サーバーへのサンプル javadoc のダウンロードの手順を記載しています。

## IBM JGSS javadoc 参照情報

IBM JGSS の javadoc 参照情報には、org.ietf.jgss api パッケージ中のクラスおよびメソッド、およびいくつかの Kerberos 信任状管理ツールの Java バージョンが含まれます。

JGSS には、公的にアクセス可能なパッケージ (たとえば com.ibm.security.jgss および com.ibm.security.jgss.spi) が含まれますが、使用するのは標準化された org.ietf.jgss パッケージの API のみにする必要があります。このパッケージだけを使用すると、アプリケーションが確実に GSS-API 仕様に準拠し、最適な相互運用性および移植性を確実にします。

- org.ietf.jgss
- 347 ページの『com.ibm.security.krb5.internal.tools Class Kinit』

- 349 ページの『com.ibm.security.krb5.internal.tools Class Ktab』
- 346 ページの『com.ibm.security.krb5.internal.tools Class Klist』

---

## IBM Developer Kit for Java を使用して Java プログラムのパフォーマンスを調整する

ご使用の iSeries サーバー用に Java アプリケーションを構築する際は、Java アプリケーションのパフォーマンスに関していくつかの点を考慮する必要があります。

以下は、パフォーマンスを向上させる方法の詳細へのリンクとヒントです。

- 「Java プログラムの作成 (CRTJVAPGM)」コマンド、Just-In-Time コンパイラー、あるいはユーザー・クラス・ローダーのキャッシュを使用して、Java コードのパフォーマンスを改善する。
- 静的コンパイルのパフォーマンスを最高にするために最適化レベルを変更する。
- ガーベージ・コレクションのパフォーマンスを最適化するために注意深く値を設定する。
- ネイティブ・メソッドは、比較的执行時間の長いシステム機能や、Java では直接に使用できないシステム機能を開始する場合にのみ使用してください。
- メソッドのインライン化を行って、メソッド呼び出しのパフォーマンスを大幅に向上させるため、コンパイル時に `javac -o` オプションを使用する。
- アプリケーションのフローが正常でない場合に Java 例外処理を使用する。

Java プログラム内のパフォーマンス上の問題を突き止めるには、Performance Explorer (PEX) と共に以下のツールを使用します。

- 383 ページの『Java イベント・トレースのパフォーマンス測定ツール』を収集するには、iSeries Java 仮想マシンを使用する。
- 各 Java メソッドに要した時間を判別するには、Java 呼び出しトレースを使用する。
- 各 Java メソッドと、Java プログラムによって使用されたすべてのシステム機能に要した、相対的な CPU 時間を検出するには、Java プロファイルを使用する。
- Java パフォーマンス・データ・コレクターを使用して、iSeries サーバーで実行されているプログラムに関するプロファイル情報を収集する。

ジョブ・セッションによって、PEX が起動および終了します。通常、収集されるデータはシステム全体に渡っており、Java プログラムを含めたシステム上のすべてのジョブに関係します。場合によっては、Java アプリケーションの内部からパフォーマンス収集を開始したり停止したりしなければならないことがあります。この場合、収集時間が短くなり、呼び出し/戻りトレースによって通常生成される大量のデータを減少させることができます。PEX は、Java スレッドの内部からは実行できません。収集を開始および停止するには、キューまたは共有メモリーを介して独立したジョブと通信するネイティブ・メソッドを作成する必要があります。この場合、2 番目のジョブが収集を適宜開始および停止します。

アプリケーション・レベルのパフォーマンス・データに加えて、既存の iSeries システム・レベルのパフォーマンス測定ツールを使用できます。これらのツールによって、Java のスレッドごとに統計が報告されます。

### 関連情報

eServer iSeries Performance Tools for iSeries (SD88-5051)



この資料には、PEX レポートの例が記載されています。

## Java イベント・トレースのパフォーマンス測定ツール

iSeries Java 仮想マシンは、特定の Java イベントのトレースを可能にします。

これらのイベントは、Java コードの計測を使用せずに収集できます。これらのイベントには、ガーベッジ・コレクション、スレッドの作成、クラスのロード、およびロックなどの活動が含まれます。「Java プログラムの実行 (RUNJAVA)」コマンドは、これらのイベントを指定しません。代わりに、ユーザーが Performance Explorer (PEX) 定義を作成し、「パフォーマンス・エクスプローラーの開始 (STRPEX)」コマンドを使用してイベントを収集します。それぞれのイベントには、タイム・スタンプおよび中央演算処理装置 (CPU) サイクルなどの便利なパフォーマンス情報が含まれています。Java イベントと、ディスクの入出力などの他のシステム活動の両方を、同じトレース定義を使用してトレースできます。

Java イベントの詳細は、「Performance Tools for iSeries (SC41-5340)」を参照してください。

## Java のパフォーマンスの考慮事項

以下の考慮事項は、Java アプリケーションのパフォーマンス向上に役立つ内容となっています。

### 最適化 Java プログラムの作成

Java コードの始動パフォーマンスを大幅に改善するため、Java クラス・ファイル、JAR ファイル、または ZIP ファイルを実行する前に、「Java プログラムの作成 (CRTJVAPGM)」制御言語 (CL) コマンドを使用してください。CRTJVAPGM は、バイトコードを使用して Java プログラム・オブジェクトを作成します。それには iSeries サーバーのための最適化ネイティブ命令が含まれていて、Java プログラム・オブジェクトをクラス・ファイル、JAR ファイル、または ZIP ファイルに関連付けます。

Java プログラムは保管されてクラス・ファイルまたは JAR ファイルに関連付けられたままとするため、それ以後の実行速度はより速くなります。バイトコードの解釈による実行は、アプリケーションの開発期間中は受容可能なパフォーマンスを提供することもあります。プロダクション環境の前に CRTJVAPGM コマンドを使用することが必要になるに違いありません。

Java クラス・ファイル、JAR ファイル、または ZIP ファイルを実行するまで CRTJVAPGM コマンドを使用しないとき、i5/OS は代わりに Just-In-Time コンパイラーを (Mixed-Mode Interpreter と共に) 使用します。

### 最適化レベルの選択

Java プログラム・オブジェクトを作成する際は、以下のガイドラインを使用して、使用する実行モードに最も適した最適化レベルを選択してください。

- 直接処理を使用する場合は、レベル 30 または 40 の最適化レベルで最適化 Java プログラム・オブジェクトを作成してください。
- JIT コンパイラーだけを使用して実行する場合は、\*Interpret 最適化パラメーターを使用して最適化 Java プログラムを作成してください。\*Interpret パラメーターを使用して作成された Java プログラムは、最適化レベル 40 を使用して作成された Java プログラムよりも小さくなります。
- 直接処理と JIT コンパイラーの混合である、デフォルトの実行モードを使用する場合は、以下の設定を使用して Java プログラム・オブジェクトを作成してください。
  - 直接処理で実行したいクラスには、最適化レベル 30 または 40 を使用する
  - JIT コンパイラーを使用して実行したいクラスには、\*Interpret 最適化パラメーターを使用する

詳しくは、以下のページを参照してください。

Java プログラム作成 (CRTJVAPGM) 制御言語コマンド

Java プログラムの実行時に使用するモードの選択

## Just-In-Time コンパイラーの使用

Mixed-Mode Interpreter (MMI) と共に Just-In-Time (JIT) コンパイラーを使用する場合、始動パフォーマンスはコンパイル・コードとほとんど同じです。MMI は `os400.jit.mmi.threshold` Java システム・プロパティーで指定されたしきい値に達するまで、Java コードを解釈します。しきい値に達した後に、i5/OS は JIT コンパイラーを使用してメソッドをコンパイルするために必要な時間とリソースを、最も頻繁に使用されるメソッドに費やします。JIT コンパイラーを使用すると、コードは高度に最適化され、プリコンパイル・コードと比べて実行時パフォーマンスが向上します。JIT コンパイラーを使用して始動パフォーマンスを高めたい場合は、CRTJVAPGM を使用して最適化 Java オブジェクトを作成してください。

プログラムの実行速度が遅い場合、「Java プログラムの表示 (DSPJVAPGM)」制御言語 (CL) コマンドを入力して、Java プログラム・オブジェクトの属性を表示してください。Java プログラム・オブジェクトが処理に最適な実行モードを使用していることを確認してください。実行モードを変更する場合は、その Java プログラム・オブジェクトを削除し、別の最適化パラメーターを使用して新しい Java プログラム・オブジェクトを作成することができます。

詳しくは、以下を参照してください。

Java プログラムの表示 (DSPJVAPGM) 制御言語コマンド

## ユーザー・クラス・ローダーのキャッシュの使用

ユーザー・クラス・ローダー用の i5/OS Java 仮想マシン (JVM) キャッシュを使用すると、ユーザー・クラス・ローダーからロードするクラスの始動パフォーマンスが向上します。キャッシュが最適化 Java プログラム・オブジェクトを保管するので、JVM はそれらを再利用できます。保管されている Java プログラムの再利用は、キャッシュされた Java プログラム・オブジェクトの再作成とバイトコードの検査の両方が避けられるので、パフォーマンスが向上させます。

ユーザー・クラス・ローダーのキャッシュの制御には、以下のプロパティーを使用します。

### `os400.define.class.cache.file`

このプロパティーの値は、有効な JAR ファイルの名前 (と絶対パス) を指定します。指定する JAR ファイルには、少なくとも有効な JAR ディレクトリー (`jar QSH` コマンドによって作成される) と、`jar` コマンドが機能するのに必要な単一のメンバーが含まれていなければなりません。Java CLASSPATH に指定した JAR ファイルは含めないでください。このプロパティーのデフォルト値は、`/QIBM/ProdData/Java400/QDefineClassCache.jar` です。キャッシュを使用不可にする場合は、このプロパティーを値を入れずに指定してください。

### `os400.define.class.cache.hours`

このプロパティーの値は、Java プログラム・オブジェクトをキャッシュに保管する長さ (時間単位) を指定します。JVM がキャッシュに入れられた Java プログラム・オブジェクトを指定された期間使用しないと、i5/OS はその Java プログラムをキャッシュから除去します。このプロパティーのデフォルト値は、768 時間 (33 日間) です。最大値は 9999 (約 59 週間) です。値 0、あるいは i5/OS が有効な 10 進数として認識しない値が指定された場合、i5/OS はデフォルト値を使用します。

### `os400.define.class.cache.maxpgms`

このプロパティーの値は、キャッシュに格納できる Java プログラム・オブジェクトの最大値を指定します。キャッシュがこの限界を超えると、i5/OS は最も古い Java プログラム・オブジェクト

をキャッシュから除去します。i5/OS は、どのキャッシュ・プログラムが最も古いかを、JVM が最後に Java プログラム・オブジェクトを参照した時刻を比較して判別します。デフォルト値は 5000 で、最大値は 40000 です。値 0、あるいは i5/OS が有効な 10 進数として認識しない値が指定された場合、i5/OS はデフォルト値を使用します。

キャッシュに入れられた Java プログラム・オブジェクトの数を判別するには、os400.define.class.cache.file プロパティに指定する JAR ファイルで DSPJVAPGM を使用してください。

- DSPJVAPGM ディスプレイの **Java プログラム** フィールドは、キャッシュに入れられた Java プログラム・オブジェクトの数を示します。
- **Java プログラムのサイズ** フィールドは、キャッシュに入れられた Java プログラム・オブジェクトによって使用されているストレージ量を示します。
- DSPJVAPGM ディスプレイの他のフィールドは、キャッシュ用に使用している JAR ファイルに対してこのコマンドを使用している場合は、意味を持ちません。

### キャッシュのパフォーマンス

いくつかの Java プログラムを実行すると、多数の Java プログラム・オブジェクトがキャッシュされる可能性があります。アプリケーションの実行が終了する前に、DSPJVAPGM を使用して、キャッシュされた Java プログラムの数が最大値に近づいているかどうかを判断してください。キャッシュがいっぱいになると、i5/OS はアプリケーションに必要な一部のプログラムをキャッシュから除去することがあるので、アプリケーション・パフォーマンスが低下する可能性があります。

キャッシュがフルになったために生じるパフォーマンスの低下は防ぐことが可能です。たとえば、頻繁に実行し、さまざまなプログラムをキャッシュにロードするアプリケーション用には別個のキャッシュを使用するようアプリケーションをセットアップできます。別個のキャッシュを使用すれば、キャッシュがいっぱいになることが防がれ、i5/OS がキャッシュから Java プログラムを除去することが防止されます。また別の方法として、os400.define.class.cache.maxpgms プロパティに指定する数を増やすこともできます。

キャッシュの中のクラスの最適化を変更するには、JAR ファイルで「Java プログラムの変更 (CHGJVAPGM)」制御言語 (CL) コマンドを使用します。CHGJVAPGM は、現在キャッシュに保管されているプログラムにのみ影響します。最適化レベルを変更した後は、os400.defineClass.optLevel プロパティが、キャッシュに追加されたクラスの最適化方法を指定します。

たとえば、最大で 10000 の Java プログラム・オブジェクト (各 Java プログラムは最大で 1 年間存続する) を含む配送済みのキャッシュ JAR を使用する場合、キャッシュ・プロパティには以下の値を設定します。

```
os400.define.class.cache.file    /QIBM/ProdData/Java400/QDefineClassCache.jar
os400.define.class.cache.hours   8760
os400.define.class.cache.maxpgms 10000
```

### Java プログラムの実行時に使用するモードの選択

Java プログラムを実行するとき、使用するモードを選択できます。どのモードでも、コードが検査され、事前に検証された形式のプログラムを保持する Java プログラム・オブジェクトが作成されます。

以下のいずれかのモードを使用できます。

- インタープリット
- 直接処理
- Just-In-Time (JIT) コンパイル
- Just-In-Time (JIT) コンパイルおよび直接処理



| 選択モード                    | 詳細   |
|--------------------------|--|
| インタープリット                 | <p>各バイトコードは、実行時にインタープリットされます。</p> <p>Java プログラムをインタープリット・モードで実行するための情報は、「Java プログラムの実行 (RUNJVA)」コマンドを参照してください。</p>   |
| 直接処理                     | <p>メソッドを最初に呼び出すときにそのメソッドのための機械命令が生成されて、そのプログラムを次に使用するときのために保管されます。さらに、1 つのコピーがシステム全体のために使用されます。</p> <p>Java プログラムを直接処理で実行するための情報は、「Java プログラムの実行 (RUNJVA)」コマンドを参照してください。</p>   |
| Just-In-Time (JIT) コンパイル | <p>i5/OS は <code>os400.jit.mmi.threshold</code> Java システム・プロパティーで指定されたしきい値に達するまで、Java メソッドを解釈します。しきい値に達した後、i5/OS は JIT コンパイラーを使用してメソッドをコンパイルし、ネイティブ・マシン命令を作成します。</p> <p>Just-In-Time コンパイラーを使用するには、コンパイラー値を <code>jitc</code> に設定しなければなりません。環境変数を追加するか、<code>java.compiler</code> システム・プロパティーを設定することにより、その値を設定できます。以下のリストから 1 つの方式を選択して、コンパイラー値を設定します。</p> <ul style="list-style-type: none"> <li>「環境変数の追加 (ADDENVVAR)」コマンドを iSeries サーバー上のコマンド行プロンプトに入力して、環境変数を追加します。その後、「Java プログラムの実行 (RUNJVA)」コマンドまたは <code>JAVA</code> コマンドを使用して、Java プログラムを実行します。たとえば、次のように入力します。</li> </ul> <pre>ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc) JAVA CLASS(Test)</pre> <ul style="list-style-type: none"> <li><code>java.compiler</code> システム・プロパティーを iSeries コマンド行で設定します。たとえば、<code>JAVA CLASS(Test) PROP((java.compiler jitc))</code> と入力します。</li> <li><code>java.compiler</code> システム・プロパティーを Qshell インタープリターのコマンド行で設定します。たとえば、<code>java -Djava.compiler=jitc Test</code> と入力します。</li> </ul> <p>この値を設定した後は、JIT コンパイラーがすべての Java コードを実行前に最適化します。</p> |



| 選択モード                                  | 詳細   |
|--|--|
| <b>Just-In-Time (JIT) コンパイルおよび直接処理</b> | <p>Just-In-Time (JIT) コンパイラーを使用する最も一般的な方法は、jit_de オプションを指定することです。このオプションを指定して実行すると、直接処理のために既に最適化されているプログラムは直接処理モードで実行されます。直接の最適化のために最適化されていないプログラムは、JIT モードで実行されます。</p> <p>JIT および直接処理を同時に使用する場合、コンパイラー値を jitc_de に設定する必要があります。環境変数を追加するか、java.compiler システム・プロパティを設定することにより、その値を設定できます。以下のリストから1つの方式を選択して、コンパイラー値を設定します。</p> <ul style="list-style-type: none"> <li>「環境変数の追加 (ADDENVVAR)」コマンドを iSeries コマンド行に入力して、環境変数を追加します。その後、「Java プログラムの実行 (RUNJAVA)」コマンドまたは JAVA コマンドを使用して、Java プログラムを実行します。たとえば、次のように入力します。<br/> ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc_de)<br/> JAVA CLASS(Test)</li> <li>java.compiler システム・プロパティを iSeries コマンド行で設定します。たとえば、JAVA CLASS(Test) PROP((java.compiler jitc_de)) と入力します。</li> <li>java.compiler システム・プロパティを Qshell インタープリターのコマンド行で設定します。たとえば、<br/> java -Djava.compiler=jitc_de Test と入力します。</li> </ul> <p>この値を設定した後は、直接処理として作成されたクラス・ファイルのための Java プログラムが使用されます。Java プログラムが直接処理として作成されていない場合、クラス・ファイルは実行前に JIT によって最適化されます。詳細については、Just-In-Time コンパイラーと直接処理の比較を参照してください。</p> |

Java プログラムを実行するには、3つの方法があります (CL、QSH、および JNI)。それぞれに、モードを指定する固有の方法があります。この表には、指定方法が示されています。

| モード            | CL コマンド                                | QShell コマンド                        | JNI 呼び出し API   |
|----------------|--|------------------------------------|--|
| インタープリット       | INTERPRET(*YES)                        | -Djava.compiler=NONE<br>-interpret | os400.run.mode=interpret   |
| DE             | INTERPRET(*NO)                         | -Djava.compiler=NONE               | <ul style="list-style-type: none"> <li>os400.run.mode=program_created=pc</li> <li>os400.create.type= direct</li> </ul> |
| JIT            | INTERPRET(*JIT)                        | -Djava.compiler=jitc               | os400.run.mode=jitc  |
| JIT_DE (デフォルト) | INTERPRET(*OPTIMIZE)<br>OPTIMIZE(*JIT) | -Djava.compiler=jitc_de            | os400.run.mode=jitc_de   |

## Java インタープリター

Java インタープリターは、特定のハードウェア・プラットフォームで Java クラス・ファイルを解釈する、Java 仮想マシンの一部です。Java インタープリターは各バイトコードをデコードし、そのバイトコードに関する一連のマシン・インストラクションを実行します。

Java 仮想マシン

### 静的コンパイル

Java 変換プログラムは IBM i5/OS コンポーネントであり、iSeries Java 仮想マシンを使ってクラス・ファイルを実行するためのプリプロセスを行います。Java 変換プログラムは、クラス・ファイルに関連付けられた永続性のある最適化されたプログラム・オブジェクトを作成します。

デフォルトの設定では、プログラム・オブジェクトにはコンパイル済みの 64 ビット RISC 機械命令バージョンのクラスが含まれます。最適化されたプログラム・オブジェクトは、実行時に Java インタープリターによって解釈されるのではなく、クラス・ファイルのロード時に直接実行されます。

Java プログラムは、デフォルトでは JIT を使用して最適化されます。Java 変換プログラムを使用するには、CRTJVAPGM を実行するか、RUNJVA または JAVA コマンドで変換プログラムを使用することを指定します。

「Java プログラムの作成 (CRTJVAPGM)」コマンドを使って Java 変換プログラムを明示的に開始する方法もあります。CRTJVAPGM コマンドは、コマンドの実行時にクラス・ファイルや JAR ファイルを最適化するので、プログラムの実行時には何も行う必要がありません。したがって、プログラムが初めて実行される際の速度が上がります。デフォルトの最適化を使用せずに CRTJVAPGM コマンドを使用すると、確実に最善の最適化を実行でき、クラス・ファイルや JAR ファイルに関連した Java プログラムのスペース使用状況も改善されます。

クラス・ファイル、JAR ファイル、または ZIP ファイルに CRTJVAPGM コマンドを使用すると、ファイル内のすべてのクラスが最適化され、結果として作成される Java プログラム・オブジェクトは永続的なものとなります。これにより、実行時のパフォーマンスが向上します。さらに、CRTJVAPGM コマンドが「Java プログラムの変更 (CHGJVAPGM)」コマンドを使用して、最適化レベルを変更したり、デフォルトの 10 以外の最適化レベルを選択することもできます。最適化レベル 40 の場合、JAR ファイル中のクラス間でクラス間バインドが実行され、場合によってはそれらのクラスがインラインになります。クラス間バインドを実行すると、呼び出しの速度が上がります。インラインにすると、メソッド呼び出し全体のオーバーヘッドがなくなります。場合によっては、JAR ファイルや ZIP ファイル中のクラス間のメソッドをインラインにすることもできます。CRTJVAPGM コマンドで OPTIMIZE(\*INTERPRET) を指定すると、コマンドで指定したすべてのクラスが検査され、解釈モードでの実行用に準備されます。

「Java プログラムの実行 (RUNJVA)」コマンドでも OPTIMIZE(\*INTERPRET) を指定できます。このパラメーターは、Java 仮想マシンのもとで実行されるクラスはすべて、関連するプログラム・オブジェクトの最適化レベルに関係なく解釈されることを指示します。このパラメーターは、最適化レベル 40 で変換されたクラスをデバッグするときに役立ちます。解釈を強制実行するには、INTERPRET(\*YES) を使用します。

クラス・ローダーが作成した Java プログラムの再利用については、Java のパフォーマンスの考慮事項中の『ユーザー・クラス・ローダーのキャッシュの使用』を参照してください。

#### Java 静的コンパイルのパフォーマンスについての考慮事項:

最適化レベルの設定値によって、変換の速度を判別できます。

最適化レベル 10 の場合、変換は最速ですが、一般的に変換後のプログラムは、10 より大きな最適化レベルを設定した場合より低速になります。最適化レベルを 40 にすると、変換時間はかかりますが、おそらく実行速度は高速になります。

少数の Java プログラムは、レベル 40 で最適化できません。したがって、レベル 40 で実行せず、代わりにレベル 30 で実行する可能性のあるプログラムもわずかながらあります。最適化レベル 40 で稼働しないプログラムを実行するには、ライセンス内部コード最適化 LICOPT パラメーター・ストリングを使用します。しかし、ご使用のプログラムにとってはレベル 30 のパフォーマンスで十分である可能性もあります。

別の Java 仮想マシンで稼働していると思われる Java コードに問題がある場合は、最適化レベル 40 ではなくレベル 30 を使用してみてください。この設定で稼働し、パフォーマンスが許容できる範囲内である場合は、他に何も行う必要はありません。パフォーマンスを改善する必要がある場合は、さまざまな形式の最適化を使用可能にしたり使用不可にしたりする方法に関する情報について、LICOPT パラメーター・ストリングを参照してください。たとえば、まず OPTIMIZE(40) LICOPT(NoPreresolveExtRef) を使用してプログラムを作成して試みるのができます。利用不能なクラスに対する非活動呼び出しがアプリケーション中にある場合は、この LICOPT 値を使用すると、問題なくプログラムを実行できます。

Java プログラムが作成されたときの最適化レベルを判別するには、「Java プログラムの表示 (DSPJVAPGM)」コマンドを使用します。Java プログラムの最適化レベルを変更するには、「Java プログラムの作成 (CRTJVAPGM)」コマンドを使用します。

## Just-In-Time コンパイラー

Just-In-Time (JIT) コンパイラーは、プラットフォーム固有のコンパイラーであり、必要に応じて各メソッド用の機械命令を生成します。

JIT コンパイラーの使用法、および JIT コンパイラーと直接処理の違いについて詳しくは、以下のページを参照してください。

Java の実行時のパフォーマンスについての考慮事項

JIT コンパイラーと直接処理の比較

注: i5/OS のデフォルトの設定では、Java メソッドは、Mixed-Mode Interpreter (MMI) を使用して解釈 (コンパイルではない) されます。MMI は、各 Java メソッドを解釈するときそのプロファイルを作成します。os400.jit.mmi.threshold プロパティーで指定されたしきい値に達すると、MMI は、i5/OS が JIT コンパイラーを使用してメソッドをコンパイルするよう指定します。

詳しくは、該当する Java システム・プロパティーのリストの中の java.compiler プロパティーの項目と、os400.jit.mmi.threshold プロパティーの項目を参照してください。

Java 2 SDK (J2SDK) Standard Edition の Java システム・プロパティー

### JIT コンパイラーと直接処理の比較:

次の表では、Java プログラムの実行に Just-In-Time コンパイラーと直接処理モードのどちらを使用するかを決定する際に、状況に最も適した選択を行うのに役立つ追加情報を提供します。

Just-In-Time コンパイラーまたは直接処理モード

| Just-In-Time コンパイラー  | 直接処理  |
|--|---|
| 必要なときに、メソッドの自動コンパイルを提供します。JIT コンパイラーは、直接処理よりもずっと速くメソッドをコンパイルできます。  | ユーザーは「Java プログラム作成 (CRTJVAPGM)」制御言語 (CL) コマンドを使用して、クラスまたは JAR ファイル全体をコンパイルできます。ユーザーがファイルをコンパイルしない場合、ファイルは実行時に直接処理によって自動的にコンパイルされます。                 |
| プログラムの開発時に CRTJVAPGM CL コマンドを使用しなくてもよくなります。さらに、JIT コンパイラーは、実行時にコードを生成または検出する、非常に動的なアプリケーションと共に使用することもできます。 | 展開可能な状態にあるサーバー・アプリケーションの大部分は、おそらく、同時に複数のユーザーによって使用されることになるため、最適化レベル 40 の直接処理を使用します。メモリー内の同一のコード・スペースが複数のユーザー・ジョブによって共用されることにより、メモリー・フットプリントが削減されます。 |
| 複雑な最適化や Java 固有の最適化を実行時にすばやく実行します。   | 直接処理は、最適化を実行時に実行するのではないため、複雑な最適化を行えます。しかし、Java プログラム・オブジェクトは独立しているはずなので、直接処理は常に (メソッドのインライン化などの) Java 固有の最適化を実行できるとは限りません。                          |
| 直接処理と比較して、コード・パフォーマンスが高くなります。大抵の場合、JIT で生成されたコードのパフォーマンスは、直接処理の最適化レベル 40 よりも良好です。                          | Java プログラムが所有者権限を採用するための、唯一の方法です。   |

## Java ガーベッジ・コレクション

ガーベッジ・コレクションは、プログラムが参照することのなくなったオブジェクトによって使用される記憶域を空にするプロセスです。ガーベッジ・コレクションを使用すると、プログラマーはオブジェクトを明示的に「空にする」かまたは「削除する」ために、エラーになりやすいコードを作成する必要がなくなります。このコードは、頻繁に「メモリー・リーク」プログラム・エラーという結果になります。ガーベッジ・コレクターは、ユーザー・プログラムが達することのできないオブジェクトまたはオブジェクトのグループを自動的に検出します。これを行うのは、どのプログラム構造でもそのオブジェクトを参照していないからです。オブジェクトを収集後、そのスペースを他のことに使用するように割り振ることができます。

Java ランタイム環境には、使用されなくなったメモリーを解放するガーベッジ・コレクターが含まれています。ガーベッジ・コレクターは、必要に応じて自動的に実行されます。

また、`java.lang.Runtime.gc()` メソッドを使用して、Java プログラムの制御下でガーベッジ・コレクターを明示的に起動することもできます。

## IBM Developer Kit for Java の拡張ガーベッジ・コレクション

IBM Developer Kit for Java は、拡張ガーベッジ・コレクター・アルゴリズムを実装します。このアルゴリズムによって、Java プログラムの操作を一時停止することなく、使用されていないオブジェクトを発見し、収集することができます。並行コレクターは協力して、単一のスレッドではなく、実行している複数のスレッドのオブジェクトへの参照を発見します。

多くのガーベッジ・コレクターは、「全機能停止」です。これは、コレクション・サイクルが生じる時点で、ガーベッジ・コレクションを行うスレッド以外のスレッドはすべて、ガーベッジ収集プログラムがその作業を行っている間は停止することを意味します。収集が始まると Java プログラムは一時停止し、コレクターがその作業をしている間、Java と関連したプラットフォームの多重処理装置機能が無駄になります。iSeries のアルゴリズムでは、すべてのプログラム・スレッドが同時に停止することはありません。このた



め、ガーベッジ・コレクターがそのタスクを完了しているときに、これらのスレッドが操作を継続することができます。これにより、一時停止が防止され、ガーベッジ・コレクション中でもすべての処理装置を使用することが可能になります。

ガーベッジ・コレクションは、Java 仮想マシンの起動時に指定したパラメーターに基づいて、自動的に実行されます。 `java.lang.Runtime.gc()` メソッドを使用することにより、Java プログラムの制御下で明示的にガーベッジ・コレクションを開始することもできます。

基本的な概念の定義については、Java ガーベッジ・コレクションを参照してください。

## Java ガーベッジ・コレクションのパフォーマンスについての考慮事項

iSeries Java 仮想マシン上のガーベッジ・コレクションは、継続される非同期モードで操作されます。

「Java プログラムの実行 (RUNJAVA)」コマンドのガーベッジ初期サイズ (GCHINL) パラメーターは、アプリケーション・パフォーマンスに影響することがあります。

GCHINL パラメーターは、ガーベッジ・コレクション間で許可される新しいオブジェクト・スペースの量を指定します。小さな値の場合、ガーベッジ・コレクションのオーバーヘッドが過大になることがあります。値が大きいと、ガーベッジ・コレクションを制限することになり、メモリー不足のエラーが生じることがあります。しかし、ほとんどのアプリケーションでは、デフォルト値が適したものになります。

ガーベッジ・コレクションでは、オブジェクトへの有効な参照があるかどうかに基づいて、そのオブジェクトが必要とされなくなったかどうか分かります。

## Java ネイティブ・メソッド呼び出しのパフォーマンスに関する考慮事項

iSeries サーバーでのネイティブ・メソッド呼び出しは、他のプラットフォームでのネイティブ・メソッド呼び出しよりもパフォーマンスが低くなる場合があります。

iSeries サーバー上の Java は、Java 仮想マシンをマシン・インターフェース (MI) の下に移動することによって最適化されています。ネイティブ・メソッド呼び出しには、MI コードの範囲を超えた呼び出しが必要であり、Java ネイティブ・インターフェース (JNI) で、Java 仮想マシンに戻るための余分な呼び出しが必要になる場合もあります。ネイティブ・メソッドは、Java で簡単に作成できる小さなルーチンの実行に使用すべきではありません。ネイティブ・メソッドは、比較的执行時間の長いシステム機能や、Java では直接に使用できないシステム機能を開始する場合にのみ使用してください。

## Java メソッドのインライン化のパフォーマンスに関する考慮事項

メソッドのインライン化により、メソッド呼び出しのパフォーマンスが著しく向上することがあります。すべての最終メソッドは、インライン化の候補になります。

インライン機能は、コンパイル時に `javac -o` オプションを使用することによって、iSeries サーバーで使用できます。 `javac -o` オプションを使用すると、クラス・ファイルと変換後の Java プログラムのサイズが大きくなります。 `-o` オプションを使用する際には、アプリケーションの空間とパフォーマンス特性の両方のプロパティを考慮に入れなければなりません。

注: 通常は、`javac` の `-o` オプションは使用せずに、インライン化を後のフェーズに任せるのが最も良い方法です。

Java 変換プログラムでは、最適化レベル 30 と 40 についてインライン化が使用可能です。最適化レベル 30 では、1 つのクラス内の最終メソッドの一部のインライン化が可能になります。最適化レベル 40 では、ZIP ファイルまたは JAR ファイル内の最終メソッドのインライン化が可能になります。LICOPT パ

ラメーター・ストリング AllowInlining および NoAllowInlining を使用すると、メソッドのインライン化を制御することができます。iSeries インタープリターでは、メソッドのインライン化は実行されません。

Just-In-Time (JIT) コンパイラーはまた、最終メソッドのインライン化も実行します。これは、いつでも JIT コンパイラーがアクティブであるときに、インライン化が有益と判断した場合に自動的に実行されます。

## Java の例外処理のパフォーマンスの考慮事項

iSeries の例外アーキテクチャーは、柔軟な割り込みおよび再試行機能を備えており、言語の混合対話も可能にします。Java 例外を iSeries サーバーにスローすると、他のプラットフォームよりも負荷が大きくなります。Java 例外が通常のアプリケーション・パスでルーチンに沿って使用されるのでない限り、アプリケーション・パフォーマンス全体に影響を与えないようにしなければなりません。

## Java 呼び出しトレースのパフォーマンス測定ツール

Java メソッド呼び出しトレースは、各 Java メソッドに要した時間に関する、重要なパフォーマンス情報を提供します。

他の Java 仮想マシンでは、java コマンドで -prof (プロファイル) オプションを使用できるものもあります。iSeries サーバーでメソッド呼び出しトレースを可能にするには、「Java プログラムの作成 (CRTJVAPGM)」コマンド行で「パフォーマンス収集使用可能 (ENBPFCOL)」コマンドを指定しなければなりません。このキーワードを使用して Java プログラムを作成した後は、呼び出し/戻りトレース・タイプを含む Performance Explorer (PEX) 定義を使用して、メソッド呼び出しトレースのコレクションを開始できます。

「PEX 報告書の印刷 (PRTPEXRPT)」コマンドを使用して生成した呼び出し/戻りトレース出力は、トレースされた Java メソッドごとの各呼び出しの中央演算処理装置 (CPU) 時間を表示します。場合によっては、すべてのクラス・ファイル呼び出し戻りトレースで使用可能にすることができないことがあります。または、トレースには使用できないネイティブ・メソッドとシステム機能呼び出ししていることがあります。このような状況では、これらのメソッドまたはシステム機能で使用されたすべての CPU 時間が累積されます。次にこれが、最後に呼び出され、使用可能になった Java メソッドに報告されます。

## Java プロファイルのパフォーマンス測定ツール

システム全体の中央演算処理装置 (CPU) プロファイルによって、各 Java メソッドと、Java プログラムによって使用されたすべてのシステム機能に要した、相対的な CPU 時間が計算されます。

パフォーマンス・モニター・カウンター桁あふれ (\*PMCO) の実行サイクル・イベントをトレースする Performance Explorer (PEX) 定義を使用してください。通常は、サンプルがミリ秒単位の間隔で指定されます。有効なトレース・プロファイルを収集するには、2、3 分の CPU 時間が経過するまで Java アプリケーションを実行する必要があります。これによって、100,000 以上のサンプルが生成されるはずですが、PEX 報告書の印刷 (PRTPEXRPT) コマンドによって、アプリケーション全体で経過する CPU 時間のヒストグラムが作成されます。これには、すべての Java メソッドおよびシステム・レベルの活動すべてが含まれます。また、パフォーマンス・データ・コレクター (PDC) ツールは、iSeries サーバー上で実行されるプログラムのプロファイル情報を提供します。

注: CPU プロファイルでは、解釈される Java プログラムの相対的な CPU 使用状況は示されません。

### リンク集

パフォーマンス・データ・コレクター (PDC) ツール

パフォーマンス・データ・コレクター (PDC) ツールは、iSeries サーバー上で実行されるプログラムのプロファイル情報を提供します。



## Java Virtual Machine Profiler Interface

JVMPI (Java Virtual Machine Profiler Interface) は、Java 仮想マシンのプロファイル作成のための実験的なインターフェースで、Sun's Java 2 SDK, Standard Edition (J2SDK), version 1.2 で最初に発表され実装されました。

- | JVMTI は、JVMPI および Java Virtual Machine Debugger Interface (JVMDI) に置き換わるものです。
- | JVMTI には、JVMDI と JVMPI の両方のすべての機能、および新機能が備えられています。JVMTI は J2SE 5.0 の一部として追加されました。将来のリリースでは、JVMDI インターフェースおよび JVMPI インターフェースの提供は打ち切れ、JVMTI だけしか選択できなくなります。
- | JVMTI のインプリメントの詳細については、Sun Microsystems, Inc. の Web サイトの JVMTI リファレンス・ページ (英語) を参照してください。

JVMPI/JVMTI サポートは JVM および Just-in-time (JIT) コンパイラーにフックを置きます。このフックは、アクティブにされたときに、プロファイル作成エージェントにイベント情報を渡します。プロファイル作成エージェントは、統合言語環境 (ILE) サービス・プログラムとして実装されています。プロファイラーは、JVMPI/JVMTI イベントを使用可能にしたり使用不可にするため、JVM に制御情報を送信します。たとえば、プロファイラーはメソッドの入り口や出口フックには関心がなく、JVM にこれらのイベント通知を受け取りたくないと言うかもしれません。JVM および JIT には、そのイベントが使用可能な場合にプロファイル作成エージェントにイベント通知を送信する JVMPI/JVMTI イベント・フックが組み込まれています。プロファイラーは JVM にどのイベントに関心があるかを知らせ、JVM はそのイベントが発生するとプロファイラーにその通知を送信します。

サービス・プログラム QSYS/QJVAJVMPI は、JVMPI 関数を提供します。

- | QSYS ライブラリーに存在する QJVAJVMTI というサービス・プログラムは、JVMTI 機能をサポートします。

詳しくは、Sun Microsystems, Inc. の JVMPI を参照してください。

## Java パフォーマンス・データを収集する

iSeries サーバーの Java パフォーマンス・データを収集するには、以下のステップを行ってください。

1. 次のものを指定する Performance Explorer (PEX) 定義を作成します。
  - ユーザー定義の名前
  - データ収集のタイプ
  - ジョブ名
  - 収集したいシステム情報に関する一連のシステム・イベント

注: 必要な出力が `java_g -prof` タイプであり、Java プログラムの特定のジョブ名を知っている場合は、`*TRACE` 定義よりも `*STATS` の PEX 定義の方が適当です。

以下に `*STATS` 定義の例を示します。

```
ADDPDXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMSRV) DTAORG(*HIER)
TEXT('your stats definition')
```

この `*STATS` 定義では、実行されているすべての Java イベントが取得されるわけではありません。各自の Java セッションにある Java イベントのみプロファイルが作成されます。この操作モードでは、Java プログラムを実行する時間が増える可能性があります。

以下に、\*TRACE 定義の例を示します。

```
ADDPXDFN DFN(YOURDFN) TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT)
SLTEVT(*YES) PGMEVT(*JVAENTRY *JVAEXIT)
```

この \*TRACE 定義では、ENBPFCOL(\*ENTRYEXIT) を使用して作成されたシステムにあるすべての Java プログラムから、すべての Java 入り口イベントと出口イベントを収集します。そのため、Java プログラム・イベントの数、そして PEX データ収集の期間によっては、このタイプの収集の分析は、\*STATS トレースのときよりも遅くなることがあります。

2. PEX 定義のプログラム・イベント・カテゴリーの下にある \*JVAENTRY および \*JVAEXIT を使用可能にして、PEX が Java 入り口と出口を認識するようにします。

**注:** Java コードを Just-in-time (JIT) コンパイラーを使用して実行している場合は、直接処理するために CRTJVAPGM コマンドを使用する場合と同じように、入り口と出口を使用可能にすることはありません。代わりに、os400.enbprfcol システム・プロパティーを作成するときに、JIT は項目入り口があるコードと出口フックを生成します。

3. プログラム・イベントを iSeries パフォーマンス・データ収集プログラムに報告するために、Java プログラムを準備します。

これを行うには、パフォーマンス・データを変更したい Java プログラムで「Java プログラムの作成 (CRTJVAPGM)」コマンドを使用します。Java プログラムは、ENBPFCOL(\*ENTRYEXIT) パラメーターを使用して作成しなければなりません。

**注:** パフォーマンス・データの収集を行いたい Java プログラムごとに、このステップを繰り返す必要があります。このステップを実行しないと、PEX によってパフォーマンス・データは収集されず、Java パフォーマンス・データ・コンバーター (JPDC) ツールを実行しても出力は生成されません。

4. 「パフォーマンス・エクスプローラーの開始 (STRPEX)」コマンドを使用して、PEX データの収集を開始します。
5. 分析したいプログラムを実行します。

このプログラムの環境は実稼働環境であってはなりません。この収集により、短時間で多くのデータが生成されます。収集時間は 5 分に制限しなければなりません。この時間で実行された Java プログラムは、多くの PEX システム・データを生成します。収集されたデータが多過ぎる場合は、データを処理するために余計な時間が必要になります。

6. 「パフォーマンス・エクスプローラーの終了 (ENDPEX)」コマンドを使用して、PEX データの収集を終了します。

**注:** PEX データの収集を終了するのが初めてではない場合は、\*YES の置き換えファイルを指定しなければなりません。これを行わないと、データが保管されません。

7. JPDC ツールを実行してください。
8. 統合ファイル・システム・ディレクトリーをシステムに接続します。java\_g -prof ビューアーまたは Jinsight ビューアーのどちらかのビューアーを選択してください。

iSeries サーバーからこのファイルをコピーして、任意の適切なプロファイル・ツールの入力として使用できます。

## パフォーマンス・データ・コレクター・ツール

パフォーマンス・データ・コレクター (PDC) ツールは、iSeries サーバー上で実行されるプログラムのプロファイル情報を提供します。

多くの Java 仮想マシンの業界標準プロファイル・オプションは、`java_g` 機能の実装にかかっています。これは、Java 仮想マシンの特殊なデバッグ・バージョンで、`-prof` オプションを提供しています。Java プログラムへの呼び出しの際に、このオプションを指定します。このオプションを指定すると、プログラムの実行時に Java プログラムのどの部分が動作していたかを示すレコード・ファイルを Java 仮想マシンは生成します。Java 仮想マシンは、この情報をリアルタイムで生成します。

iSeries サーバーでは、Performance Explorer (PEX) 機能によって、プログラムおよびレコード固有のシステム・イベントが分析されます。DB2 データベースはこの情報を格納してから、SQL 関数を使用して検出します。PEX 情報は、Java プロファイル・データを作成する特定のプログラム情報のリポジトリです。このプロファイル・データは、`java_g -prof` プログラムのプロファイル・データと互換性があります。Java パフォーマンス・データ・コンバーター (JPDC) ツールは、特定の IBM ツールの `java_g -prof` プログラム出力および Jinsight と呼ばれる特定の IBM ツールのプログラム・プロファイル情報を提供します。

Java パフォーマンス・データの収集方法の詳細については、Java パフォーマンス・データを収集するを参照してください。

## Java パフォーマンス・データ・コンバーター・ツール

Java パフォーマンス・データ・コンバーター (JDPC) ツールは、iSeries サーバーで実行されている Java プログラムについての Java パフォーマンス・データを作成する手段となります。このパフォーマンス・データは、Sun Microsystems, Inc. の Java 仮想マシンの `java_g -prof` オプションのパフォーマンス・データ出力、および IBM Jinsight の出力と互換性があります。

注: JDPC ツールでは、読み取り可能な出力は生成されません。データを分析するには、`java_g -prof` を受け入れる Java プロファイル・ツール、または Jinsight データを使用してください。

JDPC ツールは、DB2/400 (JDBC を使用している) によって保管される iSeries Performance Explorer (PEX) データにアクセスします。このツールは、データを Jinsight または汎用パフォーマンス・タイプに変換します。その後、出力ファイルを統合ファイル・システムのユーザー指定の場所に保管します。

注: 指定した Java アプリケーションが iSeries サーバーで実行されているときに、PEX データを収集するには、適切な iSeries PEX データ収集プロシージャに従わなければなりません。プログラムの入り口および出口、または収集および保管プロシージャを含む PEX 定義を設定しなければなりません。PEX データの収集および PEX 定義の設定方法については、「Performance Tools for iSeries (SC41-5340)」を参照してください。

JPDC の実行方法については、Java パフォーマンス・データ・コンバーターを実行するを参照してください。

JPDC プログラムを開始するには、Qshell コマンド・ライン・インターフェース、または「Java プログラムの実行 (RUNJVA)」コマンドを使用します。

## Java パフォーマンス・データ・コンバーターを実行する

パフォーマンス・データを収集するために Java パフォーマンス・データ・コンバーター (JPDC) を実行するには、以下のステップに従います。

1. 最初の入力引数を入力する。 `java_g -prof` の場合は `general`、または Jinsight 出力の場合は `jinsight` と入力します。
2. 2 番目の入力引数を入力する。これは、データを収集するのに使用された Performance Explorer (PEX) 定義の名前を表します。

注: この名前は、この名前を用いた接続の内部処理の関係上、4 文字から 5 文字に制限する必要があります。

3. JPDC ツールが生成するファイルの名前を表す、3 番目の入力引数を入力する。

この生成されたファイルは、現在の統合ファイル・システム・ディレクトリーに書き込みます。cd (PF4) コマンドを使用して、統合ファイル・システムの現行ディレクトリーを指定します。

4. iSeries ホスト・リレーショナル・データベース項目の名前を表す、4 番目の入力引数を入力する。

「リレーショナル・データベース・ディレクトリー項目の処理 (WRKRDBDIRE)」コマンドを使用して、その名前を表示します。これは、\*LOCAL と指定されている唯一のリレーショナル・データベースです。

このコードを実行するには、/QIBM/ProdData/Java400/ext/JPDC.jar ファイルが iSeries サーバー上の Java クラスパスになくはなりません。プログラムの実行が終了すると、現行ディレクトリーにテキスト出力ファイルが入れられます。

JPDC は、iSeries コマンド行または Qshell 環境を使用して実行できます。詳細については、例: Java パフォーマンス・データ・コンバーターを実行するを参照してください。

#### 例: Java パフォーマンス・データ・コンバーターを実行する:

Java パフォーマンス・データ・コンバーター (JPDC) を実行するには、iSeries コマンド行または Qshell 環境のいずれかを使用できます。

#### iSeries コマンド行の使用:

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

1. iSeries コマンド行に「Java プログラムの実行 (RUNJVA)」コマンドまたは JAVA コマンドを入力する。
2. クラス・パラメーター行で com.ibm.as400.jpdc.JPDC を入力する。
3. パラメーター行で general pexdfn mydir/myfile myrdbdire を入力する。
4. クラスパスのパラメーター行で '/QIBM/ProdData/Java400/ext/JPDC.jar' を入力する。

注: '/QIBM/ProdData/Java400/ext/JPDC.jar' スtringが CLASSPATH 環境変数で存在する場合には、クラスパスは省略してください。CLASSPATH 環境変数にこの Stringを追加するには、「環境変数の追加 (ADDENVVAR)」コマンド、「環境変数の変更 (CHGENVVAR)」コマンド、または「環境変数の処理 (WRKENVVAR)」コマンドのいずれかを使用できます。

#### Qshell 環境の使用:

1. 「Qshell の開始 (STRQSH)」コマンドを入力して、Qshell インタープリターを開始する。
2. コマンド行で次のように入力します。

```
java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC
jinsight pexdfn mydir/myfile myrdbdire
```

注: '/QIBM/ProdData/Java400/ext/JPDC.jar' Stringが現行の環境に追加される場合には、クラスパスは省略してください。ADDENVVAR コマンド、CHGENVVAR、または WRKENVVAR コマンドのいずれかを使用して、現行の環境にこの Stringを追加できます。

背景情報については、Java パフォーマンス・データ・コンバーターを実行するを参照してください。

---

## IBM Developer Kit for Java 用のコマンドとツール

IBM Developer Kit for Java を使用するときには、Qshell インタープリター、または CL コマンドで Java ツールが使用できます。

Qshell インタープリターによる Java ツールは Sun Microsystems, Inc. の Java Development Kit で使用するツールと似ているので、Java でプログラムを作成した経験がある方は Qshell インタープリターで Java ツールを使用することをお勧めします。Qshell 環境の使用法については、Qshell インタープリターを参照してください。

iSeries のプログラマーは、iSeries サーバー環境固有の Java 用 CL コマンドを使用することをお勧めします。CL コマンドおよび iSeries ナビゲーター・コマンドの使用法については、この後で説明します。

IBM Developer Kit for Java では、次のコマンドおよびツールを使用できます。

- Qshell 環境。プログラム開発に通常必要とされる Java 開発ツールを使用できます。
- CL 環境。この場合、Java プログラムの最適化および管理には、CL コマンドを使用します。
- 406 ページの『Java でサポートされる iSeries ナビゲーター・コマンド』でも、最適化された Java プログラムを作成し、実行することができます。

## IBM Developer Kit for Java がサポートする Java ツール

IBM Developer Kit for Java では、以下のツールがサポートされます。

ajar 以外の上記の Java ツールは、いくつかの例外を除き、Sun Microsystems, Inc. の公式資料に記載された構文とオプションをサポートしています。これらの Java ツールすべての実行には、Qshell インタープリターを使用する必要があります。

Qshell インタープリターを起動するには、「Qshell の開始 (STRQSH または QSH)」コマンドを使用します。Qshell インタープリターの実行中は、「QSH コマンドの入力」画面が表示されます。Qshell のもとで実行される Java のツールやプログラムからの出力とメッセージは、すべてこの画面に表示されます。また、Java プログラムへの入力もこの画面から読み取られます。詳細については、Java Qshell コマンドを参照してください。

注: iSeries コマンド入力の機能を Qshell 内から直接使用することはできません。コマンド行を表示するには、F21 (CL コマンドの入力) を押してください。

### Java ツール

Java ツールの説明については、ここに記載するトピックを参照してください。

#### Java ajar ツール:

ajar ツールは、jar ツールの代替インターフェースであり、これは Java ARchive (JAR) ファイルを作成し、操作するために使用します。ajar ツールを使用して JAR ファイルと ZIP ファイルの両方を操作できます。

ajar ツールは jar ツールが行うのと同じように、JAR ファイルの内容をリストし、JAR ファイルから取り出し、新しい JAR ファイルを作成し、多くの ZIP 形式をサポートします。さらに、ajar ツールは既存の JAR ファイルのファイルの追加または削除をサポートします。

ajar ツールの運用には、Qshell インタープリターが必要です。詳細については、ajar - 代替 Java アーカイブを参照してください。



## Java appletviewer ツール:

Java appletviewer ツールを使用すると、Web ブラウザーを使わずにアプレットを実行できます。このツールは、Sun Microsystems, Inc. が提供する appletviewer ツールと互換性があります。

appletviewer ツールを使用するには、Native Abstract Window Toolkit (NAWT) を使用し、さらに sun.applet.AppletViewer クラスを使用するか、または Qshell インタープリターで appletviewer ツールを実行する必要があります。

以下は、sun.applet.AppletViewer クラスを使用して、TicTacToe 実例を実行する例です。実例のロード方法については、サンプル・ファイルの抽出方法を参照してください。

コマンド行から、次のように入力します。

```
cd '/home/MyUserID/demo/applets/TicTacToe'
```

JDK 1.3 の場合は、以下のコマンドを実行します。

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
PROP((os400.class.path.rawt 2)(java.version 1.3))
```

JDK 1.4 の場合は、以下のコマンドを実行します。

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.4))
```

1 | JDK 1.5 の場合は、以下のコマンドを実行します。

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html')
prop((os400.awt.native true)(java.version 1.5))
```

以下は、Qshell インタープリターで appletviewer ツールを使用して、TicTacToe 実例を実行する例です。実例のロード方法については、サンプル・ファイルの抽出方法を参照してください。

対応するコマンドは、次のとおりです。

```
cd /home/MyUserID/demo/applets/TicTacToe
```

JDK 1.3 の場合は、以下のコマンドを実行します。

```
Appletviewer -J-Dos400.class.path.rawt=2 -J-Djava.version=1.3 example1.html
```


JDK 1.4 の場合は、以下のコマンドを実行します。

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.4 example1.html
```

1 | JDK 1.5 の場合は、以下のコマンドを実行します。

```
Appletviewer -J-Dos400.awt.native=true -J-Djava.version=1.5 example1.html
```

注: -J は AppletViewer の実行時フラグです。-D はプロパティです。

appletviewer ツールの詳細については、appletviewer tool by Sun Microsystems, Inc.  を参照してください。

## サンプル・ファイルの抽出方法:

以下の手順は、Java appletviewer ツールを実行する前にサンプル・ファイルを抽出する 1 つの方法です。この手順では、サンプル・ファイルをホーム・ディレクトリーに抽出することを想定しています。

1. コマンド行に「Qshell の開始 (QSH)」コマンドを入力する。



2. ご使用のユーザー ID 用のホーム・レベル統合ファイル・システム (IFS) ディレクトリーが存在しない場合、これを作成する。

```
mkdir /home/MyUserID
```

3. IFS ディレクトリー内に demo ディレクトリーを作成する。

```
mkdir /home/MyUserID/demo
```

4. demo ディレクトリーに移動する。

```
cd /home/myUserId/demo
```

5. JDK 1.3 の場合は、コマンド行に以下を入力して、demo ファイルを抽出する。

```
jar xf /QIBM/ProdData/Java400/jdk13/demo.zip
```

JDK 1.4 の場合は、次のコマンドを使用します。

```
jar xf /QIBM/ProdData/Java400/jdk14/demo.jar
```

| JDK 1.5 の場合は、次のコマンドを使用します。

| 

```
jar xf /QIBM/ProdData/Java400/jdk15/demo.jar
```

#### | **Java apt ツール:**

| Java apt ツールは、プログラム注釈を処理します。

| apt ツールは、JDK 1.5 およびそれ以降のバージョンでのみ使用できます。 apt ツールの運用には、

| Qshell インタープリターが必要です。

| apt ツールの詳細については、apt tool by Sun Microsystems, Inc.  を参照してください。

#### **Java extcheck ツール:**

Java 2 SDK (J2SDK), Standard Edition バージョン 1.2 以上では、extcheck ツールが JAR ファイルと現在インストールされている拡張 JAR ファイルの間のバージョンの矛盾を検出します。このツールは、Sun Microsystems, Inc. によって提供される keytool と互換性があります。

extcheck ツールの運用には、Qshell インタープリターが必要です。

extcheck ツールの詳細については、extcheck tool by Sun Microsystems, Inc. を参照してください。

#### **Java idlj ツール:**

idlj ツールは、指定された Interface Definition Language (IDL) ファイルから Java バインディングを生成します。

| idlj ツールは、IDL-to-Java コンパイラーとも呼ばれています。このツールは、Sun Microsystems, Inc. によって提供される idlj ツールと互換性があります。このツールは、Java Development Kit 1.3 および後続のバージョンでのみ機能します。

idlj ツールの詳細については、idlj tool by Sun Microsystems, Inc.  を参照してください。

#### **Java jar ツール:**

jar ツールは、複数のファイルをまとめて 1 つの Java アーカイブ (JAR) ファイルにします。このツールは、Sun Microsystems, Inc. が提供する jar ツールと互換性があります。

jar ツールを使用するには、Qshell インタープリターが必要です。

jar ツールの代替インターフェースについては、JAR ファイルを作成および操作するための ajar ツールを参照してください。

iSeries ファイル・システムの詳細については、統合ファイル・システムまたは統合ファイル・システム内のファイルを参照してください。

jar ツールの詳細については、jar tool by Sun Microsystems, Inc. を参照してください。

### Java jarsigner ツール:

Java 2 SDK (J2SDK)、標準版、バージョン 1.2 以降において、jarsigner ツールは、JAR ファイルの署名と、署名された JAR ファイル上の署名の検査を行います。

jarsigner ツールは、JAR ファイルの署名のために秘密鍵を見つけることが必要なときには、keytool によって作成および管理される鍵ストアにアクセスします。J2SDK では、jarsigner および keytool ツールが javakey ツールに取って代わります。このツールは、Sun Microsystems, Inc. によって提供される jarsigner ツールと互換性があります。

jarsigner ツールを使用するには、Qshell インタープリターが必要です。

jarsigner ツールの詳細については、jarsigner tool by Sun Microsystems, Inc. を参照してください。

### Java javac ツール:

javac ツールは、Java プログラムをコンパイルします。このツールは、Sun Microsystems, Inc. によって提供される javac ツールと互換性がありますが、1 つの例外があります。

#### -classpath

デフォルト・クラスパスを一時変更しません。その代わりに、システムのデフォルト・クラスパスに付加されます。-classpath オプションは、CLASSPATH 環境変数をオーバーライドします。

javac ツールを使用するには、Qshell インタープリターが必要です。

iSeries サーバーにデフォルトとして JDK 1.1.x がインストールされているが、1.2 バージョン以降の java コマンドを実行することが必要な場合は、次のコマンドを入力してください。

```
javac -djava.version=1.2 <my_dir> MyProgram.java
```

javac ツールの詳細については、javac tool by Sun Microsystems, Inc. を参照してください。

### Java javadoc ツール:

javadoc ツールは、API 文書を作成します。このツールは、Sun Microsystems, Inc. によって提供される javadoc ツールと互換性があります。

javadoc ツールを使用するには、Qshell インタープリターが必要です。

javadoc ツールの詳細については、javadoc tool by Sun Microsystems, Inc. を参照してください。

### Java javah ツール:

javah ツールは、Java ネイティブ・メソッドの実装を容易にします。このツールは、Sun Microsystems, Inc. によって提供される javah ツールと互換性がありますが、いくつかの例外があります。

注: ネイティブ・メソッドを作成することは、アプリケーションが 100% pure Java ではないことを意味します。さらに、アプリケーションがプラットフォーム間で直接に移植可能でないことを意味します。ネイティブ・メソッドは、その性質上、プラットフォームまたはシステムに固有です。ネイティブ・メソッドを使用すると、アプリケーションの開発および保守にかかるコストが増加する可能性があります。

javah ツールを使用するには、Qshell インタープリターが必要です。このツールは、Java クラス・ファイルを読み取り、現行作業ディレクトリーで C 言語ヘッダー・ファイルを作成します。作成されるヘッダー・ファイルは、iSeries ストリーム・ファイル (STMF) です。それを iSeries サーバー上の C プログラムに組み込むためには、まず、ファイル・メンバーにコピーしなければなりません。

javah ツールは、Sun Microsystems, Inc. によって提供されるツールと互換性があります。ただし、次のオプションを指定しても、iSeries サーバーでは無視されます。

- td iSeries サーバー上の javah ツールは、一時ディレクトリーを必要としません。
- stubs iSeries サーバー上の Java では、Java ネイティブ・インターフェース (JNI) 形式のネイティブ・メソッドのみがサポートされます。スタブが必要とされるのは、JNI 以前の形式のネイティブ・メソッドの場合だけです。
- trace これは、iSeries サーバー上の Java によってサポートされない .c スタブ・ファイル出力に関連するオプションです。
- v サポートされません。

注: -jni オプションは、必ず指定しなければなりません。iSeries サーバー・システムでは、JNI 以前のネイティブ・メソッドの実装はサポートされません。

javah ツールの詳細については、javah tool by Sun Microsystems, Inc. を参照してください。

### Java javap ツール:

javap ツールは、コンパイル済みの Java ファイルを逆アセンブルし、Java プログラムのソース・コードを出力します。これは、システムで元のソース・コードが使用可能でなくなったときに役立つ可能性があります。

- b このオプションは無視されます。iSeries サーバー上の Java では、Java Development Kit (JDK) 1.1.4 以上のみがサポートされるため、逆方向の互換性は必要ありません。
- p iSeries サーバーでは、-p は無効なオプションです。-private という完全なつづりを使用しなければなりません。
- verify このオプションは無視されます。javap ツールは、iSeries サーバー上では検査を行いません。

javap ツールを使用するには、Qshell インタープリターが必要です。

注: javap ツールを使用してクラスを逆アセンブルすると、それらのクラスについてのライセンス契約に違反する可能性があります。javap ツールを使用する前に、クラスについてのライセンス契約を確認してください。

javap ツールの詳細については、javap tool by Sun Microsystems, Inc. を参照してください。

### Java keytool:

Java 2 SDK (J2SDK), Standard Edition バージョン 1.2 以降において、keytool は、公開鍵と秘密鍵の対、および自己署名証明書を作成し、鍵ストアを管理します。J2SDK では、jarsigner および keytool ツールが javakey ツールに取って代わります。このツールは、Sun Microsystems, Inc. によって提供される keytool と互換性があります。

keytool を使用するには、Qshell インタープリターが必要です。

keytool の詳細については、keytool by Sun Microsystems, Inc. を参照してください。

#### **Java native2ascii ツール:**

native2ascii ツールは、ネイティブ・コードの文字 (Latin 1 でも Unicode でもない文字) を使用したファイルを、Unicode 文字を使用したファイルに変換します。このツールは、Sun Microsystems, Inc. によって提供される native2ascii ツールと互換性があります。

native2ascii ツールの運用には、Qshell インタープリターが必要です。

native2ascii ツールの詳細については、native2ascii tool by Sun Microsystems, Inc. を参照してください。

#### **Java orbd ツール:**

orbd ツールは、CORBA 環境においてサーバー上の persistent オブジェクトを容易に見付けて呼び出すためのサポートをクライアントに提供します。

ORBD は、Transient Naming Service (tnameserv) の代わりに使用され、Transient Naming Service と Persistent Naming Service の両方を含んでいます。orbd ツールは、Server Manager、Interoperable Naming Service、および Bootstrap Name Server の機能を組み入れています。servertool と共に使用すると、クライアントがサーバーへのアクセスを望むときに、Server Manager がそのサーバーを見付けて、登録し、活性化します。

| orbd ツールの詳細については、orbd tool by Sun Microsystems, Inc. を参照してください。

#### **| Java pack200 ツール:**

| pack200 ツールは、JAR ファイルを圧縮して pack200 ファイルにする Java アプリケーションです。

| pack200 ツールは、JDK 1.5 およびそれ以降のバージョンでのみ使用できます。pack200 ツールの運用には、Qshell インタープリターが必要です。

| 詳細については、pack200 tool by Sun Microsystems, Inc.  を参照してください。

#### **| 関連概念**

| 404 ページの『Java unpack200 ツール』

| Java unpack200 ツールは、pack200 ファイルを解凍して JAR ファイルに戻します。

#### **Java policytool:**

Java 2 SDK, Standard Edition では、policytool は、ご使用のインストールの Java セキュリティー・ポリシーを定義する外部ポリシー構成ファイルを作成および変更します。このツールは、Sun Microsystems, Inc. が提供する policytool ツールと互換性があります。

policytool は、Qshell インタープリターおよび Native Abstract Window Toolkit (NAWT) を使用する際に提供されるグラフィカル・ユーザー・インターフェース (GUI) です。詳しくは、IBM Developer Kit for Java Native Abstract Window Toolkit を参照してください。

policytool の詳細については、policytool by Sun Microsystems, Inc. を参照してください。

#### **Java rmic ツール:**

rmic ツールは、Java オブジェクトのスタブ・ファイルとクラス・ファイルを生成します。このツールは、Sun Microsystems, Inc. が提供する rmic ツールと互換性があります。

rmic ツールの運用には、Qshell インタープリターが必要です。

rmic ツールの詳細については、rmic tool by Sun Microsystems, Inc. を参照してください。

#### **Java rmid ツール:**

Java 2 SDK (J2SDK), Standard Edition では、rmid ツールが活動化システム・デーモンを開始します。ですから、オブジェクトは Java 仮想マシンで登録および活動化することができます。このツールは、Sun Microsystems, Inc. が提供する rmid ツールと互換性があります。

rmid ツールの運用には、Qshell インタープリターが必要です。

rmid ツールの詳細については、rmid tool by Sun Microsystems, Inc. を参照してください。

#### **Java rmiregistry ツール:**

rmiregistry ツールは、指定されたポートで遠隔オブジェクト・レジストリーを開始します。このツールは、Sun Microsystems, Inc. が提供する rmiregistry ツールと互換性があります。

rmiregistry ツールの運用には、Qshell インタープリターが必要です。

rmiregistry ツールの詳細については、rmiregistry tool by Sun Microsystems, Inc. を参照してください。

#### **Java serialver ツール:**

serialver ツールは、1 つまたは複数のクラスについて、バージョン番号または固有な通番で表された ID を戻します。このツールは、Sun Microsystems, Inc. が提供する serialver ツールと互換性があります。

serialver ツールの運用には、Qshell インタープリターが必要です。

serialver ツールの詳細については、serialver tool by Sun Microsystems, Inc. を参照してください。

#### **Java servertool:**

servertool は、アプリケーション・プログラマーが、永続サーバーを登録、登録抹消、起動、およびシャットダウンするためのコマンド行インターフェースを提供します。

servertool の詳細については、servertool by Sun Microsystems, Inc. を参照してください。

#### **Java tnameserv ツール:**

Java 2 SDK (J2SDK), Standard Edition バージョン 1.3 以降では、tnameserv (Transient Naming Service) ツールを使用すると、命名サービスにアクセスできます。このツールは、Sun Microsystems, Inc. が提供する tnameserv ツールと互換性があります。

| tnameserv ツールを使用するには、Qshell インタープリターが必要です。

#### | **Java unpack200 ツール:**

| Java unpack200 ツールは、pack200 ファイルを解凍して JAR ファイルに戻します。

| unpack200 ツールは、JDK 1.5 およびそれ以降のバージョンでのみ使用できます。 unpack200 ツールの運

| 用には、Qshell インタープリターが必要です。

| 詳細については、unpack200 tool by Sun Microsystems, Inc.  を参照してください。

#### | **関連概念**

| 402 ページの『Java pack200 ツール』

| pack200 ツールは、JAR ファイルを圧縮して pack200 ファイルにする Java アプリケーションです。

## **Qshell の Java コマンド**

java Qshell コマンドは、Java プログラムを実行します。このツールは、Sun Microsystems, Inc. によって提供される java ツールと互換性がありますが、いくつかの例外があります。

IBM Developer Kit for Java では、java Qshell コマンドの以下のようなオプションが無視されます。

| オプション         | 説明   |
|---------------|--|
| -cs           | このオプションはサポートされていません。                                   |
| -checksource  | このオプションはサポートされていません。                                   |
| -debug        | このオプションは iSeries 内部デバッガーによってサポートされます。                  |
| -noasyncgc    | IBM Developer Kit for Java では、ガーベッジ・コレクションは常に実行されています。 |
| -noclassgc    | IBM Developer Kit for Java では、ガーベッジ・コレクションは常に実行されています。 |
| -prof         | iSeries サーバーには独自のパフォーマンス測定ツールがあります。                    |
| -ss           | このオプションは iSeries サーバーでは適用できません。                        |
| -oss          | このオプションは iSeries サーバーでは適用できません。                        |
| -t            | iSeries サーバーでは、独自のトレース機能が使用されます。                       |
| -verify       | iSeries サーバーでは、検査は必ず行われます。                             |
| -verifyremote | iSeries サーバーでは、検査は必ず行われます。                             |
| -noverify     | iSeries サーバーでは、検査は必ず行われます。                             |

iSeries サーバーでは、-classpath オプションはデフォルトのクラスパスをオーバーライドしません。その代わりに、システムのデフォルト・クラスパスに付加されます。 -classpath オプションは、CLASSPATH 環境変数をオーバーライドします。

java Qshell コマンドは、iSeries サーバーの新しいオプションをサポートします。サポートされている新しいオプションを以下に示します。



| オプション                            | 説明   |
|----------------------------------|--|
| -chkpath                         | このオプションは、CLASSPATH で指定されたディレクトリに対するパブリック書き込みアクセス権があるかどうかをチェックします。  |
| -opt                             | このオプションは、最適化レベルを指定します。   |
| -Xrun[:]                         | JVM の始動時に、サービス・プログラムと JVM_OnLoad のオプションのパラメーター・ストリングが機能していることを示すメッセージが表示されます。  |
| -agentlib:                       | VM エージェントを含む i5/OS サービス・プログラムを示します。VM は、始動時に、ライブラリー・リストに含まれている i5/OS ライブラリーから、このサービス・プログラムをロードしようとしています。   |
| -agentpath:                      | このオプションの後に続く絶対パスからライブラリーをロードします。ライブラリー名拡張は行われず、オプションは始動時にエージェントに渡されます。   |
| -javaagent:<jarpath>[=<options>] | java.lang.instrument パッケージで使用する Java プログラム言語エージェントをロードします。<br><br><i>jarpath</i> は、エージェント JAR ファイルへのパスです。<br><i>options</i> は、エージェント・オプションです。複数のエージェントを作成するために、-javaagent:<jarpath>[=<options>] を同じコマンド行上で複数回使用できます。複数のエージェントが同じ <i>jarpath</i> を使用できます。 |

CL コマンドのリファレンス情報の「Java プログラムの実行 (RUNJVA)」コマンドでは、これら新しいオプションが詳しく説明されています。また、CL コマンドのリファレンス情報の「Java プログラムの作成 (CRTJVAPGM)」コマンド、「Java プログラムの削除 (DLTJVAPGM)」コマンド、および「Java プログラムの表示 (DSPJVAPGM)」コマンドの項に、Java プログラムの管理に関する情報が記載されています。

Qshell の java コマンドの運用には、Qshell インタープリターが必要です。

Qshell の java コマンドの詳細については、java tool by Sun Microsystems, Inc. を参照してください。

## Java でサポートされる CL コマンド

IBM Developer Kit for Java は、以下の CL コマンドをサポートしています。

- 「Java プログラム分析 (ANZJVAPGM)」コマンドは Java プログラムを分析し、そのクラスをリストして、それぞれのクラスの現行の状況を表示します。
- 「Java 仮想マシンの分析 (ANZJVM)」コマンドは、Java 仮想マシン (JVM) 内の情報を設定および取得します。このコマンドは、活動中のクラスの情報を戻すため、Java プログラムのデバッグを行うときに便利です。
- 「Java プログラムの変更 (CHGJVAPGM)」コマンドは、Java プログラムの属性を変更します。
- 「Java プログラムの作成 (CRTJVAPGM)」コマンドは、Java クラス・ファイル、ZIPファイル、または JAR ファイルから、iSeries サーバーの Java プログラムを作成します。
- 「Java プログラムの削除 (DLTJVAPGM)」コマンドは、Java クラス・ファイル、ZIP ファイル、または JAR ファイルに関連付けられている iSeries の Java プログラムを削除します。

- 「Java プログラムの表示 (DSPJVAPGM)」 コマンドは、iSeries の Java プログラムに関する情報を表示します。
- 「Java 仮想マシンのジョブの表示 (DSPJVMJOB)」 コマンドは、プログラム一時修正 (PTF) の適用の管理を支援するために、アクティブ JVM ジョブに関する情報を表示します。 DSPJVMJOB の詳細については、557 ページの『プログラム一時修正を適用する』を参照してください。
- 「Java 仮想マシンのダンプ (DMPJVM) コマンドは、指定したジョブの Java 仮想マシンに関する情報をスプール・プリンター・ファイルにダンプします。
- JAVA コマンドと「Java プログラムの実行 (RUNJVA)」 コマンドは、iSeries の Java プログラムを実行します。

詳しくは、以下のページを参照してください。

ANZJVM コマンドの使用上の考慮事項

ライセンス内部コード・オプション・パラメーター・ストリング

プログラムおよび CL コマンド API

## ANZJVM コマンドの使用上の考慮事項

ANZJVM の実行できる時間の長さが原因で、ANZJVM が完了できる前に JVM が終了してしまう高い可能性があります。JVM が終了してしまった場合には、ANZJVM は取得できたデータと共に、JVAB606 メッセージ (ANZJVM の処理中に JVM が終了した) を戻します。

また、JVM がハンドルできるクラスの数の上限はありません。ハンドルできるクラスの数を上回った場合、ANZJVM はハンドルできたデータと共に、報告されなかった追加の情報があることを示すメッセージを戻します。データの切り捨てが必要な場合は、ANZJVM は可能なだけの情報を戻します。

内部パラメーターでの長さは、3600 秒 (1 時間) に制限されています。ANZJVM が情報を戻すことのできるクラスの数は、システムの記憶域の量によって限定されます。

## Java でサポートされる iSeries ナビゲーター・コマンド

iSeries ナビゲーターは、Windows デスクトップのグラフィカル・インターフェースです。これは iSeries Access for Windows の一部であり、管理者やユーザーが日常業務を行うのに必要な iSeries の多数の機能を搭載しています。

iSeries ナビゲーターは、Java を iSeries Access for Windows の File Systems オプションに含まれるプラグインとしてサポートしています。iSeries ナビゲーターの Java プラグインを使用するには、iSeries サーバー上に IBM Developer Kit for Java をインストールする必要があります。次に、Java プラグインをパーソナル・コンピューターにインストールするため、Client Access フォルダーの選択セットアップを使用して、ファイル・システムを選択します。

クラス・ファイル、JAR ファイル、ZIP ファイル、および Java ファイルは統合ファイル・システムに常駐します。iSeries ナビゲーターは、右側のペインにこれらのファイルを表示します。使用したいクラス、JAR ファイル、ZIP ファイル、または java ファイルを右クリックします。これにより、コンテキスト・メニューが表示されます。

コンテキスト・メニューから「関連 Java プログラム」->「新規...」の順に選択すると、Java 変換プログラムが起動されて、選択したクラス、JAR、または ZIP ファイルに関連付けられた iSeries Java プログラムが作成されます。ダイアログ・ボックスで、プログラムの実行方法を詳しく指定できます。プログラムは、Java 変換形式にすることも Java インタープリター形式にすることもできます。

注: 変換形式を選択すると、クラス・ファイル内のバイトコードが RISC 命令に変換されるので、インタープリター形式を使用した場合に比べてパフォーマンスが高くなります。

コンテキスト・メニューから「**関連 Java プログラム**」-->「**編集...**」の順に選択すると、Java クラス・ファイル、ZIP ファイル、または JAR ファイルに付加される Java プログラムの属性が変更されます。

コンテキスト・メニューから「**関連 Java プログラム**」-->「**実行...**」の順に選択すると、iSeries サーバーでクラス・ファイルが実行されます。JAR ファイルまたは ZIP ファイルを選択して、その中にあるクラス・ファイルを実行することもできます。表示されるダイアログで、プログラムの実行方法を詳しく指定できます。「**関連 Java プログラム**」-->「**新規...**」の順に選択済みの場合は、指定したクラス・ファイルに関連付けられている iSeries Java プログラムを使ってプログラムが実行されます。iSeries Java プログラムがクラス・ファイルにまだ関連付けられていない場合は、プログラムの実行前に iSeries Java プログラムが作成されます。

コンテキスト・メニューから、「**関連 Java プログラム**」-->「**削除...**」の順に選択すると、指定したクラス・ファイル、JAR ファイル、または ZIP ファイルに関連付けられている iSeries Java プログラムが削除されます。

コンテキスト・メニューから「**プロパティ**」を選択すると、「**Java プログラム**」および「**Java オプション**」タブを含むプロパティ・ダイアログ・ボックスが表示されます。これらのタブから、関連付けられている iSeries Java プログラムがどのようにクラス・ファイル、JAR ファイル、または ZIP ファイルに対して作成されたか、という点の詳細を見ることができます。

注: これらのパネルは、「Java プログラムの表示」に関する情報です。

コンテキスト・メニューから「**Java ファイルのコンパイル**」を選択すると、選択した java ファイルがクラス・ファイルのバイトコードに変換されます。

iSeries ナビゲーターの「**新規 Java プログラム**」、「**Java プログラムの編集**」、「**Java プログラムの実行**」、「**Java プログラム (Java Programs)**」、「**Java オプション (Java Options)**」、「**Java ファイルのコンパイル**」、および「**Java プログラムの削除**」ダイアログのパラメーターとオプションについては、iSeries ナビゲーターに付属のヘルプ情報を参照してください。

---

## サーバーで実行する Java プログラムをデバッグする

サーバーで実行する Java プログラムのデバッグとトラブルシューティングには、いくつかの方法があります。これには、IBM iSeries システム・デバッガー、サーバー対話式画面、Java Debug Wire Protocol 対応デバッガー、および Java Watcher があります。

以下にいくつかのオプションを示します (ただし、この情報はすべての可能性を示すものではありません)。iSeries サーバー上で実行する Java プログラムをデバッグする最も簡単な方法の 1 つは、IBM iSeries System Debugger を使用することです。IBM iSeries System Debugger は、iSeries サーバーのデバッグ機能をより使いやすくするグラフィカル・ユーザー・インターフェース (GUI) を提供します。

Java プログラムはサーバーの対話式画面を使用してデバッグできますが、iSeries System Debugger は、同じ機能を実行できる、より使いやすい GUI を提供しています。

さらに、iSeries Java 仮想マシン (JVM) は、Java Platform Debugger Architecture の一部である Java Debug Wire Protocol (JDWP) をサポートしています。JDWP 対応デバッガーでは、異なるオペレーティング・システムを稼働しているクライアントからリモート・デバッグを実行することができます。(IBM iSeries

Debugger でも、同様の方法でリモート・デバッグを実行できます。ただし、これは JDWP は使用しません。) このような JDWP 対応プログラムの 1 つは、Eclipse プロジェクト・ユニバーサル・ツール・プラットフォームの Java デバッガーです。

プログラムの実行時間が長くなるとパフォーマンスが低下する場合は、誤ってメモリー・リークがコーディングされている可能性があります。Java Watcher を使用すれば、時間を追って Java アプリケーション・ヒープ分析とオブジェクト作成プロファイルの作成を行うことにより、プログラムをデバッグして、メモリー・リークを検出することができます。

IBM iSeries System Debugger

417 ページの『Java Platform Debugger Architecture』

Java Platform Debugger Architecture (JPDA) は、JVM Debug Interface/JVM Tool Interface、Java Debug Wire Protocol、および Java Debug Interface で構成されます。JPDA のこれらの部分はすべて、デバッグ操作を実行するために JDWP を使用するデバッガーのフロントエンドを使用可能にします。デバッガー・フロントエンドは、リモート側で実行することもできますし、iSeries アプリケーションとして実行することもできます。

Java development tool debug

Eclipse project Web site

JavaWatcher

## i5/OS コマンド行から Java プログラムをデバッグする

i5/OS コマンド行から Java プログラムをデバッグする場合は、以下のいずれかを選択してください。

- Java プログラムをデバッグする
- Java およびネイティブ・メソッド・プログラムをデバッグする
- 別の画面から Java プログラムをデバッグする
- カスタム・クラス・ローダーを通してロードされた Java クラスをデバッグする
- デバッグ・サーブレット

Java プログラムをデバッグするときに、Java プログラムは実際にはバッチ即時 (BCI) ジョブの Java 仮想マシンで実行されます。ソース・コードが対話式画面に表示されますが、Java プログラムはそこでは実行されません。これは、別のジョブ (サービス・ジョブ) で実行されます。Java プログラムが終了するときに、サービス・ジョブは終了し、「サービスされたジョブは終了しました。(Job being serviced ended)」というメッセージが表示されます。

Just-In-Time (JIT) コンパイラーで動作している Java プログラムはデバッグすることができません。ファイルに関連付けされた Java プログラムがない場合、デフォルトでは JIT で実行されます。デバッグを許可するため、いくつかの方法でこれを使用不可にすることができます。

- Java 仮想マシンを開始する時に、プロパティー `java.compiler=NONE` を指定する。
- 「Java プログラムの実行 (RUNJVA)」コマンドで `OPTION(*DEBUG)` を指定する。
- 「Java プログラムの実行 (RUNJVA)」コマンドで `INTERPRET(*YES)` を指定する。
- `CRTJVAPGM OPTIMIZATION(10)` を使って、Java 仮想マシンを開始する前に関連付けされた Java プログラムを作成する。

**注:** これらの解決策は、実行中の Java 仮想マシンには効果がありません。開始されていない Java 仮想マシンでこれらの方法を使う場合以外は、デバッグのために Java 仮想マシンを停止し、再始動する必要があります。

「Java プログラムの実行 (RUNJVA)」コマンドで \*DEBUG オプションを指定すると、2 つのジョブ間のインターフェースが確立されます。

システム・デバッガーについては、WebSphere Development Studio: ILE C/C++ Programmer's Guide (SC09-2712-04) およびオンライン・ヘルプの情報を参照してください。

## Java プログラムをデバッグする

iSeries サーバー上で実行する Java プログラムをデバッグする最も簡単な方法は、IBM iSeries System Debugger を使用することです。IBM iSeries System Debugger は、iSeries サーバーのデバッグ機能をより使いやすくするグラフィカル・ユーザー・インターフェースを提供します。

iSeries System Debugger を使用して iSeries サーバー上で実行する Java プログラムをデバッグおよびテストする方法については、IBM iSeries System Debugger を参照してください。

望むなら、サーバーの対話式表示を使用することができます。プログラムを実行する前にソース・コードを表示するには \*DEBUG オプションを使用します。これにより、停止点を設定したり、プログラムを 1 ステップずつ実行して、プログラムの実行中にエラーを分析したりできます。

Java プログラムをデバッグするには、以下のステップに従ってください。

1. javac ツールの -g オプションである DEBUG オプションを使って、Java プログラムをコンパイルする。詳細については、\*DEBUG オプションを使って Java プログラムをデバッグするを参照してください。
2. クラス・ファイル (.class) とソース・ファイル (.java) を iSeries サーバー上の同じディレクトリーに挿入する。
3. iSeries コマンド行で「Java プログラムの実行 (RUNJVA)」を使って、Java プログラムを実行する。「Java プログラムの実行 (RUNJVA)」コマンドで OPTION(\*DEBUG) を指定する。

クラスだけがデバッグされます。CLASS キーワードで JAR ファイル名を入力した場合は、OPTION(\*DEBUG) はサポートされません。

4. Java プログラムのソースが表示される。
5. F6 (停止点の追加/消去) を押して停止点を設定するか、または F10 (ステップ) を押してプログラム内に入る。停止点の設定の詳細については、停止点の設定を参照してください。ステップの詳細については、Java プログラムを 1 ステップずつ実行して、デバッグするを参照してください。

### ヒント:

1. 停止点とステップを使用するときには、Java プログラムの論理フローをチェックしてから、必要に応じて変数を表示および変更してください。
2. RUNJVA コマンド上で OPTION(\*DEBUG) を使用すると、Just-In-Time (JIT) コンパイラーが使用できなくなります。Java プログラムに関連付けられていないファイルは、インタープリット・モードで実行されます。

### \*DEBUG オプションを使って Java プログラムをデバッグする:

プログラムを実行する前にソース・コードを表示するには、\*DEBUG オプションを使用します。\*DEBUG オプションを指定すると、コード内に停止点を設定できます。

\*DEBUG オプションを使用するには、コマンド行で「Java プログラムの実行 (RUNJVA)」コマンドを入力し、それに続けてクラス・ファイルの名前と OPTION(\*DEBUG) を入力します。たとえば、iSeries コマンド行は次のようになります。







```

1=プログラムの追加   4=プログラムの除去   5=モジュール・ソースの表示
8=モジュール停止点の処理

OPT      PROGRAM/MODULE      LIBRARY      TYPE
                HELLOD                *LIBL                *SRVPGM
                *CLASS                選択

                                                                    終わり

コマンド
====>
F3=終了   F4=プロンプト   F5=最新表示   F9=コマンド複写   F12=取消し
F22=クラス・ファイル名の表示

```

• デバッグするクラスを追加するときには、「PROGRAM/MODULE」入力フィールドよりも長いパッケージ修飾クラス名を入力することが必要になる可能性があります。より長い名前を入力するには、以下のステップに従ってください。

1. オプション 1 (プログラムの追加) を入力します。
2. 「PROGRAM/MODULE」フィールドをブランクのままにします。
3. 「LIBRARY」フィールドを \*LIBL のままにします。
4. 「TYPE」として \*CLASS を入力します。
5. Enter を押します。
6. パッケージ修飾クラス・ファイル名を入力するためのスペースがあるポップアップ・ウィンドウが表示されます。

### 停止点の設定:

プログラムの実行は停止点を使って制御できます。停止点は、特定のステートメントで実行中のプログラムを停止させます。

停止点を設定するには、以下のステップを実行してください。

1. 停止点を設定したいコードの行にカーソルを置く。
2. F6 (停止点の追加/消去) を押して、停止点を設定する。
3. F12 (再開) を押してプログラムを実行する。

注: 停止点を設定したコード行が実行される直前に、プログラム・ソースが表示され、停止点に達したことが示されます。

```

-----+
                                モジュール・ソースの表示
現行スレッド:  00000019      停止スレッド:  00000019
クラス・ファイル名:  HelloD
35 public static void main(String[] args)
36 {
37     int i,j,h,B[],D[][];
38     HelloD A=new HelloD();
39     A.myHelloD = A;
40     HelloD C[];

```

```

41  C = new Hellod[5];
42  for (int counter=0; counter<2; counter++) {
43      C[counter] = new Hellod();
44      C[counter].myHellod = C[counter];
45  }
46  C[2] = A;
47  C[0].myString = null;
48  C[0].myHellod = null;


49  A.method1();
デバッグ . . .

F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示
F12=再開  F17=ウォッチ変数  F18=ウォッチの処理  F24=キーの続き
停止点が行 41 に追加されました。

```

停止点に達したときに、現行スレッド内で到達した停止点だけを設定したい場合は、TBREAK コマンドを使用してください。

システム・デバッガー・コマンドの詳細については、「WebSphere Development Studio: ILE C/C++

Programmer's Guide (SC09-2712)  およびオンライン・ヘルプの情報を参照してください。

停止点でプログラムの実行が停止したときに変数を評価することについては、Java プログラム中の変数を評価するを参照してください。

#### Java プログラムを 1 ステップずつ実行して、デバッグする:

デバッグしながらプログラムを 1 ステップずつ実行することができます。他の関数をステップオーバーしたり、ステップイントゥすることができます。Java プログラムおよびネイティブ・メソッドは、ステップ関数を使用することができます。

最初にプログラム・ソースが表示されると、ステップ実行を開始することができます。プログラムは、最初のステートメントを実行する前に停止します。F10 (ステップ) を押してください。プログラムを 1 ステップずつ実行するには、F10 (ステップ) を押し続けてください。プログラムが呼び出す関数をステップイントゥするには、F22 (ステップイン) を押してください。また、停止点に達した場合に、いつでもステップ実行を開始することができます。停止点を設定することについては、停止点の設定を参照してください。

#### モジュール・ソースの表示

```

現行スレッド: 00000019  停止スレッド: 00000019
クラス・ファイル名: Hellod
35  public static void main(String[] args)
36  {
37      int i,j,h,B[],D[] [];
38      Hellod A=new Hellod();
39      A.myHellod = A;
40      Hellod C[];
41      C = new Hellod[5];
42      for (int counter=0; counter<2; counter++) {
43          C[counter] = new Hellod();
44          C[counter].myHellod = C[counter];
45      }
46      C[2] = A;
47      C[0].myString = null;
48      C[0].myHellod = null;
49      A.method1();
デバッグ . . .

F3=終了プログラム  F6=停止点の追加/消去  F10=ステップ  F11=変数の表示

```



## Java およびネイティブ・メソッド・プログラムをデバッグする

Java プログラムとネイティブ・メソッド・プログラムを同時にデバッグできます。対話式画面でソースをデバッグする一方で、サービス・プログラム (\*SRVPGM) 内にある、C でプログラミングされたネイティブ・メソッドをデバッグできます。\*SRVPGM はデバッグ・データ付きでコンパイルおよび作成されている必要があります。

Java プログラムとネイティブ・メソッド・プログラム (またはサービス・プログラム) をデバッグする最も簡単な方法は、IBM iSeries System Debugger を使用することです。IBM iSeries System Debugger は、iSeries サーバーにグラフィカル・ユーザー・デバッグ環境を提供します。iSeries System Debugger を使用して iSeries サーバー上のプログラムをデバッグおよびテストする方法については、IBM iSeries System Debugger を参照してください。

サーバーの対話式画面を使用して、Java プログラムとネイティブ・メソッド・プログラムを同時にデバッグするには、以下の手順のようにします。

1. Java プログラム・ソースが表示されるときに F14 (モジュール・リストの処理) を押して、「モジュール・リストの処理 (WRKMODLST)」画面を表示する。
2. オプション 1 (プログラムの追加) を選択して、サービス・プログラムを追加する。
3. オプション 5 (モジュール・ソースの表示) を選択して、デバッグしたい \*MODULE とソースを表示する。
4. F6 (停止点の追加/消去) を押して、サービス・プログラムに停止点を設定する。停止点の設定の詳細については、停止点の設定を参照してください。
5. F12 (再開) を押してプログラムを実行する。

注: サービス・プログラム内の停止点に達すると、プログラムの実行が停止し、サービス・プログラムのソースが表示されます。

## 別の画面から Java プログラムをデバッグする

iSeries サーバー上で実行する Java プログラムをデバッグする最も簡単な方法は、IBM iSeries System Debugger を使用することです。IBM iSeries System Debugger は、iSeries サーバーのデバッグ機能をより使いやすくするグラフィカル・ユーザー・インターフェースを提供します。

iSeries System Debugger を使用して iSeries サーバー上で実行する Java プログラムをデバッグおよびテストする方法については、IBM iSeries System Debugger を参照してください。

サーバーの対話式画面を使用して Java プログラムをデバッグしているときは、停止点に達すると必ずプログラム・ソースが表示されます。これにより、Java プログラムの表示出力が妨げられることがあります。これを避けるには、別の画面から Java プログラムをデバッグします。Java プログラムの出力は Java コマンドが実行されている画面に表示され、プログラム・ソースは別の画面に表示されます。

この方法でのデバッグは、Just-In-Time (JIT) コンパイラーを使用していなければ、既に実行されている Java プログラムでも可能です。

別の画面から Java をデバッグするには、次のようにします。

1. デバッグの設定を開始するときには、Java プログラムを必ず保留にする。

プログラムで次のことを行くと、Java プログラムを保留できます。

- キーボードからの入力を待機する。
- 一定時間待機する。

- 変数をテストするためにループする。それには、Java プログラムのループを最終的に終了させるように値を設定しておく必要があります。
2. Java プログラムが保留されたら、別の画面に移って次のステップを実行する。
    - a. コマンド行に「活動ジョブの処理 (WRKACTJOB)」コマンドを入力する。
    - b. Java プログラムが実行されているバッチ即時 (BCI) ジョブを見つける。 QJVACMDSRV の「サブシステム/ジョブ」リストを調べる。使用している「ユーザー ID」の「ユーザー」リストを調べる。「タイプ」を調べて、BCI を探す。
    - c. オプション 5 を入力してそのジョブを処理する。
    - d. 「ジョブの処理」画面の上部に、「番号 (Number)」、「ユーザー (User)」、および「ジョブ (Job)」が表示される。 STRSRVJOB Number/User/Job と入力する。
    - e. STRDBG CLASS(classname) と入力する。 classname はデバッグしたい Java クラスの名前です。この名前は、Java コマンドで指定したクラス名でも、別のクラス名でもかまいません。
    - f. そのクラスのソースが「モジュール・ソースの表示」画面に表示される。
    - g. その Java クラス内でストップしたい位置で、F6 (停止点の追加/消去) を押して、停止点を設定する。デバッグする他のクラス、プログラム、サービス・プログラムを追加するには、F14 を押す。停止点の設定の詳細については、停止点の設定を参照してください。
    - h. F12 (再開) を押してプログラムの実行を続ける。
  3. 元の Java プログラムの保留を停止する。停止点に達すると、「モジュール・ソースの表示」画面が、「サービス・ジョブ開始 (STRSRVJOB)」コマンドと「デバッグ開始 (STRDBG)」コマンドが入力された画面に表示されます。Java プログラムが終了すると、Job being serviced ended (サービス対象のジョブが終了しました) というメッセージが表示されます。
  4. 「デバッグ・モード終了 (ENDDBG)」コマンドを入力する。
  5. 「サービス・ジョブ終了 (ENDSRVJOB)」コマンドを入力する。

注: Java 仮想マシンを開始するときには、元のジョブの中で Just-In-Time (JIT) が使用不可になっていることを確認してください。これは、java.compiler=NONE プロパティを使ってでも行えます。デバッグ中に JIT が実行されている場合、予期しない結果が起きることがあります。

Java 仮想マシンを呼び出すまで BCI ジョブが待機するかどうかを制御するこの変数の詳細については、QIBM\_CHILD\_JOB\_SNDINQMSG 環境変数を参照してください。

#### QIBM\_CHILD\_JOB\_SNDINQMSG 環境変数:

QIBM\_CHILD\_JOB\_SNDINQMSG 環境変数は、Java 仮想マシンが実行されるバッチ即時 (BCI) ジョブが、Java 仮想マシンが起動されるまで待機するかどうかを制御する変数です。

「Java プログラムの実行 (RUNJVA)」コマンドの実行時に環境変数を 1 に設定すると、メッセージがユーザーのメッセージ待ち行列に送られます。このメッセージは、BCI ジョブ内で Java 仮想マシンが開始される前に送られます。このメッセージは次のような形式です。

```
Spawned (child) process 023173/JOB/QJVACMDSRV is stopped (G C)
```

このメッセージを表示するには、SYSREQ と入力して、オプション 4 を選択します。

このメッセージに対する応答が入力されるまで BCI ジョブが待機します。(G) という応答で Java 仮想マシンが起動します。

メッセージに応答する前に、BCI ジョブが呼び出す \*SRVPGM または \*PGM で停止点を設定できます。



注: この時点では Java 仮想マシンが起動していないため、Java クラスに停止点を設定することはできません。

## カスタム・クラス・ローダーを通してロードされた Java クラスをデバッグする

iSeries サーバー上で実行する Java プログラムをデバッグする最も簡単な方法は、IBM iSeries System Debugger を使用することです。IBM iSeries System Debugger は、iSeries サーバーのデバッグ機能をより使いやすくするグラフィカル・ユーザー・インターフェースを提供します。

iSeries System Debugger を使用して iSeries サーバー上で実行する Java プログラムをデバッグおよびテストする方法については、IBM iSeries System Debugger を参照してください。

サーバーの対話式画面を使用して、カスタム・クラス・ローダーを通してロードされたクラスをデバッグするには、以下の手順のようにします。

1. ソース・コードが含まれているディレクトリー、またはパッケージ修飾クラスの場合はそのパッケージ名の開始ディレクトリーを `DEBUGSOURCEPATH` 環境変数に設定する。

たとえば、カスタム・クラス・ローダーが `/MYDIR` の下にあるクラスをロードする場合は、次のようにします。

```
ADDENVVAR ENVVAR(DEBUGSOURCEPATH) VALUE('/MYDIR')
```

2. 「モジュール・ソースの表示」画面からデバッグ・ビューにそのクラスを追加する。

そのクラスが既に Java 仮想マシン (JVM) にロードされている場合は、通常のように `*CLASS` を追加し、デバッグするソース・コードを表示します。

たとえば、`pkg1/test14/class` のソースを表示するには、次のように入力します。

| Opt | Program/module | Library | Type   |
|-----|----------------|---------|--------|
| 1   | pkg1.test14_   | *LIBL   | *CLASS |

クラスが JVM にロードされていない場合は、前述と同様の手順で `*CLASS` を追加してください。その結果、「**Java クラス・ファイルが使用できません。(Java class file not available)**」というメッセージが表示されます。ここで、プログラム処理を再開します。指定された名前と合致するクラスの任意のメソッドが入力されると、JVM は自動的に停止します。そのクラスのソース・コードが表示され、デバッグが可能になります。

## デバッグ・サブレット

デバッグ・サブレットは、カスタム・クラス・ローダーを通してロードされるデバッグ・クラスの特異なケースです。サブレットは、IBM HTTP サーバーの Java ランタイム上で実行されます。サブレットのデバッグには、いくつかの方法があります。

iSeries サーバー上で実行する Java プログラムおよびサブレットをデバッグする最も簡単な方法は、IBM iSeries System Debugger を使用することです。IBM iSeries System Debugger は、iSeries サーバーのデバッグ機能をより使いやすくするグラフィカル・ユーザー・インターフェースを提供します。

iSeries System Debugger を使用して iSeries サーバー上で実行する Java プログラムおよびサブレットをデバッグおよびテストする方法については、IBM iSeries System Debugger を参照してください。

サブレットをデバッグするもう 1 つの方法は、カスタム・クラス・ローダーを通してロードされた Java クラスをデバッグするにある方法に従うことです。



また、以下のステップを実行することにより、サーバーの対話式画面を使用してサーブレットをデバッグすることもできます。

1. Qshell インタープリターで `javac -g` コマンドを使用して、サーブレットをコンパイルします。
2. ソース・コード (.java ファイル) とコンパイル済みコード (.class ファイル) を `/QIBM/ProdData/Java400` にコピーします。
3. 最適化レベル 10, `OPTIMIZE(10)` を使用して、.class ファイルに対して Java プログラムの作成 (`CRTJVAPGM`) コマンドを実行します。
4. サーバーを開始します。
5. サーブレットを実行するジョブで「サービス・ジョブ開始 (`STRSRVJOB`)」コマンドを実行します。
6. `STRDBG CLASS(myServlet)` を実行します。myServlet はサーブレットの名前です。ソースが表示されるはずですが。
7. サーブレットに停止点を設定して `F12` を押します。
8. サーブレットを実行します。サーブレットが停止点に達しても、デバッグを継続できます。

## Java Platform Debugger Architecture

Java Platform Debugger Architecture (JPDA) は、JVM Debug Interface/JVM Tool Interface、Java Debug Wire Protocol、および Java Debug Interface で構成されます。JPDA のこれらの部分はすべて、デバッグ操作を実行するために JDWP を使用するデバッガーのフロントエンドを使用可能にします。デバッガー・フロントエンドは、リモート側で実行することもできますし、iSeries アプリケーションとして実行することもできます。

### | Java Virtual Machine Tool Interface (JVMTI)

- | JVMTI は、JVMDI および Java Virtual Machine Profiler Interface (JVMPPI) に置き換わるものです。
- | JVMTI には、JVMDI と JVMPPI の両方のすべての機能、および新機能が備えられています。JVMTI は J2SE 5.0 の一部として追加されました。将来のリリースでは、JVMDI インターフェースおよび JVMPPI インターフェースの提供は打ち切れ、JVMTI だけしか選択できなくなります。
- | QSYS ライブラリーに存在する QJVAJVMTI というサービス・プログラムは、JVMTI 機能をサポートします。
- | JVMTI のインプリメントの詳細については、Sun Microsystems, Inc. の Web サイトの JVMTI リファレンス・ページ (英語) を参照してください。

## Java Virtual Machine Debug Interface

Java 2 SDK (J2SDK), Standard Edition バージョン 1.2 以降において、Java Virtual Machine Debug Interface (JVMDI) は、Sun Microsystems, Inc. のプラットフォーム・アプリケーション・プログラム・インターフェース (API) の一部です。JVMDI を使用すると、誰でも iSeries C コードで iSeries サーバー用の Java デバッガーを作成することができます。デバッガーでは、JVMDI インターフェースを使用するため、Java 仮想マシンの内部構造を認識する必要はありません。JVMDI は、Java 仮想マシンに最も近い、JPDA の最低レベルのインターフェースです。

デバッガーは、Java 仮想マシンと同じマルチスレッド対応ジョブで実行されます。デバッガーは、Java ネイティブ・インターフェース (JNI) の呼び出し API を使用して、Java 仮想マシンを作成します。その後、ユーザー・クラスの main メソッドの先頭にフックを置き、main メソッドを呼び出します。main メソッドが開始されると、フックがヒットされ、デバッグが開始されます。停止点の設定、ステップ実行、変数の表示、および変数の変更など、一般的なデバッグ機能が使用可能です。

デバッガーは、Java 仮想マシンが実行されているジョブと、ユーザー・インターフェースを処理するジョブの間の通信を処理します。このユーザー・インターフェースは、iSeries サーバーまたは別のシステムにあります。

QSYS ライブラリーに存在する QJVAJVMDI というサービス・プログラムは、JVMDI 機能をサポートします。

## Java Debug Wire Protocol

Java Debug Wire Protocol (JDWP) は、デバッガー・プロセスと JVMDI/JVMTI の間の定義済み通信プロトコルです。JDWP はリモート・システムから使用することもできますし、ローカル・ソケットを介して使用することもできます。これは、JVMDI/JVMTI から取り外された 1 つの階層ですが、より複雑なインターフェースです。

## JDWP を QShell で開始する

- | JDWP を開始して Java クラス SomeClass を実行するには、QShell で次のコマンドを入力してください。

```
java -interpret -agentlib:jdwp=transport=dt_socket,  
address=8000,server=y,suspend=n SomeClass
```

この例では、JDWP は TCP/IP ポート 8000 上でリモート・デバッガーからの接続を listen しますが、任意のポート番号を使用できます。dt\_socket は JDWP トランスポートを処理する SRVPGM の名前が変わることはできません。

- | -Xrunjdwp で使用できる追加のオプションについては、Sun Microsystems, Inc. の Sun VM Invocation Options を参照してください。これらのオプションは、i5/OS 上の JDK 1.4 および 1.5 の両方で選択可能です。

## JDWP を CL コマンド行から開始する

- | JDWP を CL コマンドで開始するために、AGTPGM および AGTOPTIONS の 2 つの新規オプションが追加されています。

- | AGTPGM の値は JDWP であり、AGTOPTIONS の値は QShell コマンド行で使用するのと同じストリングになるように定義できます。

- | JDWP を開始して Java クラス SomeClass を実行するには、次のコマンドを入力してください。

```
JAVA CLASS(SomeClass) INTERPRET(*YES) AGTPGM(JDWP)  
AGTOPTIONS('transport=dt_socket,address=8000,server=y,suspend=n')
```

- | 直接実行コードに対して JVMDI/JVMTI を使用することはお勧めしません。アプリケーションをインタープリターを使用して実行するか、あるいは、Just-In-Time (JIT) コンパイラーとフルスピード・デバッグを使用する必要があります。

## Java Debug Interface

Java Debug Interface (JDI) は、ツール開発用に用意されているハイレベル Java 言語インターフェースです。JDI では、Java クラス定義を使用することで、JVMDI/JVMTI および JDWP の複雑さが隠されています。JDI は rt.jar ファイルに入っているため、デバッガーのフロントエンドは、Java がインストールされているすべてのプラットフォーム上に存在することになります。

Java 用のデバッガーを作成する場合、JDI は最も単純なインターフェースであり、コードはプラットフォームに依存していないので、JDI を使用してください。

JDK の詳細については、Sun Microsystems, Inc. の [Java Platform Debugger Architecture Overview](#) を参照してください。

### フルスピード・デバッグ:

iSeries Java Virtual Machine (JVM) は現在「フルスピード・デバッグ」をサポートしています。V5R3 より前は、デバッグを使用可能にすると、Just-In-Time (JIT) コンパイラーが使用できなくなりました。多くのメソッドは、低速のインタープリターを使用して実行しなければならなかったため、アプリケーション・パフォーマンスが低下することになりました。この大きなパフォーマンスの低下は、デバッグを開始したいポイントに到達するまでに何日もかかることがあるアプリケーションにとって特に問題でした。

フルスピード・デバッグでは、停止点の設定、コードのステップ実行、およびローカル変数の表示などといった、最も一般的なデバッグ・アクティビティーを実行する機能を失うことなく、JIT コンパイル・コードのすべてのパフォーマンス上の利点を活用してアプリケーションを実行することができます。

フルスピード・デバッグでは、メソッドを JIT でコンパイルできるので、デバッグに関して 2 つの制限があります。

- 呼び出し元がコンパイル済みコードである場合、戻りステートメントでのステップ実行操作が機能しません。
- 監視ポイントは、監視対象フィールドを変更する非コンパイル・メソッドでのみトリガーします。

**注:** このフィーチャーは、Java Debug Wire Protocol (JDWP) を使用してデバッグ操作を実行するデバッガーでのみサポートされます。システム・デバッガーは現在はフルスピード・デバッグをサポートしていません。

### メモリー・リークを検出する

プログラムの実行時間が長くなるとパフォーマンスが低下する場合は、誤ってメモリー・リークがコーディングされている可能性があります。Java Watcher を使用すれば、時間を追って Java アプリケーション・ヒープ分析とオブジェクト作成プロファイルの作成を行うことにより、プログラムをデバッグして、メモリー・リークを検出することができます。

詳しくは、JavaWatcher を参照してください。

「Java 仮想マシンの分析 (ANZJVM)」制御言語 (CL) コマンドを使用してオブジェクト・リークを検出することもできます。ANZJVM は、指定された時間間隔でガーベッジ・コレクション・ヒープのコピーを取ることで、オブジェクト・リークを検出します。オブジェクト・リークを検出するには、ヒープ内のそれぞれのクラスのインスタンスの数を確認します。インスタンス数が異常に多くなっているクラスがあれば、そのクラスでリークが発生している可能性があります。

また、ガーベッジ・コレクション・ヒープの 2 つのコピーの間での、各クラスのインスタンスの数の変化にも注意します。あるクラスのインスタンス数が継続して増加し続けている場合は、そのクラスでリークが発生している可能性があります。2 つのコピーの時間間隔を広げていくと、そのオブジェクトが実際にリークを発生させている可能性が高くなります。ANZJVM を一連の回数で長い時間間隔で実行させると、どこでリークが発生しているか、高い確度で診断することが可能になります。

---

## IBM Developer Kit for Java のコード例

以下に、IBM Developer Kit for Java のコード例の一覧を示します。

### 国際化対応

- DateFormat
- NumberFormat
- ResourceBundle

### JDBC

- Access プロパティ
- Blob
- CallableStatement インターフェース
- 他のステートメントのカーソルを介してステートメントで値を変更する
- Clob
- UDBDataSource を作成し、JNDI を使用してバインドする
- UDBDataSource を作成し、ユーザー ID とパスワードを取得する
- UDBDataSourceBind を作成し、DataSource プロパティを設定する
- DatabaseMetaData インターフェース
- UDBDataSource を作成し、JNDI を使用してバインドする
- Datalink
- 特殊タイプ
- 組み込み SQL ステートメント
- トランザクションを終了する
- 無効なユーザー ID とパスワード
- JDBC
- 単一のトランザクション上で動作する複数の接続
- UDBDataSource をバインディングする前に初期コンテキストを取得する
- ParameterMetaData
- 他のステートメントのカーソルを介してテーブルからデータを除去する
- ResultSet インターフェース
- ResultSet の感度
- 感知および非感知の ResultSet
- UDBDataSource および UDBConnectionPoolDataSource で接続プーリングをセットアップする
- SQLException
- トランザクションを中断および再開する
- 中断された ResultSet
- 接続プーリングのパフォーマンスをテストする
- 2 つの DataSource のパフォーマンスをテストする
- BLOB を更新する
- CLOB を更新する

- 複数のトランザクションで単一の接続を使用する
- BLOB を使用する
- CLOB を使用する
- DB2CachedRowSet プロパティと DataSource を使用する
- DB2CachedRowSet プロパティと JDBC URL を使用する
- トランザクションを処理するために JTA を使用する
- 複数の列を持ったメタデータ ResultSet を使用する
- ネイティブ JDBC と IBM Toolbox for Java JDBC を同時に使用する
- ResultSet を取得するために PreparedStatement を使用する
- execute(Connection) メソッドを使って、既存のデータベース接続を使用する
- データベース要求をまとめるため、execute(int) メソッドを使用する
- populate メソッドを使用する
- setConnection(Connection) メソッドを使って、既存のデータベース接続を使用する
- Statement オブジェクトの executeUpdate メソッドを使用する

#### **Java Authentication and Authorization Service**

- JAAS HelloWorld サンプル
- JAAS SampleThreadSubjectLogin サンプル

#### **Java Generic Security Service**

- 非 JAAS クライアントのサンプル・プログラム
- 非 JAAS サーバーのサンプル・プログラム
- JAAS 対応クライアントのサンプル・プログラム
- JAAS 使用可能サーバー・プログラムのサンプル

#### **Java セキュア・ソケット拡張機能**

- SSLContext オブジェクトを使用した SSL クライアントおよびサーバー

#### **Java と他のプログラム言語**

- CL プログラムを呼び出す
- CL コマンドを呼び出す
- 他の Java プログラムを呼び出す
- C から Java を呼び出す
- RPG から Java を呼び出す
- 入出力ストリーム
- 呼び出し API
- Java 用の i5/OS PASE ネイティブ・メソッド
- ソケット
- ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用する

#### **パフォーマンス測定ツール**

- Java パフォーマンス・データ・コンバーター

## SQLJ

- SQL ステートメントを Java アプリケーションに組み込む

## Secure Sockets Layer

- ソケット・ファクトリー
- サーバー・ソケット・ファクトリー
- Secure Sockets Layer
- Secure Sockets Layer サーバー

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

1 強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとしします。

1 IBM、そのプログラム開発者、または供給者は、いかなる場合においてもその予見の有無を問わず、以下に対する責任を負いません。

1. データの喪失、または損傷。
1. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
1. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用

1 国または地域によっては、法律の強行規定により、上記の責任の制限が適用されない場合があります。

## 例: java.util.DateFormat クラスを使用して日付を国際化する

この例では、ロケールを使用して日付を形式化する方法を示します。

### 例 1: 日付を国際化するための java.util.DateFormat クラスの使用

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとしします。

```
//*****  
// File: DateExample.java  
//*****  
  
import java.text.*;  
import java.util.*;  
import java.util.Date;  
  
public class DateExample {  
  
    public static void main(String args[]) {  
  
        // Get the Date  
        Date now = new Date();  
  
        // Get date formatters for default, German, and French locales  
        DateFormat theDate = DateFormat.getDateInstance(DateFormat.LONG);  
        DateFormat germanDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.GERMANY);  
        DateFormat frenchDate = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);  
  
        // Format and print the dates
```



```

        System.out.println("Date in the default locale: " + theDate.format(now));
        System.out.println("Date in the German locale : " + germanDate.format(now));
        System.out.println("Date in the French locale : " + frenchDate.format(now));
    }
}

```

詳細については、国際化 Java プログラムを作成するを参照してください。

### リンク集

コードのサンプルに関する特記事項

国際化 Java(TM) プログラムを作成するを参照してください。

特定の地域に Java プログラムをカスタマイズする必要がある場合は、Java ロケールを使用して、国際化 Java プログラムを作成できます。

## 例: java.util.NumberFormat クラスを使用して数値表示を国際化する

この例では、ロケールを使用して数値を形式化する方法を示します。

**例 1:** 数値出力を国際化するための java.util.NumberFormat クラスの使用

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

//*****
// File: NumberExample.java
//*****

import java.lang.*;
import java.text.*;
import java.util.*;

public class NumberExample {

    public static void main(String args[]) throws NumberFormatException {

        // The number to format
        double number = 12345.678;

        // Get formatters for default, Spanish, and Japanese locales
        NumberFormat defaultFormat = NumberFormat.getInstance();
        NumberFormat spanishFormat = NumberFormat.getInstance(new
Locale("es", "ES"));
        NumberFormat japaneseFormat = NumberFormat.getInstance(Locale.JAPAN);

        // Print out number in the default, Spanish, and Japanese formats
        // (Note: NumberFormat is not necessary for the default format)
        System.out.println("The number formatted for the default locale; " +
            defaultFormat.format(number));
        System.out.println("The number formatted for the Spanish locale; " +
            spanishFormat.format(number));
        System.out.println("The number formatted for the Japanese locale; " +
            japaneseFormat.format(number));
    }
}

```

詳細については、国際化 Java プログラムを作成するを参照してください。

### リンク集

コードのサンプルに関する特記事項

国際化 Java プログラムを作成する

特定の地域に Java プログラムをカスタマイズする必要がある場合は、Java ロケールを使用して、国際化 Java プログラムを作成できます。

## 例: java.util.ResourceBundle クラスを使用してロケール固有データを国際化する

この例では、リソース・バンドルとともにロケールを使用して、プログラム・ストリングを国際化する方法を示します。

ResourceBundleExample プログラムが意図されたとおりに機能するためには、以下のプロパティ・ファイルが必要です。

### RBExample.properties の内容

```
Hello.text=Hello
```

### RBExample\_de.properties の内容

```
Hello.text=Guten Tag
```

### RBExample\_fr\_FR.properties の内容

```
Hello.text=Bonjour
```

**例 1:** ロケール固有データを国際化するための java.util.ResourceBundle クラスの使用

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
//*****  
// File: ResourceBundleExample.java  
//*****  
  
import java.util.*;  
  
public class ResourceBundleExample {  
    public static void main(String args[]) throws MissingResourceException {  
  
        String resourceName = "RBExample";  
        ResourceBundle rb;  
  
        // Default locale  
        rb = ResourceBundle.getBundle(resourceName);  
        System.out.println("Default : " + rb.getString("Hello" + ".text"));  
  
        // Request a resource bundle with explicitly specified locale  
        rb = ResourceBundle.getBundle(resourceName, Locale.GERMANY);  
        System.out.println("German : " + rb.getString("Hello" + ".text"));  
  
        // No property file for China in this example... use default  
        rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);  
        System.out.println("Chinese : " + rb.getString("Hello" + ".text"));  
  
        // Here is another way to do it...  
        Locale.setDefault(Locale.FRANCE);  
        rb = ResourceBundle.getBundle(resourceName);  
        System.out.println("French : " + rb.getString("Hello" + ".text"));  
  
        // No property file for China in this example... use default, which is now fr_FR.  
        rb = ResourceBundle.getBundle(resourceName, Locale.CHINA);  
        System.out.println("Chinese : " + rb.getString("Hello" + ".text"));  
    }  
}
```

詳細については、国際化 Java<sup>TM</sup> プログラムを作成するを参照してください。

## リンク集

コードのサンプルに関する特記事項

国際化 Java(TM) プログラムを作成するを参照してください。

特定の地域に Java プログラムをカスタマイズする必要がある場合は、Java ロケールを使用して、国際化 Java プログラムを作成できます。

## 例: Access プロパティ

以下に、Access プロパティの使用法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// Note: This program assumes directory cujosql exists.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class AccessPropertyTest {
    public String url = "jdbc:db2:*local";
    public Connection connection = null;

    public static void main(java.lang.String[] args)
        throws Exception
    {
        AccessPropertyTest test = new AccessPropertyTest();

        test.setup();

        test.run();
        test.cleanup();
    }

    /**
     * Set up the DataSource used in the testing.
     */
    public void setup()
        throws Exception
    {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        connection = DriverManager.getConnection(url);
        Statement s = connection.createStatement();
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.TEMP");
        } catch (SQLException e) { // Ignore it - it doesn't exist
        }

        try {
            String sql = "CREATE PROCEDURE CUJOSQL.TEMP "
                + " LANGUAGE SQL SPECIFIC CUJOSQL.TEMP "
                + " MYPROC: BEGIN"
                + "     RETURN 11;"
                + " END MYPROC";
            s.executeUpdate(sql);
        } catch (SQLException e) {
            // Ignore it - it exists.
        }
        s.executeUpdate("create table cujosql.temp (col1 char(10))");
        s.executeUpdate("insert into cujosql.temp values ('compare')");
        s.close();
    }
}
```

```

}

public void resetConnection(String property)
throws SQLException
{
    if (connection != null)
        connection.close();

    connection = DriverManager.getConnection(url + ";access=" + property);
}

public boolean canQuery() {
    Statement s = null;
    try {
        s = connection.createStatement();
        ResultSet rs = s.executeQuery("SELECT * FROM cujosql.temp");
        if (rs == null)
            return false;

        rs.next();

        if (rs.getString(1).equals("compare  "))
            return true;

        return false;
    } catch (SQLException e) {
        // System.out.println("Exception: SQLState(" +
        //                      e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}

public boolean canUpdate() {
    Statement s = null;
    try {
        s = connection.createStatement();
        int count = s.executeUpdate("INSERT INTO CUJOSQL.TEMP VALUES('x')");
        if (count != 1)
            return false;

        return true;
    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}

```

```

}

public boolean canCall() {
    CallableStatement s = null;
    try {
        s = connection.prepareCall("? = CALL CUJOSQL.TEMP()");
        s.registerOutParameter(1, Types.INTEGER);
        s.execute();
        if (s.getInt(1) != 11)
            return false;

        return true;
    } catch (SQLException e) {
        //System.out.println("Exception: SQLState(" +
        //                    e.getSQLState() + ") " + e + " (" + e.getErrorCode() + ")");
        return false;
    } finally {
        if (s != null) {
            try {
                s.close();
            } catch (Exception e) {
                // Ignore it.
            }
        }
    }
}

public void run()
throws SQLException
{
    System.out.println("Set the connection access property to read only");
    resetConnection("read only");

    System.out.println("Can run queries -->" + canQuery());
    System.out.println("Can run updates -->" + canUpdate());
    System.out.println("Can run sp calls -->" + canCall());

    System.out.println("Set the connection access property to read call");
    resetConnection("read call");

    System.out.println("Can run queries -->" + canQuery());
    System.out.println("Can run updates -->" + canUpdate());
    System.out.println("Can run sp calls -->" + canCall());

    System.out.println("Set the connection access property to all");
    resetConnection("all");

    System.out.println("Can run queries -->" + canQuery());
    System.out.println("Can run updates -->" + canUpdate());
    System.out.println("Can run sp calls -->" + canCall());
}

public void cleanup() {
    try {
        connection.close();
    } catch (Exception e) {
        // Ignore it.
    }
}
}

```

## 例: BLOB

以下は、 BLOB をデータベースに書き込んだり、データベースから検索したりする方法の例です。

注: サンプル・コードをご使用の場合は、 560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
// PutGetBlobs is an example application
// that shows how to work with the JDBC
// API to obtain and put BLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two BLOB values
// in a new table. Both are identical
// and contain 500k of random byte
// data.
////////////////////////////////////
import java.sql.*;
import java.util.Random;

public class PutGetBlobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any previous run of this application.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.BLOBTABLE");
        } catch (SQLException e) {
            // Ignore it - assume the table did not exist.
        }

        // Create a table with a BLOB column. The default BLOB column
        // size is 1 MB.
        s.executeUpdate("CREATE TABLE CUJOSQL.BLOBTABLE (COL1 BLOB)");

        // Create a PreparedStatement object that allows you to put
        // a new Blob object into the database.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.BLOBTABLE VALUES(?)");

        // Create a big BLOB value...
        Random random = new Random ();
        byte [] inByteArray = new byte[500000];
        random.nextBytes (inByteArray);

        // Set the PreparedStatement parameter. Note: This is not
        // portable to all JDBC drivers. JDBC drivers do not have
        // support when using setBytes for BLOB columns. This is used to
        // allow you to generate new BLOBs. It also allows JDBC 1.0
        // drivers to work with columns containing BLOB data.
        ps.setBytes(1, inByteArray);

        // Process the statement, inserting the BLOB into the database.
        ps.executeUpdate();
    }
}
```



```

// Process a query and obtain the BLOB that was just inserted out
// of the database as a Blob object.
ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");
rs.next();
Blob blob = rs.getBlob(1);

// Put that Blob back into the database through
// the PreparedStatement.
ps.setBlob(1, blob);
ps.execute();

c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: IBM Developer Kit for Java 用の CallableStatement インターフェース

次に、CallableStatement インターフェースの使用法の例を示します。

### 例: CallableStatement インターフェース

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

// Connect to iSeries server.
Connection c = DriverManager.getConnection("jdbc:db2://mySystem");

// Create the CallableStatement object.
// It precompiles the specified call to a stored procedure.
// The question marks indicate where input parameters must be set and
// where output parameters can be retrieved.
// The first two parameters are input parameters, and the third parameter is an output parameter.
CallableStatement cs = c.prepareCall("CALL MYLIBRARY.ADD (?, ?, ?)");

// Set input parameters.
cs.setInt (1, 123);
cs.setInt (2, 234);

// Register the type of the output parameter.
cs.registerOutParameter (3, Types.INTEGER);

// Run the stored procedure.
cs.execute ();

// Get the value of the output parameter.
int sum = cs.getInt (3);

// Close the CallableStatement and the Connection.
cs.close();
c.close();

```

詳しくは、CallableStatementsを参照してください。

### リンク集

コードのサンプルに関する特記事項

### CallableStatement

CallableStatement インターフェースは PreparedStatement を拡張し、パラメーターの出力および入出力のサポートを提供します。 CallableStatement インターフェースは、PreparedStatement インターフェースによって提供される入力パラメーターもサポートします。

## 例: 他のステートメントのカーソルを介してテーブルから値を除去する

以下は、他のステートメントのカーソルを介してテーブルから値を除去する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;

public class UsingPositionedDelete {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {
        UsingPositionedDelete test = new UsingPositionedDelete();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }

    /**
    Handle all the required setup work.
    */
    public void setup() {
        try {
            // Register the JDBC driver.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignore problems here.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
            e.printStackTrace();
        }
    }

    /**
    In this section, all the code to perform the testing should
    be added. If only one connection to the database is needed,
    the global variable 'connection' can be used.
    */
    public void run() {
        try {
            Statement stmt1 = connection.createStatement();
```

```

// Update each value using next().
stmt1.setCursorName("CUJO");
ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
                                   "FOR UPDATE OF COL_VALUE");

System.out.println("Cursor name is " + rs.getCursorName());

PreparedStatement stmt2 = connection.prepareStatement
    ("DELETE FROM " + " CUJOSQL.WHERECUREX WHERE CURRENT OF " +
     rs.getCursorName ());

// Loop through the ResultSet and update every other entry.
while (rs.next ()) {
    if (rs.next())
        stmt2.execute ();
}

// Clean up the resources after they have been used.
rs.close ();
stmt2.close ();

} catch (Exception e) {
    System.out.println("Caught exception: ");
    e.printStackTrace();
}
}

/**
In this section, put all clean-up work for testing.
**/
public void cleanup() {
    try {
        // Close the global connection opened in setup().
        connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}

/**
Display the contents of the table.
**/
public void displayTable()
{
    try {
        Statement s = connection.createStatement();
        ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

        while (rs.next ()) {
            System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
        }

        rs.close ();
        s.close();
        System.out.println("-----");
    } catch (Exception e) {
        System.out.println("Caught exception: ");
    }
}

```

```

        e.printStackTrace();
    }
}

```

## リンク集

コードのサンプルに関する特記事項

## 例: CLOB

以下は、CLOB をデータベースに書き込んだり、データベースから検索したりする方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// PutGetClobs is an example application
// that shows how to work with the JDBC
// API to obtain and put CLOBs to and from
// database columns.
//
// The results of running this program
// are that there are two CLOB values
// in a new table. Both are identical
// and contain about 500k of repeating
// text data.
////////////////////////////////////
import java.sql.*;

public class PutGetClobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any previous run of this application.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.CLOBTABLE");
        } catch (SQLException e) {
            // Ignore it - assume the table did not exist.
        }

        // Create a table with a CLOB column. The default CLOB column
        // size is 1 MB.
        s.executeUpdate("CREATE TABLE CUJOSQL.CLOBTABLE (COL1 CLOB)");

        // Create a PreparedStatement object that allow you to put
        // a new Clob object into the database.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.CLOBTABLE VALUES(?)");

        // Create a big CLOB value...
        StringBuffer buffer = new StringBuffer(500000);
        while (buffer.length() < 500000) {
            buffer.append("All work and no play makes Cujo a dull boy.");
        }
        String clobValue = buffer.toString();
    }
}

```

```

// Set the PreparedStatement parameter. This is not
// portable to all JDBC drivers. JDBC drivers do not have
// to support setBytes for CLOB columns. This is done to
// allow you to generate new CLOBs. It also
// allows JDBC 1.0 drivers a way to work with columns containing
// Clob data.
ps.setString(1, clobValue);

// Process the statement, inserting the clob into the database.
ps.executeUpdate();

// Process a query and get the CLOB that was just inserted out of the
// database as a Clob object.
ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");
rs.next();
Clob clob = rs.getClob(1);

// Put that Clob back into the database through
// the PreparedStatement.
ps.setClob(1, clob);
ps.execute();

c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: UDBDataSource を作成して JNDI でバインドする

次に、UDBDataSource を作成し、それを JNDI でバインドする方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS", ds);
    }
}

```

## 例: UDBDataSource の作成、およびユーザー ID とパスワードの取得

以下は、UDBDataSource を作成し、実行時に getConnection メソッドを使用してユーザー ID とパスワードを取得する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
/// Import the required packages. There is
// no driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse2
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know
        // what the implementation class is. The logical JNDI name
        // is only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");

        // Once the DataSource is obtained, it can be used to establish
        // a connection. The user profile cujo and password newtiger
        // used to create the connection instead of any default user
        // ID and password for the DataSource.
        Connection connection = ds.getConnection("cujo", "newtiger");

        // The connection can be used to create Statement objects and
        // update the database or process queries as follows.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }

        // The connection is closed before the application ends.
        connection.close();
    }
}
```

## 例: UDBDataSourceBind を作成して DataSource プロパティを設定する

次に、UDBDataSource を作成し、DataSource のプロパティとしてユーザー ID とパスワードを設定する方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。



```

// Import the required packages. At deployment time,
// the JDBC driver-specific class that implements
// DataSource must be imported.
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;

public class UDBDataSourceBind2
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Create a new UDBDataSource object and give it
        // a description.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("A simple UDBDataSource " +
            "with cujo as the default " +
            "profile to connect with.");

        // Provide a user ID and password to be used for
        // connection requests.
        ds.setUser("cujo");
        ds.setPassword("newtiger");

        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Bind the newly created UDBDataSource object
        // to the JNDI directory service, giving it a name
        // that can be used to look up this object again
        // at a later time.
        ctx.rebind("SimpleDS2", ds);
    }
}

```

## 例: IBM Developer Kit for Java 用の DatabaseMetaData インターフェース

次の例は、テーブルのリストを戻す方法を示しています。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

// Connect to iSeries server.
Connection c = DriverManager.getConnection("jdbc:db2:mySystem");

// Get the database meta data from the connection.
DatabaseMetaData dbMeta = c.getMetaData();

// Get a list of tables matching this criteria.
String catalog = "myCatalog";
String schema = "mySchema";
String table = "myTable%"; // % indicates search pattern
String types[] = {"TABLE", "VIEW", "SYSTEM TABLE"};
ResultSet rs = dbMeta.getTables(catalog, schema, table, types);

// ... iterate through the ResultSet to get the values.

// Close the connection.
c.close();

```

詳細については、IBM Developer Kit for Java 用の DatabaseMetaData インターフェースを参照してください。

## 例: Datalink

以下は、アプリケーションでの Datalink の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
// PutGetDatalinks is an example application
// that shows how to use the JDBC
// API to handle datalink database columns.
////////////////////////////////////
import java.sql.*;
import java.net.URL;
import java.net.MalformedURLException;

public class PutGetDatalinks {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        // Establish a Connection and Statement with which to work.
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any previous run of this application.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.DLTABLE");
        } catch (SQLException e) {
            // Ignore it - assume the table did not exist.
        }

        // Create a table with a datalink column.
        s.executeUpdate("CREATE TABLE CUJOSQL.DLTABLE (COL1 DATALINK)");

        // Create a PreparedStatement object that allows you to add
        // a new datalink into the database. Since conversing
        // to a datalink cannot be accomplished directly in the database, you
        // can code the SQL statement to perform the explicit conversion.
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.DLTABLE
            VALUES(DLVALUE( CAST(? AS VARCHAR(100))))");

        // Set the datalink. This URL points you to an article about
        // the new features of JDBC 3.0.
        ps.setString (1, "http://www-106.ibm.com/developerworks/java/library/j-jdbcnew/index.html");

        // Process the statement, inserting the CLOB into the database.
        ps.executeUpdate();

        // Process a query and obtain the CLOB that was just inserted out of the
        // database as a Clob object.
        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
        rs.next();
        String datalink = rs.getString(1);

        // Put that datalink value into the database through
```

```

// the PreparedStatement. Note: This function requires JDBC 3.0
// support.
/*
try {
    URL url = new URL(dataLink);
    ps.setURL(1, url);
    ps.execute();
} catch (MalformedURLException mue) {
    // Handle this issue here.
}

rs = s.executeQuery("SELECT * FROM CUJOSQL.DLTABLE");
rs.next();
URL url = rs.getURL(1);
System.out.println("URL value is " + url);
*/

c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: 特殊タイプ

以下に、特殊タイプの使用法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// This example program shows examples of
// various common tasks that can be done
// with distinct types.
////////////////////////////////////
import java.sql.*;

public class Distinct {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        // Clean up any old runs.
        try {
            s.executeUpdate("DROP TABLE CUJOSQL.SERIALNOS");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        try {
            s.executeUpdate("DROP DISTINCT TYPE CUJOSQL.SSN");
        } catch (SQLException e) {
            // Ignore it and assume the table did not exist.
        }

        // Create the type, create the table, and insert a value.
        s.executeUpdate("CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)");
        s.executeUpdate("CREATE TABLE CUJOSQL.SERIALNOS (COL1 CUJOSQL.SSN)");
    }
}

```

```

PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.SERIALNOS VALUES(?)");
ps.setString(1, "399924563");
ps.executeUpdate();
ps.close();

// You can obtain details about the types available with new metadata in
// JDBC 2.0
DatabaseMetaData dmd = c.getMetaData();

int types[] = new int[1];
types[0] = java.sql.Types.DISTINCT;

ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN", types);
rs.next();
System.out.println("Type name " + rs.getString(3) +
    " has type " + rs.getString(4));

// Access the data you have inserted.
rs = s.executeQuery("SELECT COL1 FROM CUJOSQL.SERIALNOS");
rs.next();
System.out.println("The SSN is " + rs.getString(1));

c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: SQL ステートメントを Java アプリケーションに組み込む

以下の SQLJ アプリケーション例、App.sqlj は、静的 SQL を使用して更新データを DB2 サンプル・データベースの EMPLOYEE テーブルから検索します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // 1
#sql iterator App_Cursor2 (String) ;

class App
{
    /**
     * Register Driver **
     */

    static
    {
        try
        {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    /**
     * Main **
     */

    public static void main(String argv[])

```

```

{
  try
  {
    App_Cursor1 cursor1;
    App_Cursor2 cursor2;

    String str1 = null;
    String str2 = null;
    long count1;

    // URL is jdbc:db2:dbname
    String url = "jdbc:db2:sample";

    DefaultContext ctx = DefaultContext.getDefaultContext();
    if (ctx == null)
    {
      try
      {
        // connect with default id/password
        Connection con = DriverManager.getConnection(url);
        con.setAutoCommit(false);
        ctx = new DefaultContext(con);
      }
      catch (SQLException e)
      {
        System.out.println("Error: could not get a default context");
        System.err.println(e);
        System.exit(1);
      }
      DefaultContext.setDefaultContext(ctx);
    }

    // retrieve data from the database
    System.out.println("Retrieve some data from the database.");
    #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2

    // display the result set
    // cursor1.next() returns false when there are no more rows
    System.out.println("Received results:");
    while (cursor1.next()) // 3
    {
      str1 = cursor1.empno(); // 4
      str2 = cursor1.firstnme();

      System.out.print (" empno= " + str1);
      System.out.print (" firstnme= " + str2);
      System.out.println("");
    }
    cursor1.close(); // 9

    // retrieve number of employee from the database
    #sql { SELECT count(*) into :count1 FROM employee }; // 5
    if (1 == count1)
      System.out.println ("There is 1 row in employee table");
    else
      System.out.println ("There are " + count1
        + " rows in employee table");

    // update the database
    System.out.println("Update the database.");
    #sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

    // retrieve the updated data from the database
    System.out.println("Retrieve the updated data from the database.");
    str1 = "000010";
    #sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6
  }
}

```

```

// display the result set
// cursor2.next() returns false when there are no more rows
System.out.println("Received results:");
while (true)
{
    #sql { FETCH :cursor2 INTO :str2 }; // 7
    if (cursor2.endFetch()) break; // 8

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
}
cursor2.close(); // 9

// rollback the update
System.out.println("Rollback the update.");
#sql { ROLLBACK work };
System.out.println("Rollback done.");
}
catch( Exception e )
{
    e.printStackTrace();
}
}
}

```

1 反復子を宣言する。このセクションでは、次の 2 種類の反復子を宣言します。

- App\_Cursor1: 列データのタイプおよび名前を宣言して、列名 (列に結び付けられた名前) に応じた列の値を戻します。
- App\_Cursor2: 列データのタイプを宣言して、列位置 (列に結び付けられた定位置) に応じた列の値を戻します。

2 反復子を初期設定する。反復子オブジェクト `cursor1` が照会の結果を使用して初期設定されます。照会は結果を `cursor1` に格納します。

3 反復子を次の行に進める。 `cursor1.next()` メソッドは、検索する行がなくなった場合にブール値の偽を戻します。

4 データを移動する。名前付きアクセス機構メソッド `empno()` は、現在の行にある `empno` という名前の列の値を戻します。名前付きアクセス機構メソッド `firstnme()` は、現在の行にある `firstnme()` という名前の列の値を戻します。

5 データをホスト変数に SELECT する。SELECT ステートメントは、テーブル内の行数をホスト変数 `count1` に渡します。

6 反復子を初期設定する。反復子オブジェクト `cursor2` が照会の結果を使用して初期設定されます。照会は結果を `cursor2` に格納します。

7 データを検索する。FETCH ステートメントは、結果テーブルから ByPos カーソル内で宣言された最初の列の現行値を、ホスト変数 `str2` に戻します。

8 FETCH.INTO ステートメントが成功したかを検査する。 `endFetch()` メソッドは、反復子が行に位置していない場合、つまり行を取り出す前回の試行が失敗した場合に、ブール値の真を戻します。 `endFetch()` メソッドは、行を取り出す前回の試行が成功した場合に、偽を戻します。DB2 は `next()` メソッドが呼び出されたときに行の取り出しを試行します。FETCH...INTO ステートメントは、暗黙的に `next()` メソッドを呼び出します。



<sup>9</sup> 反復子をクローズする。close() メソッドは、反復子が保持しているリソースを解放します。反復子を明示的にクローズして、システム・リソースが適時に解放されるようにしてください。

この例に関する背景情報は、SQL ステートメントを Java アプリケーションに組み込むを参照してください。

## 例: トランザクションを終了する

以下は、アプリケーション内でトランザクションを終了する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEnd {

    public static void main(java.lang.String[] args) {
        JTATxEnd test = new JTATxEnd();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... does not exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
            s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.')");

            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }

    /**
     * This test use JTA support to handle transactions.
     */
    public void run() {
```

```

Connection c = null;

try {
    Context ctx = new InitialContext();

    // Assume the data source is backed by a UDBXADDataSource.
    UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADataSource");

    // From the DataSource, obtain an XAConnection object that
    // contains an XAResource and a Connection object.
    XAConnection xaConn = ds.getXAConnection();
    XAResource xaRes = xaConn.getXAResource();
    Connection c = xaConn.getConnection();

    // For XA transactions, transaction identifier is required.
    // An implementation of the XID interface is not included
    // with the JDBC driver. See Transactions with JTA for a
    // description of this interface to build a class for it.
    Xid xid = new XidImpl();

    // The connection from the XAResource can be used as any other
    // JDBC connection.
    Statement stmt = c.createStatement();

    // The XA resource must be notified before starting any
    // transactional work.
    xaRes.start(xid, XAResource.TMNOFLAGS);

    // Create a ResultSet during JDBC processing and fetch a row.
    ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
    rs.next();

    // When the end method is called, all ResultSet cursors close.
    // Accessing the ResultSet after this point results in an
    // exception being thrown.
    xaRes.end(xid, XAResource.TMNOFLAGS);

    try {
        String value = rs.getString(1);
        System.out.println("Something failed if you receive this message.");
    } catch (SQLException e) {
        System.out.println("The expected exception was thrown.");
    }

    // Commit the transaction to ensure that all locks are
    // released.
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);

} catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
} finally {
    try {
        if (c != null)
            c.close();
    } catch (SQLException e) {
        System.out.println("Note: Cleanup exception.");
        e.printStackTrace();
    }
}
}
}

```

## リンク集

コードのサンプルに関する特記事項

## JTA を使ったトランザクション

通常、Java Database Connectivity (JDBC) のトランザクションはローカルです。これは、単一の接続がトランザクションのすべての作業を行い、その接続では一度に 1 つのトランザクションだけが動作できることを意味します。このトランザクションのすべての動作が完了するか、失敗すると、永続化するためにコミットまたはロールバックが呼び出され、新しいトランザクションが開始されます。ただしこれは、ローカル・トランザクション以上の機能を提供する Java で使用可能な、トランザクションの拡張サポートです。このサポートの完全な仕様は、「Java Transaction API」を参照してください。

## 例: 無効なユーザー ID とパスワード

以下は、SQL 命名モードでの Connection プロパティの使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
//
// InvalidConnect example.
//
// This program uses the Connection property in SQL naming mode.
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////
import java.sql.*;
import java.util.*;

public class InvalidConnect {

    public static void main(java.lang.String[] args)
    {
        // Register the driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: JDBC driver did not load.");
            System.exit(0);
        }

        // Attempt to obtain a connection without specifying any user or
        // password. The attempt works and the connection uses the
        // same user profile under which the job is running.
    }
}
```

```

    try {
        Connection c1 = DriverManager.getConnection("jdbc:db2:*local");
        c1.close();
    } catch (SQLException e) {
        System.out.println("This test should not get into this exception path.");
        e.printStackTrace();
        System.exit(1);
    }

    try {
        Connection c2 = DriverManager.getConnection("jdbc:db2:*local",
                                                    "notvalid", "notvalid");
    } catch (SQLException e) {
        System.out.println("This is an expected error.");
        System.out.println("Message is " + e.getMessage());
        System.out.println("SQLSTATE is " + e.getSQLState());
    }
}
}
}

```

## 例: JDBC

以下に、BasicJDBC プログラムの使用法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
//
// BasicJDBC example. This program uses the native JDBC driver for the
// Developer Kit for Java to build a simple table and process a query
// that displays the data in that table.
//
// Command syntax:
//   BasicJDBC
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////

// Include any Java classes that are to be used. In this application,
// many classes from the java.sql package are used and the
// java.util.Properties class is also used as part of obtaining

```

```

// a connection to the database.
import java.sql.*;
import java.util.Properties;

// Create a public class to encapsulate the program.
public class BasicJDBC {

    // The connection is a private variable of the object.
    private Connection connection = null;

    // Any class that is to be an 'entry point' for running
    // a program must have a main method. The main method
    // is where processing begins when the program is called.
    public static void main(java.lang.String[] args) {

        // Create an object of type BasicJDBC. This
        // is fundamental to object-oriented programming. Once
        // an object is created, call various methods on
        // that object to accomplish work.
        // In this case, calling the constructor for the object
        // creates a database connection that the other
        // methods use to do work against the database.
        BasicJDBC test = new BasicJDBC();

        // Call the rebuildTable method. This method ensures that
        // the table used in this program exists and looks
        // correct. The return value is a boolean for
        // whether or not rebuilding the table completed
        // successfully. If it did no, display a message
        // and exit the program.
        if (!test.rebuildTable()) {
            System.out.println("Failure occurred while setting up " +
                " for running the test.");
            System.out.println("Test will not continue.");
            System.exit(0);
        }

        // The run query method is called next. This method
        // processes an SQL select statement against the table that
        // was created in the rebuildTable method. The output of
        // that query is output to standard out for you to view.
        test.runQuery();

        // Finally, the cleanup method is called. This method
        // ensures that the database connection that the object has
        // been hanging on to is closed.
        test.cleanup();
    }

    /**
    This is the constructor for the basic JDBC test. It creates a database
    connection that is stored in an instance variable to be used in later
    method calls.
    */
    public BasicJDBC() {

        // One way to create a database connection is to pass a URL
        // and a java Properties object to the DriverManager. The following
        // code constructs a Properties object that has your user ID and
        // password. These pieces of information are used for connecting
        // to the database.
        Properties properties = new Properties ();
        properties.put("user", "cujo");
        properties.put("user", "newtiger");

        // Use a try/catch block to catch all exceptions that can come out of the

```

```

// following code.
try {
    // The DriverManager must be aware that there is a JDBC driver available
    // to handle a user connection request. The following line causes the
    // native JDBC driver to be loaded and registered with the DriverManager.
    Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

    // Create the database Connection object that this program uses in all
    // the other method calls that are made. The following code specifies
    // that a connection is to be established to the local database and that
    // that connection should conform to the properties that were set up
    // previously (that is, it should use the user ID and password specified).
    connection = DriverManager.getConnection("jdbc:db2:*local", properties);

} catch (Exception e) {
    // If any of the lines in the try/catch block fail, control transfers to
    // the following line of code. A robust application tries to handle the
    // problem or provide more details to you. In this program, the error
    // message from the exception is displayed and the application allows
    // the program to return.
    System.out.println("Caught exception: " + e.getMessage());
}
}

/**
Ensures that the qgpl.basicjdbc table looks you want it to at the start of
the test.

@returns boolean    Returns true if the table was rebuild successfully;
                    returns false if any failure occurred.
**/
public boolean rebuildTable() {
    // Wrap all the functionality in a try/catch block so an attempt is
    // made to handle any errors that may happen within this method.
    try {

        // Statement objects are used to process SQL statements against the
        // database. The Connection object is used to create a Statement
        // object.
        Statement s = connection.createStatement();

        try {
            // Build the test table from scratch. Process an update statement
            // that attempts to delete the table if it currently exists.
            s.executeUpdate("drop table qgpl.basicjdbc");
        } catch (SQLException e) {
            // Do not perform anything if an exception occurred. Assume
            // that the problem is that the table that was dropped does not
            // exist and that it can be created next.
        }

        // Use the statement object to create our table.
        s.executeUpdate("create table qgpl.basicjdbc(id int, name char(15))");

        // Use the statement object to populate our table with some data.
        s.executeUpdate("insert into qgpl.basicjdbc values(1, 'Frank Johnson')");
        s.executeUpdate("insert into qgpl.basicjdbc values(2, 'Neil Schwartz')");
        s.executeUpdate("insert into qgpl.basicjdbc values(3, 'Ben Rodman')");
        s.executeUpdate("insert into qgpl.basicjdbc values(4, 'Dan Gloore')");

        // Close the SQL statement to tell the database that it is no longer
        // needed.
        s.close();

        // If the entire method processed successfully, return true. At this point,
        // the table has been created or refreshed correctly.
    }
}

```



```

        return true;
    } catch (SQLException sqle) {
        // If any of our SQL statements failed (other than the drop of the table
        // that was handled in the inner try/catch block), the error message is
        // displayed and false is returned to the caller, indicating that the table
        // may not be complete.
        System.out.println("Error in rebuildTable: " + sqle.getMessage());
        return false;
    }
}

/**
 * Runs a query against the demonstration table and the results are displayed to
 * standard out.
 */
public void runQuery() {
    // Wrap all the functionality in a try/catch block so an attempts is
    // made to handle any errors that might happen within this
    // method.
    try {
        // Create a Statement object.
        Statement s = connection.createStatement();

        // Use the statement object to run an SQL query. Queries return
        // ResultSet objects that are used to look at the data the query
        // provides.
        ResultSet rs = s.executeQuery("select * from qqpl.basicjdbc");

        // Display the top of our 'table' and initialize the counter for the
        // number of rows returned.
        System.out.println("-----");
        int i = 0;

        // The ResultSet next method is used to process the rows of a
        // ResultSet. The next method must be called once before the
        // first data is available for viewing. As long as next returns
        // true, there is another row of data that can be used.
        while (rs.next()) {

            // Obtain both columns in the table for each row and write a row to
            // our on-screen table with the data. Then, increment the count
            // of rows that have been processed.
            System.out.println("| " + rs.getInt(1) + " | " + rs.getString(2) + "|");
            i++;
        }

        // Place a border at the bottom on the table and display the number of rows
        // as output.
        System.out.println("-----");
        System.out.println("There were " + i + " rows returned.");
        System.out.println("Output is complete.");
    } catch (SQLException e) {
        // Display more information about any SQL exceptions that are
        // generated as output.
        System.out.println("SQLException exception: ");
        System.out.println("Message:....." + e.getMessage());
        System.out.println("SQLState:...." + e.getSQLState());
        System.out.println("Vendor Code:." + e.getErrorCode());
        e.printStackTrace();
    }
}

```

```

/**
The following method ensures that any JDBC resources that are still
allocated are freed.
**/
public void cleanup() {
    try {
        if (connection != null)
            connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## 例: 単一トランザクション上で動作する複数の接続

以下は、単一トランザクション上で動作する複数の接続の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;
public class JTAMultiConn {
    public static void main(java.lang.String[] args) {
        JTAMultiConn test = new JTAMultiConn();
        test.setup();
        test.run();
    }
/**
* Handle the previous cleanup run so that this test can recommence.
*/
    public void setup() {
        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            }
            catch (SQLException e) {
                // Ignore... does not exist
            }
            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR
                (50))");
            s.close();
        }
        finally {
            if (c != null) {
                c.close();
            }
        }
    }
/**
* This test uses JTA support to handle transactions.
*/
    public void run() {
        Connection c1 = null;
        Connection c2 = null;

```

```

Connection c3 = null;
try {
    Context ctx = new InitialContext();
    // Assume the data source is backed by a UDBXADDataSource.
    UDBXADDataSource ds = (UDBXADDataSource)
        ctx.lookup("XADataSource");
    // From the DataSource, obtain some XAConnection objects that
    // contain an XAResource and a Connection object.
    XAConnection xaConn1 = ds.getXAConnection();
    XAConnection xaConn2 = ds.getXAConnection();
    XAConnection xaConn3 = ds.getXAConnection();
    XAResource xaRes1 = xaConn1.getXAResource();
    XAResource xaRes2 = xaConn2.getXAResource();
    XAResource xaRes3 = xaConn3.getXAResource();
    c1 = xaConn1.getConnection();
    c2 = xaConn2.getConnection();
    c3 = xaConn3.getConnection();
    Statement stmt1 = c1.createStatement();
    Statement stmt2 = c2.createStatement();
    Statement stmt3 = c3.createStatement();
    // For XA transactions, a transaction identifier is required.
    // Support for creating XIDs is again left to the application
    // program.
    Xid xid = JDXATest.xidFactory();
    // Perform some transactional work under each of the three
    // connections that have been created.
    xaRes1.start(xid, XAResource.TMNOFLAGS);
    int count1 = stmt1.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-A')");
    xaRes1.end(xid, XAResource.TMNOFLAGS);

    xaRes2.start(xid, XAResource.TMJOIN);
    int count2 = stmt2.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-B')");
    xaRes2.end(xid, XAResource.TMNOFLAGS);

    xaRes3.start(xid, XAResource.TMJOIN);
    int count3 = stmt3.executeUpdate("INSERT INTO " + tableName + "VALUES('Value 1-C')");
    xaRes3.end(xid, XAResource.TMSUCCESS);
    // When completed, commit the transaction as a single unit.
    // A prepare() and commit() or 1 phase commit() is required for
    // each separate database (XAResource) that participated in the
    // transaction. Since the resources accessed (xaRes1, xaRes2, and xaRes3)
    // all refer to the same database, only one prepare or commit is required.
    int rc = xaRes.prepare(xid);
    xaRes.commit(xid, false);
}
catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
}
finally {
    try {
        try {
            if (c1 != null) {
                c1.close();
            }
        }
        catch (SQLException e) {
            System.out.println("Note: Cleanup exception " +
                e.getMessage());
        }
        try {
            if (c2 != null) {
                c2.close();
            }
        }
        catch (SQLException e) {
            System.out.println("Note: Cleanup exception " +
                e.getMessage());
        }
    }
}

```

```

    }
    try {
        if (c3 != null) {
            c3.close();
        }
    }
    catch (SQLException e) {
        System.out.println("Note: Cleanup exception " +
            e.getMessage());
    }
}
}
}
}

```

## 例: UDBDataSource をバインドする前に初期コンテキストを取得する

次の例では、UDBDataSource をバインドするにあたって、その前に初期コンテキストを取得します。その後、そのコンテキスト上で lookup メソッドを使用して、アプリケーションが使用する DataSource タイプのオブジェクトを戻します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

// Import the required packages. There is no
// driver-specific code needed in runtime
// applications.
import java.sql.*;
import javax.sql.*;
import javax.naming.*;

public class UDBDataSourceUse
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        // Retrieve a JNDI context. The context serves
        // as the root for where objects are bound or
        // found in JNDI.
        Context ctx = new InitialContext();

        // Retrieve the bound UDBDataSource object using the
        // name with which it was previously bound. At runtime,
        // only the DataSource interface is used, so there
        // is no need to convert the object to the UDBDataSource
        // implementation class. (There is no need to know what
        // the implementation class is. The logical JNDI name is
        // only required).
        DataSource ds = (DataSource) ctx.lookup("SimpleDS");

        // Once the DataSource is obtained, it can be used to establish
        // a connection. This Connection object is the same type
        // of object that is returned if the DriverManager approach
        // to establishing connection is used. Thus, so everything from
        // this point forward is exactly like any other JDBC
        // application.
        Connection connection = ds.getConnection();

        // The connection can be used to create Statement objects and
        // update the database or process queries as follows.
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery("select * from qsys2.sysprocs");
        while (rs.next()) {
            System.out.println(rs.getString(1) + "." + rs.getString(2));
        }
    }
}

```

```

        // The connection is closed before the application ends.
        connection.close();
    }
}

```

## 例: ParameterMetaData

これは、ParameterMetaData インターフェースを使用して、パラメーターについての情報を検索するときの一例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
//
// ParameterMetaData example. This program demonstrates
// the new support of JDBC 3.0 for learning information
// about parameters to a PreparedStatement.
//
// Command syntax:
//   java PMD
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////

import java.sql.*;

public class PMD {

    // Program entry point.
    public static void main(java.lang.String[] args)
        throws Exception
    {
        // Obtain setup.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        PreparedStatement ps = c.prepareStatement("INSERT INTO CUJOSQL.MYTABLE VALUES(?, ?, ?)");
        ParameterMetaData pmd = ps.getParameterMetaData();

        for (int i = 1; i < pmd.getParameterCount(); i++) {
            System.out.println("Parameter number " + i);
            System.out.println("  Class name is " + pmd.getParameterClassName(i));
        }
    }
}

```

```

        // Note: Mode relates to input, output or inout
        System.out.println(" Mode is " + pmd.getParameterClassName(i));
        System.out.println(" Type is " + pmd.getParameterType(i));
        System.out.println(" Type name is " + pmd.getParameterTypeName(i));
        System.out.println(" Precision is " + pmd.getPrecision(i));
        System.out.println(" Scale is " + pmd.getScale(i));
        System.out.println(" Nullable? is " + pmd.isNullable(i));
        System.out.println(" Signed? is " + pmd.isSigned(i));
    }
}
}

```

## 例: 他のステートメントのカーソルを介してステートメントで値を変更する

以下は、他のステートメントのカーソルを介したステートメントによる値の変更方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;

public class UsingPositionedUpdate {
    public Connection connection = null;
    public static void main(java.lang.String[] args) {

        UsingPositionedUpdate test = new UsingPositionedUpdate();

        test.setup();
        test.displayTable();

        test.run();
        test.displayTable();

        test.cleanup();
    }

    /**
    Handle all the required setup work.
    */
    public void setup() {
        try {
            // Register the JDBC driver.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            try {
                s.executeUpdate("DROP TABLE CUJOSQL.WHERECUREX");
            } catch (SQLException e) {
                // Ignore problems here.
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.WHERECUREX ( " +
                "COL_IND INT, COL_VALUE CHAR(20)) ");

            for (int i = 1; i <= 10; i++) {
                s.executeUpdate("INSERT INTO CUJOSQL.WHERECUREX VALUES(" + i + ", 'FIRST')");
            }

            s.close();

        } catch (Exception e) {
            System.out.println("Caught exception: " + e.getMessage());
        }
    }
}

```



```

        e.printStackTrace();
    }
}

/**
In this section, all the code to perform the testing should
be added. If only one connection to the database is required,
the global variable 'connection' can be used.
**/
public void run() {
    try {
        Statement stmt1 = connection.createStatement();

        // Update each value using next().
        stmt1.setCursorName("CUJO");
        ResultSet rs = stmt1.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX " +
            "FOR UPDATE OF COL_VALUE");

        System.out.println("Cursor name is " + rs.getCursorName());

        PreparedStatement stmt2 = connection.prepareStatement ("UPDATE "
            + " CUJOSQL.WHERECUREX
            SET COL_VALUE = 'CHANGED'
            WHERE CURRENT OF "
            + rs.getCursorName ());

        // Loop through the ResultSet and update every other entry.
        while (rs.next ()) {
            if (rs.next())
                stmt2.execute ();
        }

        // Clean up the resources after they have been used.
        rs.close ();
        stmt2.close ();

    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}

/**
In this section, put all clean-up work for testing.
**/
public void cleanup() {
    try {
        // Close the global connection opened in setup().
        connection.close();

    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}

/**
Display the contents of the table.
**/

```

```

public void displayTable()
{
    try {
        Statement s = connection.createStatement();
        ResultSet rs = s.executeQuery ("SELECT * FROM CUJOSQL.WHERECUREX");

        while (rs.next ()) {
            System.out.println("Index " + rs.getInt(1) + " value " + rs.getString(2));
        }

        rs.close ();
        s.close();
        System.out.println("-----");
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## リンク集

コードのサンプルに関する特記事項

## 例: IBM Developer Kit for Java 用の ResultSet インターフェース

以下は、ResultSet インターフェースの使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;

/**
 * ResultSetExample.java
 *
 * This program demonstrates using a ResultSetMetaData and
 * a ResultSet to display all the data in a table even though
 * the program that gets the data does not know what the table
 * is going to look like (the user passes in the values for the
 * table and library).
 */
public class ResultSetExample {

    public static void main(java.lang.String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage: java ResultSetExample <library> <table>");
            System.out.println(" where <library> is the library that contains <table>");
            System.exit(0);
        }

        Connection con = null;
        Statement s = null;
        ResultSet rs = null;
        ResultSetMetaData rsmd = null;

        try {
            // Get a database connection and prepare a statement.
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            con = DriverManager.getConnection("jdbc:db2:*local");

            s = con.createStatement();

            rs = s.executeQuery("SELECT * FROM " + args[0] + "." + args[1]);
            rsmd = rs.getMetaData();

```



```

Statement s = connection.createStatement();
try {
    s.executeUpdate("drop table cujosql.sensitive");
} catch (SQLException e) {
    // Ignored.
}

s.executeUpdate("create table cujosql.sensitive(col1 int)");
s.executeUpdate("insert into cujosql.sensitive values(1)");
s.executeUpdate("insert into cujosql.sensitive values(2)");
s.executeUpdate("insert into cujosql.sensitive values(3)");
s.executeUpdate("insert into cujosql.sensitive values(4)");
s.executeUpdate("insert into cujosql.sensitive values(5)");

try {
    s.executeUpdate("drop table cujosql.sensitive2");
} catch (SQLException e) {
    // Ignored.
}

s.executeUpdate("create table cujosql.sensitive2(col2 int)");
s.executeUpdate("insert into cujosql.sensitive2 values(1)");
s.executeUpdate("insert into cujosql.sensitive2 values(2)");
s.executeUpdate("insert into cujosql.sensitive2 values(3)");
s.executeUpdate("insert into cujosql.sensitive2 values(4)");
s.executeUpdate("insert into cujosql.sensitive2 values(5)");

s.close();

} catch (Exception e) {
    System.out.println("Caught exception: " + e.getMessage());
    if (e instanceof SQLException) {
        SQLException another = ((SQLException) e).getNextException();
        System.out.println("Another: " + another.getMessage());
    }
}

}

public void run(String sensitivity) {
    try {

        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select col1, col2 From cujosql.sensitive,
            cujosql.sensitive2 where col1 = col2");

        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
        System.out.println("value is " + rs.getInt(1));
        rs.next();
    }
}

```

```

System.out.println("value is " + rs.getInt(1));
rs.next();
System.out.println("value is " + rs.getInt(1));

System.out.println("fetched the four rows...");

// Another statement creates a value that does not fit the where clause.
Statement s2 =
    connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATEABLE);
ResultSet rs2 = s2.executeQuery("select *
from cujosql.sensitive where col1 = 5 FOR UPDATE");
rs2.next();
rs2.updateInt(1, -1);
rs2.updateRow();
s2.close();

if (rs.next()) {
    System.out.println("There is still a row: " + rs.getInt(1));
} else {
    System.out.println("No more rows.");
}

} catch (SQLException e) {
    System.out.println("SQLException exception: ");
    System.out.println("Message:....." + e.getMessage());
    System.out.println("SQLState:....." + e.getSQLState());
    System.out.println("Vendor Code:." + e.getErrorCode());
    System.out.println("-----");
    e.printStackTrace();
}
catch (Exception ex) {
    System.out.println("An exception other
than an SQLException was thrown: ");
    ex.printStackTrace();
}
}

public void cleanup() {
    try {
        connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## 例: 感知および非感知の ResultSet

以下の例は、テーブルに行が挿入される際の、感知 ResultSet と非感知 ResultSet との違いを示しています。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;

public class Sensitive {

```

```

public Connection connection = null;

public static void main(java.lang.String[] args) {
    Sensitive test = new Sensitive();

    test.setup();
    test.run("sensitive");
    test.cleanup();

    test.setup();
    test.run("insensitive");
    test.cleanup();
}

public void setup() {
    try {
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        connection = DriverManager.getConnection("jdbc:db2:*local");

        Statement s = connection.createStatement();
        try {
            s.executeUpdate("drop table cujosql.sensitive");
        } catch (SQLException e) {
            // Ignored.
        }

        s.executeUpdate("create table cujosql.sensitive(coll int)");
        s.executeUpdate("insert into cujosql.sensitive values(1)");
        s.executeUpdate("insert into cujosql.sensitive values(2)");
        s.executeUpdate("insert into cujosql.sensitive values(3)");
        s.executeUpdate("insert into cujosql.sensitive values(4)");
        s.executeUpdate("insert into cujosql.sensitive values(5)");
        s.close();

    } catch (Exception e) {
        System.out.println("Caught exception: " + e.getMessage());
        if (e instanceof SQLException) {
            SQLException another = ((SQLException) e).getNextException();
            System.out.println("Another: " + another.getMessage());
        }
    }
}

public void run(String sensitivity) {
    try {
        Statement s = null;
        if (sensitivity.equalsIgnoreCase("insensitive")) {
            System.out.println("creating a TYPE_SCROLL_INSENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        } else {
            System.out.println("creating a TYPE_SCROLL_SENSITIVE cursor");
            s = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        }

        ResultSet rs = s.executeQuery("select * From cujosql.sensitive");

        // Fetch the five values that are there.
        rs.next();
    }
}

```



```

System.out.println("value is " + rs.getInt(1));
rs.next();
System.out.println("value is " + rs.getInt(1));
rs.next();
System.out.println("value is " + rs.getInt(1));
rs.next();
System.out.println("value is " + rs.getInt(1));
rs.next();
System.out.println("value is " + rs.getInt(1));
rs.next();
System.out.println("value is " + rs.getInt(1));
System.out.println("fetched the five rows...");

// Note: If you fetch the last row, the ResultSet looks
//        closed and subsequent new rows that are added
//        are not be recognized.

// Allow another statement to insert a new value.
Statement s2 = connection.createStatement();
s2.executeUpdate("insert into cujosql.sensitive values(6)");
s2.close();

// Whether a row is recognized is based on the sensitivity setting.
if (rs.next()) {
    System.out.println("There is a row now: " + rs.getInt(1));
} else {
    System.out.println("No more rows.");
}

} catch (SQLException e) {
    System.out.println("SQLException exception: ");
    System.out.println("Message:....." + e.getMessage());
    System.out.println("SQLState:....." + e.getSQLState());
    System.out.println("Vendor Code:." + e.getErrorCode());
    System.out.println("-----");
    e.printStackTrace();
}
catch (Exception ex) {
    System.out.println("An exception other than an SQLException was thrown: ");
    ex.printStackTrace();
}
}

public void cleanup() {
    try {
        connection.close();
    } catch (Exception e) {
        System.out.println("Caught exception: ");
        e.printStackTrace();
    }
}
}

```

## 例: UDBDataSource および UDBConnectionPoolDataSource で接続プーリングをセットアップする

以下に、UDBDataSource および UDBConnectionPoolDataSource で接続プーリングを使用する方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;
import javax.naming.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class ConnectionPoolingSetup
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        // Create a ConnectionPoolDataSource implementation
        UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
        cpds.setDescription("Connection Pooling DataSource object");

        // Establish a JNDI context and bind the connection pool data source
        Context ctx = new InitialContext();
        ctx.rebind("ConnectionSupport", cpds);

        // Create a standard data source that references it.
        UDBDataSource ds = new UDBDataSource();
        ds.setDescription("DataSource supporting pooling");
        ds.setDataSourceName("ConnectionSupport");
        ctx.rebind("PoolingDataSource", ds);
    }
}
```

## 例: SQLException

以下に、SQLException をキャッチし、提供されたすべての情報をダンプする例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.sql.*;

public class ExceptionExample {

    public static Connection connection = null;

    public static void main(java.lang.String[] args) {

        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            connection = DriverManager.getConnection("jdbc:db2:*local");

            Statement s = connection.createStatement();
            int count = s.executeUpdate("insert into cujofake.cujofake values(1, 2,3)");

            System.out.println("Did not expect that table to exist.");

        } catch (SQLException e) {
            System.out.println("SQLException exception: ");
            System.out.println("Message:....." + e.getMessage());
            System.out.println("SQLState:...." + e.getSQLState());
            System.out.println("Vendor Code:." + e.getErrorCode());
            System.out.println("-----");
            e.printStackTrace();
        } catch (Exception ex) {
            System.out.println("An exception other than an SQLException was thrown: ");
            ex.printStackTrace();
        } finally {
            try {
```



```

* This test uses JTA support to handle transactions.
*/
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADatasource.
        UDBXADatasource ds = (UDBXADatasource) ctx.lookup("XADatasource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with
        // the JDBC driver. Transactions with JTA for a
        // description of this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Create a ResultSet during JDBC processing and fetch a row.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // The end method is called with the suspend option. The
        // ResultSets associated with the current transaction are 'on hold'.
        // They are neither gone nor accessible in this state.
        xaRes.end(xid, XAResource.TMSUSPEND);

        // Other work can be performed with the transaction.
        // As an example, you can create a statement and process a query.
        // This work and any other transactional work that the transaction may
        // perform is separate from the work done previously under the XID.
        Statement nonXASmt = conn.createStatement();
        ResultSet nonXARS = nonXASmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Process here...
        }
        nonXARS.close();
        nonXASmt.close();

        // If an attempt is made to use any suspended transactions
        // resources, an exception results.
        try {
            rs.getString(1);
            System.out.println("Value of the first row is " + rs.getString(1));
        } catch (SQLException e) {
            System.out.println("This was an expected exception - " +
                "suspended ResultSet was used.");
        }

        // Resume the suspended transaction and complete the work on it.
        // The ResultSet is exactly as it was before the suspension.
    }
}

```

```

xaRes.start(newXid, XAResource.TMRESUME);
rs.next();
System.out.println("Value of the second row is " + rs.getString(1));

// When the transaction has completed, end it
// and commit any work under it.
xaRes.end(xid, XAResource.TMNOFLAGS);
int rc = xaRes.prepare(xid);
xaRes.commit(xid, false);

} catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
} finally {
    try {
        if (c != null)
            c.close();
    } catch (SQLException e) {
        System.out.println("Note: Cleanup exception.");
        e.printStackTrace();
    }
}
}
}
}

```

## 例: 中断状態の ResultSets

以下は、作業を実行するために Statement オブジェクトを別のトランザクションで再処理する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTATxEffect {

    public static void main(java.lang.String[] args) {
        JTATxEffect test = new JTATxEffect();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            }
        }
    }
}

```

```

    } catch (SQLException e) {
        // Ignore... does not exist
    }

    s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
    s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Fun with JTA')");
    s.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is fun.')");

    s.close();
} finally {
    if (c != null) {
        c.close();
    }
}
}

/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADatasource.
        UDBXADatasource ds = (UDBXADatasource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with
        // the JDBC driver. See Transactions with JTA
        // for a description of this interface to build a
        // class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Create a ResultSet during JDBC processing and fetch a row.
        ResultSet rs = stmt.executeUpdate("SELECT * FROM CUJOSQL.JTATABLE");
        rs.next();

        // The end method is called with the suspend option. The
        // ResultSets associated with the current transaction are 'on hold'.
        // They are neither gone nor accessible in this state.
        xaRes.end(xid, XAResource.TMSUSPEND);

        // In the meantime, other work can be done outside the transaction.
        // The ResultSets under the transaction can be closed if the
        // Statement object used to create them is reused.
        ResultSet nonXARS = stmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
        while (nonXARS.next()) {
            // Process here...
        }
    }
}

```

```

// Attempt to go back to the suspended transaction. The suspended
// transaction's ResultSet has disappeared because the statement
// has been processed again.
xaRes.start(newXid, XAResource.TMRESUME);
try {
    rs.next();
} catch (SQLException ex) {
    System.out.println("This exception is expected. " +
        "The ResultSet closed due to another process.");
}

// When the transaction had completed, end it
// and commit any work under it.
xaRes.end(xid, XAResource.TMNOFLAGS);
int rc = xaRes.prepare(xid);
xaRes.commit(xid, false);

} catch (Exception e) {
    System.out.println("Something has gone wrong.");
    e.printStackTrace();
} finally {
    try {
        if (c != null)
            c.close();
    } catch (SQLException e) {
        System.out.println("Note: Cleanup exception.");
        e.printStackTrace();
    }
}
}
}
}

```

## 例: 接続プーリングのパフォーマンスをテストする

以下に、プーリングされたときのパフォーマンスとプーリングされていないときのパフォーマンスを対比してテストする方法を示します。

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;

public class ConnectionPoolingTest
{
    public static void main(java.lang.String[] args)
        throws Exception
    {
        Context ctx = new InitialContext();
        // Do the work without a pool:
        DataSource ds = (DataSource) ctx.lookup("BaseDataSource");
        System.out.println("Start timing the non-pooling DataSource version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));
    }
}

```



```

// Do the work with pooling:
ds = (DataSource) ctx.lookup("PoolingDataSource");
System.out.println("Start timing the pooling version...");

startTime = System.currentTimeMillis();
for (int i = 0; i < 100; i++) {
    Connection c1 = ds.getConnection();
    c1.close();
}
endTime = System.currentTimeMillis();
System.out.println("Time spent: " + (endTime - startTime));
}
}

```

## 例: 2 つの DataSource のパフォーマンスをテストする

次に、接続プーリングだけを使用する 1 つの DataSource と、ステートメントと接続プーリングを使用する別の DataSource をテストする例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.naming.*;
import java.util.*;
import javax.sql.*;
import com.ibm.db2.jdbc.app.UDBDataSource;
import com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource;

public class StatementPoolingTest
{
    public static void main(java.lang.String[] args)
    throws Exception
    {
        Context ctx = new InitialContext();

        System.out.println("deploying statement pooling data source");
        deployStatementPoolDataSource();

        // Do the work with connection pooling only.
        DataSource ds = (DataSource) ctx.lookup("PoolingDataSource");
        System.out.println("Start timing the connection pooling only version...");

        long startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("Time spent: " + (endTime - startTime));

        // Do the work with statement pooling added.
        ds = (DataSource) ctx.lookup("StatementPoolingDataSource");
        System.out.println("Start timing the statement pooling version...");

        startTime = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Connection c1 = ds.getConnection();
            PreparedStatement ps = c1.prepareStatement("select * from qsys2.sysprocs");
            ResultSet rs = ps.executeQuery();
            c1.close();
        }
    }
}

```

```

    }
    endTime = System.currentTimeMillis();
    System.out.println("Time spent: " + (endTime - startTime));
}

private static void deployStatementPoolDataSource()
throws Exception
{
    // Create a ConnectionPoolDataSource implementation
    UDBConnectionPoolDataSource cpds = new UDBConnectionPoolDataSource();
    cpds.setDescription("Connection Pooling DataSource object with Statement pooling");
    cpds.setMaxStatements(10);

    // Establish a JNDI context and bind the connection pool data source
    Context ctx = new InitialContext();
    ctx.rebind("StatementSupport", cpds);

    // Create a standard datasource that references it.
    UDBDataSource ds = new UDBDataSource();
    ds.setDescription("DataSource supporting statement pooling");
    ds.setDataSourceName("StatementSupport");
    ctx.rebind("StatementPoolingDataSource", ds);
}
}

```

## 例: BLOB の更新

以下は、アプリケーション中で BLOB を更新する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UpdateBlobs is an example application
// that shows some of the APIs providing
// support for changing Blob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetBlobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UpdateBlobs {
    public static void main(String[] args)
    throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

        rs.next();
        Blob blob1 = rs.getBlob(1);
    }
}

```

```

rs.next();
Blob blob2 = rs.getBlob(1);

// Truncate a BLOB.
blob1.truncate((long) 150000);
System.out.println("Blob1's new length is " + blob1.length());

// Update part of the BLOB with a new byte array.
// The following code obtains the bytes that are at
// positions 4000-4500 and set them to positions 500-1000.

// Obtain part of the BLOB as a byte array.
byte[] bytes = blob1.getBytes(4000L, 4500);

int bytesWritten = blob2.setBytes(500L, bytes);

System.out.println("Bytes written is " + bytesWritten);

// The bytes are now found at position 500 in blob2
long startInBlob2 = blob2.position(bytes, 1);

System.out.println("pattern found starting at position " + startInBlob2);

c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: CLOB の更新

以下は、アプリケーション中で CLOB を更新する方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetClobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UpdateClobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

        rs.next();
        Clob clob1 = rs.getClob(1);
        rs.next();
    }
}

```

```

Clob clob2 = rs.getClob(1);

// Truncate a CLOB.
clob1.truncate((long) 150000);
System.out.println("Clob1's new length is " + clob1.length());

// Update a portion of the CLOB with a new String value.
String value = "Some new data for once";
int charsWritten = clob2.setString(500L, value);

System.out.println("Characters written is " + charsWritten);

// The bytes can be found at position 500 in clob2
long startInClob2 = clob2.position(value, 1);

System.out.println("pattern found starting at position " + startInClob2);

c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: 複数のトランザクションで単一の接続を使用する

以下は、複数のトランザクションでの単一接続の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTAMultiTx {

    public static void main(java.lang.String[] args) {
        JTAMultiTx test = new JTAMultiTx();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... does not exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");

```

```

        s.close();
    } finally {
        if (c != null) {
            c.close();
        }
    }
}

/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADDataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();
        Statement stmt = c.createStatement();

        // For XA transactions, a transaction identifier is required.
        // This is not meant to imply that all the XIDs are the same.
        // Each XID must be unique to distinguish the various transactions
        // that occur.
        // Support for creating XIDs is again left to the application
        // program.
        Xid xid1 = JDXTTest.xidFactory();
        Xid xid2 = JDXTTest.xidFactory();
        Xid xid3 = JDXTTest.xidFactory();

        // Do work under three transactions for this connection.
        xaRes.start(xid1, XAResource.TMNOFLAGS);
        int count1 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-A')");
        xaRes.end(xid1, XAResource.TMNOFLAGS);

        xaRes.start(xid2, XAResource.TMNOFLAGS);
        int count2 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-B')");
        xaRes.end(xid2, XAResource.TMNOFLAGS);

        xaRes.start(xid3, XAResource.TMNOFLAGS);
        int count3 = stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('Value 1-C')");
        xaRes.end(xid3, XAResource.TMNOFLAGS);

        // Prepare all the transactions
        int rc1 = xaRes.prepare(xid1);
        int rc2 = xaRes.prepare(xid2);
        int rc3 = xaRes.prepare(xid3);

        // Two of the transactions commit and one rolls back.
        // The attempt to insert the second value into the table is
        // not committed.
        xaRes.commit(xid1, false);
        xaRes.rollback(xid2);
        xaRes.commit(xid3, false);

    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    }
}

```

```

    } finally {
        try {
            if (c != null)
                c.close();
        } catch (SQLException e) {
            System.out.println("Note: Cleanup exception.");
            e.printStackTrace();
        }
    }
}
}
}

```

## 例: BLOB の使用

以下は、アプリケーション中で BLOB を使用方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UseBlobs is an example application
// that shows some of the APIs associated
// with Blob objects.
//
// This program must be run after
// the PutGetBlobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UseBlobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.BLOBTABLE");

        rs.next();
        Blob blob1 = rs.getBlob(1);
        rs.next();
        Blob blob2 = rs.getBlob(1);

        // Determine the length of a LOB.
        long end = blob1.length();
        System.out.println("Blob1 length is " + blob1.length());

        // When working with LOBs, all indexing that is related to them
        // is 1-based, and is not 0-based like strings and arrays.
        long startingPoint = 450;
        long endingPoint = 500;

        // Obtain part of the BLOB as a byte array.
        byte[] outByteArray = blob1.getBytes(startingPoint, (int)endingPoint);

        // Find where a sub-BLOB or byte array is first found within a
        // BLOB. The setup for this program placed two identical copies of
        // a random BLOB into the database. Thus, the start position of the

```

```

// byte array extracted from blob1 can be found in the starting
// position in blob2. The exception would be if there were 50
// identical random bytes in the LOBs previously.
long startInBlob2 = blob2.position(outByteArray, 1);

System.out.println("pattern found starting at position " + startInBlob2);

c.close(); // Connection close closes stmt and rs too.
}
}

```

## 例: CLOB の使用

以下は、アプリケーション中で CLOB を使用方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
// UpdateClobs is an example application
// that shows some of the APIs providing
// support for changing Clob objects
// and reflecting those changes to the
// database.
//
// This program must be run after
// the PutGetClobs program has completed.
////////////////////////////////////
import java.sql.*;

public class UseClobs {
    public static void main(String[] args)
        throws SQLException
    {
        // Register the native JDBC driver.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
        } catch (Exception e) {
            System.exit(1); // Setup error.
        }

        Connection c = DriverManager.getConnection("jdbc:db2:*local");
        Statement s = c.createStatement();

        ResultSet rs = s.executeQuery("SELECT * FROM CUJOSQL.CLOBTABLE");

        rs.next();
        Clob clob1 = rs.getClob(1);
        rs.next();
        Clob clob2 = rs.getClob(1);

        // Determine the length of a LOB.
        long end = clob1.length();
        System.out.println("Clob1 length is " + clob1.length());

        // When working with LOBs, all indexing that is related to them
        // is 1-based, and not 0-based like strings and arrays.
        long startingPoint = 450;
        long endingPoint = 50;

        // Obtain part of the CLOB as a byte array.
        String outString = clob1.getSubString(startingPoint, (int)endingPoint);
        System.out.println("Clob substring is " + outString);

        // Find where a sub-CLOB or string is first found within a

```



```

    // CLOB. The setup for this program placed two identical copies of
    // a repeating CLOB into the database. Thus, the start position of the
    // string extracted from clob1 can be found in the starting
    // position in clob2 if the search begins close to the position where
    // the string starts.
    long startInClob2 = clob2.position(outString, 440);

    System.out.println("pattern found starting at position " + startInClob2);

    c.close(); // Connection close also closes stmt and rs.
}
}

```

## 例: トランザクションを処理するために JTA を使用する

以下は、アプリケーション内でトランザクションを処理するための Java Transaction API (JTA) の使用法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import javax.sql.*;
import java.util.*;
import javax.transaction.*;
import javax.transaction.xa.*;
import com.ibm.db2.jdbc.app.*;

public class JTACommit {

    public static void main(java.lang.String[] args) {
        JTACommit test = new JTACommit();

        test.setup();
        test.run();
    }

    /**
     * Handle the previous cleanup run so that this test can recommence.
     */
    public void setup() {

        Connection c = null;
        Statement s = null;
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            c = DriverManager.getConnection("jdbc:db2:*local");
            s = c.createStatement();

            try {
                s.executeUpdate("DROP TABLE CUJOSQL.JTATABLE");
            } catch (SQLException e) {
                // Ignore... does not exist
            }

            s.executeUpdate("CREATE TABLE CUJOSQL.JTATABLE (COL1 CHAR (50))");
            s.close();
        } finally {
            if (c != null) {
                c.close();
            }
        }
    }
}

```

```

/**
 * This test uses JTA support to handle transactions.
 */
public void run() {
    Connection c = null;

    try {
        Context ctx = new InitialContext();

        // Assume the data source is backed by a UDBXADDataSource.
        UDBXADDataSource ds = (UDBXADDataSource) ctx.lookup("XADataSource");

        // From the DataSource, obtain an XAConnection object that
        // contains an XAResource and a Connection object.
        XAConnection xaConn = ds.getXAConnection();
        XAResource xaRes = xaConn.getXAResource();
        Connection c = xaConn.getConnection();

        // For XA transactions, a transaction identifier is required.
        // An implementation of the XID interface is not included with the
        // JDBC driver. See Transactions with JTA for a description of
        // this interface to build a class for it.
        Xid xid = new XidImpl();

        // The connection from the XAResource can be used as any other
        // JDBC connection.
        Statement stmt = c.createStatement();

        // The XA resource must be notified before starting any
        // transactional work.
        xaRes.start(xid, XAResource.TMNOFLAGS);

        // Standard JDBC work is performed.
        int count =
            stmt.executeUpdate("INSERT INTO CUJOSQL.JTATABLE VALUES('JTA is pretty fun.')");

        // When the transaction work has completed, the XA resource must
        // again be notified.
        xaRes.end(xid, XAResource.TMSUCCESS);

        // The transaction represented by the transaction ID is prepared
        // to be committed.
        int rc = xaRes.prepare(xid);

        // The transaction is committed through the XAResource.
        // The JDBC Connection object is not used to commit
        // the transaction when using JTA.
        xaRes.commit(xid, false);

    } catch (Exception e) {
        System.out.println("Something has gone wrong.");
        e.printStackTrace();
    } finally {
        try {
            if (c != null)
                c.close();
        } catch (SQLException e) {
            System.out.println("Note: Cleanup exception.");
            e.printStackTrace();
        }
    }
}
}

```

## 例: 複数の列を持ったメタデータ ResultSet を使用する

以下は、複数の列があるメタデータ ResultSet を使用方法の例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
//
// SafeGetUDTs example. This program demonstrates one way to deal with
// metadata ResultSets that have more columns in JDK 1.4 than they
// had in previous releases.
//
// Command syntax:
//   java SafeGetUDTs
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////

import java.sql.*;

public class SafeGetUDTs {

    public static int jdbcLevel;

    // Note: Static block runs before main begins.
    // Therefore, there is access to jdbcLevel in
    // main.
    {
        try {
            Class.forName("java.sql.Blob");

            try {
                Class.forName("java.sql.ParameterMetaData");
                // Found a JDBC 3.0 interface. Must support JDBC 3.0.
                jdbcLevel = 3;
            } catch (ClassNotFoundException ez) {
                // Could not find the JDBC 3.0 ParameterMetaData class.
                // Must be running under a JVM with only JDBC 2.0
                // support.
                jdbcLevel = 2;
            }
        }
    }

    } catch (ClassNotFoundException ex) {
```

```

        // Could not find the JDBC 2.0 Blob class. Must be
        // running under a JVM with only JDBC 1.0 support.
        jdbcLevel = 1;
    }
}

// Program entry point.
public static void main(java.lang.String[] args)
{
    Connection c = null;

    try {
        // Get the driver registered.
        Class.forName("com.ibm.db2.jdbc.app.DB2Driver");

        c = DriverManager.getConnection("jdbc:db2:*local");
        DatabaseMetaData dmd = c.getMetaData();

        if (jdbcLevel == 1) {
            System.out.println("No support is provided for getUDTs. Just return.");
            System.exit(1);
        }

        ResultSet rs = dmd.getUDTs(null, "CUJOSQL", "SSN%", null);
        while (rs.next()) {

            // Fetch all the columns that have been available since the
            // JDBC 2.0 release.
            System.out.println("TYPE_CAT is " + rs.getString("TYPE_CAT"));
            System.out.println("TYPE_SCHEM is " + rs.getString("TYPE_SCHEM"));
            System.out.println("TYPE_NAME is " + rs.getString("TYPE_NAME"));
            System.out.println("CLASS_NAME is " + rs.getString("CLASS_NAME"));
            System.out.println("DATA_TYPE is " + rs.getString("DATA_TYPE"));
            System.out.println("REMARKS is " + rs.getString("REMARKS"));

            // Fetch all the columns that were added in JDBC 3.0.
            if (jdbcLevel > 2) {
                System.out.println("BASE_TYPE is " + rs.getString("BASE_TYPE"));
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    } finally {
        if (c != null) {
            try {
                c.close();
            } catch (SQLException e) {
                // Ignoring shutdown exception.
            }
        }
    }
}
}

```

## 例: ネイティブ JDBC と IBM Toolbox for Java JDBC を同時に使用する

以下に、プログラム中でネイティブ JDBC 接続と IBM Toolbox for Java JDBC 接続を使用する方法を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

////////////////////////////////////
//
// GetConnections example.
//
// This program demonstrates being able to use both JDBC drivers at
// once in a program. Two Connection objects are created in this
// program. One is a native JDBC connection and one is a IBM Toolbox for Java
// JDBC connection.
//
// This technique is convenient because it allows you to use different
// JDBC drivers for different tasks concurrently. For example, the
// IBM Toolbox for Java JDBC driver is ideal for connecting to remote iSeries servers
// and the native JDBC driver is faster for local connections.
// You can use the strengths of each driver concurrently in your
// application by writing code similar to this example.
//
////////////////////////////////////
//
// This source is an example of the IBM Developer for Java JDBC driver.
// IBM grants you a nonexclusive license to use this as an example
// from which you can generate similar function tailored to
// your own specific needs.
//
// This sample code is provided by IBM for illustrative purposes
// only. These examples have not been thoroughly tested under all
// conditions. IBM, therefore, cannot guarantee or imply
// reliability, serviceability, or function of these programs.
//
// All programs contained herein are provided to you "AS IS"
// without any warranties of any kind. The implied warranties of
// merchantability and fitness for a particular purpose are
// expressly disclaimed.
//
// IBM Developer Kit for Java
// (C) Copyright IBM Corp. 2001
// All rights reserved.
// US Government Users Restricted Rights -
// Use, duplication, or disclosure restricted
// by GSA ADP Schedule Contract with IBM Corp.
//
////////////////////////////////////
import java.sql.*;
import java.util.*;

public class GetConnections {

    public static void main(java.lang.String[] args)
    {
        // Verify input.
        if (args.length != 2) {
            System.out.println("Usage (CL command line): java GetConnections PARM(<user> <password>");
            System.out.println(" where <user> is a valid iSeries user ID");
            System.out.println(" and <password> is the password for that user ID");
            System.exit(0);
        }

        // Register both drivers.
        try {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver");
            Class.forName("com.ibm.as400.access.AS400JDBCdriver");
        } catch (ClassNotFoundException cnf) {
            System.out.println("ERROR: One of the JDBC drivers did not load.");
            System.exit(0);
        }

        try {
            // Obtain a connection with each driver.

```

```

Connection conn1 = DriverManager.getConnection("jdbc:db2://localhost", args[0], args[1]);
Connection conn2 = DriverManager.getConnection("jdbc:as400://localhost", args[0], args[1]);

// Verify that they are different.
if (conn1 instanceof com.ibm.db2.jdbc.app.DB2Connection)
    System.out.println("conn1 is running under the native JDBC driver.");
else
    System.out.println("There is something wrong with conn1.");

if (conn2 instanceof com.ibm.as400.access.AS400JDBCCConnection)
    System.out.println("conn2 is running under the IBM Toolbox for Java JDBC driver.");
else
    System.out.println("There is something wrong with conn2.");

    conn1.close();
    conn2.close();
} catch (SQLException e) {
    System.out.println("ERROR: " + e.getMessage());
}
}
}

```

## リンク集

コードのサンプルに関する特記事項

## 例: ResultSet を取得するために PreparedStatement を使用する

これは、PreparedStatement オブジェクトの executeQuery メソッドを使用して、ResultSet を入手するときの一例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import java.util.Properties;

public class PreparedStatementExample {

    public static void main(java.lang.String[] args)
    {
        // Load the following from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Register the native JDBC driver. If the driver cannot
        // be registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }

        Connection c = null;
        Statement s = null;

        // This program creates a table that is
        // used by prepared statements later.
        try {
            // Create the connection properties.
            Properties properties = new Properties ();
            properties.put ("user", "userid");
            properties.put ("password", "password");

```

```

// Connect to the local iSeries database.
c = DriverManager.getConnection(URL, properties);

// Create a Statement object.
s = c.createStatement();
// Delete the test table if it exists. Note that
// this example assumes throughout that the collection
// MYLIBRARY exists on the system.
try {
    s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
} catch (SQLException e) {
    // Just continue... the table probably did not exist.
}

// Run an SQL statement that creates a table in the database.
s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (s != null) {
            s.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}

// This program then uses a prepared statement to insert many
// rows into the database.
PreparedStatement ps = null;
String[] nameArray = {"Rich", "Fred", "Mark", "Scott", "Jason",
    "John", "Jessica", "Blair", "Erica", "Barb"};
try {
    // Create a PreparedStatement object that is used to insert data into the
    // table.
    ps = c.prepareStatement("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES (?, ?)");

    for (int i = 0; i < nameArray.length; i++) {
        ps.setString(1, nameArray[i]);    // Set the Name from our array.
        ps.setInt(2, i+1);                // Set the ID.
        ps.executeUpdate();
    }

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}

// Use a prepared statement to query the database
// table that has been created and return data from it. In
// this example, the parameter used is arbitrarily set to
// 5, meaning return all rows where the ID field is less than

```



```

//    or equal to 5.
try {
    ps = c.prepareStatement("SELECT * FROM MYLIBRARY.MYTABLE " +
                            "WHERE ID <= ?");

    ps.setInt(1, 5);

    // Run an SQL query on the table.
    ResultSet rs = ps.executeQuery();
    // Display all the data in the table.
    while (rs.next()) {
        System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
    }

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (ps != null) {
            ps.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }

    try {
        if (c != null) {
            c.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Connection.");
    }

}
}
}

```

## 例: Statement オブジェクトの executeUpdate メソッドを使用する

次に、Statement オブジェクトの executeUpdate メソッドを使用する方法の例を示します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import java.util.Properties;

public class StatementExample {

    public static void main(java.lang.String[] args)
    {

        // Suggestion: Load these from a properties object.
        String DRIVER = "com.ibm.db2.jdbc.app.DB2Driver";
        String URL    = "jdbc:db2://*local";

        // Register the native JDBC driver. If the driver cannot be
        // registered, the test cannot continue.
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            System.out.println("Driver failed to register.");
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}

```

```

}

Connection c = null;
Statement s = null;

try {
    // Create the connection properties.
    Properties properties = new Properties ();
    properties.put ("user", "userid");
    properties.put ("password", "password");

    // Connect to the local iSeries database.
    c = DriverManager.getConnection(URL, properties);

    // Create a Statement object.
    s = c.createStatement();
    // Delete the test table if it exists. Note: This
    // example assumes that the collection MYLIBRARY
    // exists on the system.
    try {
        s.executeUpdate("DROP TABLE MYLIBRARY.MYTABLE");
    } catch (SQLException e) {
        // Just continue... the table probably does not exist.
    }

    // Run an SQL statement that creates a table in the database.
    s.executeUpdate("CREATE TABLE MYLIBRARY.MYTABLE (NAME VARCHAR(20), ID INTEGER)");

    // Run some SQL statements that insert records into the table.
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('RICH', 123)");
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('FRED', 456)");
    s.executeUpdate("INSERT INTO MYLIBRARY.MYTABLE (NAME, ID) VALUES ('MARK', 789)");

    // Run an SQL query on the table.
    ResultSet rs = s.executeQuery("SELECT * FROM MYLIBRARY.MYTABLE");

    // Display all the data in the table.
    while (rs.next()) {
        System.out.println("Employee " + rs.getString(1) + " has ID " + rs.getInt(2));
    }

} catch (SQLException sqle) {
    System.out.println("Database processing has failed.");
    System.out.println("Reason: " + sqle.getMessage());
} finally {
    // Close database resources
    try {
        if (s != null) {
            s.close();
        }
    } catch (SQLException e) {
        System.out.println("Cleanup failed to close Statement.");
    }
}

try {
    if (c != null) {
        c.close();
    }
} catch (SQLException e) {
    System.out.println("Cleanup failed to close Connection.");
}
}
}
}

```

## 例: JAAS HelloWorld

ここで示されている例は、JAAS の HelloWorld をコンパイルして実行するために必要な 3 つのファイルを示しています。

### HelloWorld.java

以下は、ファイル HelloWorld.java のソースです。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
/*
 * =====
 * Licensed Materials - Property of IBM
 *
 * (C) Copyright IBM Corp. 2000 All Rights Reserved.
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * =====
 *
 * File: HelloWorld.java
 */

import java.io.*;
import java.util.*;
import java.security.Principal;
import java.security.PrivilegedAction;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;

/**
 * This SampleLogin application attempts to authenticate a user.
 *
 * If the user successfully authenticates itself,
 * the user name and number of Credentials is displayed.
 *
 * @version 1.1, 09/14/99
 */
public class HelloWorld {

    /**
     * Attempt to authenticate the user.
     */
    public static void main(String[] args) {
        // use the configured LoginModules for the "helloWorld" entry
        LoginContext lc = null;
        try {
            lc = new LoginContext("helloWorld", new MyCallbackHandler());
        } catch (LoginException le) {
            le.printStackTrace();
            System.exit(-1);
        }

        // the user has 3 attempts to authenticate successfully
        int i;
        for (i = 0; i < 3; i++) {
            try {

                // attempt authentication
                lc.login();
            }
        }
    }
}
```

```

        // if we return with no exception, authentication succeeded
        break;
    } catch (AccountExpiredException aee) {
        System.out.println("Your account has expired");
        System.exit(-1);
    } catch (CredentialExpiredException cee) {
        System.out.println("Your credentials have expired.");
        System.exit(-1);
    } catch (FailedLoginException fle) {
        System.out.println("Authentication Failed");
        try {
            Thread.currentThread().sleep(3000);
        } catch (Exception e) {
            // ignore
        }
    } catch (Exception e) {
        System.out.println("Unexpected Exception - unable to continue");
        e.printStackTrace();
        System.exit(-1);
    }
}

// did they fail three times?
if (i == 3) {
    System.out.println("Sorry");
    System.exit(-1);
}

// Look at what Principals we have:
Iterator principalIterator = lc.getSubject().getPrincipals().iterator();
System.out.println("%n%nAuthenticated user has the following Principals:");
while (principalIterator.hasNext()) {
    Principal p = (Principal)principalIterator.next();
    System.out.println("%t" + p.toString());
}

// Look at some Principal-based work:
Subject.doAsPrivileged(lc.getSubject(), new PrivilegedAction() {
    public Object run() {
        System.out.println("%nYour java.home property: "
            +System.getProperty("java.home"));

        System.out.println("%nYour user.home property: "
            +System.getProperty("user.home"));

        File f = new File("foo.txt");
        System.out.print("%nfoo.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        System.out.println("%nOh, by the way ...");

        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignore
        }
        System.out.println("%n%nHello World!\n");
        return null;
    }
});

```

```

        }
        }, null);
        System.exit(0);
    }
}

/**
 * The application must implement the CallbackHandler.
 *
 * This application is text-based. Therefore it displays information
 * to the user using the OutputStreams System.out and System.err,
 * and gathers input from the user using the InputStream, System.in.
 */
class MyCallbackHandler implements CallbackHandler {

    /**
     * Invoke an array of Callbacks.
     *
     * @param callbacks an array of Callback objects which contain
     *     the information requested by an underlying security
     *     service to be retrieved or displayed.
     *
     * @exception java.io.IOException if an input or output error occurs.
     *
     * @exception UnsupportedCallbackException if the implementation of this
     *     method does not support one or more of the Callbacks
     *     specified in the callbacks parameter.
     */
    public void handle(Callback[] callbacks)
        throws IOException, UnsupportedCallbackException {

        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof TextOutputCallback) {

                // display the message according to the specified type
                TextOutputCallback toc = (TextOutputCallback)callbacks[i];
                switch (toc.getMessageType()) {
                    case TextOutputCallback.INFORMATION:
                        System.out.println(toc.getMessage());
                        break;
                    case TextOutputCallback.ERROR:
                        System.out.println("ERROR: " + toc.getMessage());
                        break;
                    case TextOutputCallback.WARNING:
                        System.out.println("WARNING: " + toc.getMessage());
                        break;
                    default:
                        throw new IOException("Unsupported message type: " +
                            toc.getMessageType());
                }

            } else if (callbacks[i] instanceof NameCallback) {

                // prompt the user for a user name
                NameCallback nc = (NameCallback)callbacks[i];

                // ignore the provided defaultName
                System.err.print(nc.getPrompt());
                System.err.flush();
                nc.setName((new BufferedReader
                    (new InputStreamReader(System.in))).readLine());

            } else if (callbacks[i] instanceof PasswordCallback) {

                // prompt the user for sensitive information

```

```

        PasswordCallback pc = (PasswordCallback)callbacks[i];
        System.err.print(pc.getPrompt());
        System.err.flush();
        pc.setPassword(readPassword(System.in));

    } else {
        throw new UnsupportedOperationException
            (callbacks[i], "Unrecognized Callback");
    }
}
}

// Reads user password from given input stream.
private char[] readPassword(InputStream in) throws IOException {

    char[] lineBuffer;
    char[] buf;
    int i;

    buf = lineBuffer = new char[128];

    int room = buf.length;
    int offset = 0;
    int c;

loop: while (true) {
    switch (c = in.read()) {
        case -1:
        case '\n':
            break loop;

        case '\r':
            int c2 = in.read();
            if ((c2 != '\n') && (c2 != -1)) {
                if (!(in instanceof PushbackInputStream)) {
                    in = new PushbackInputStream(in);
                }
                ((PushbackInputStream)in).unread(c2);
            } else
                break loop;

        default:
            if (--room < 0) {
                buf = new char[offset + 128];
                room = buf.length - offset - 1;
                System.arraycopy(lineBuffer, 0, buf, 0, offset);
                Arrays.fill(lineBuffer, ' ');
                lineBuffer = buf;
            }
            buf[offset++] = (char) c;
            break;
    }
}

    if (offset == 0) {
        return null;
    }

    char[] ret = new char[offset];
    System.arraycopy(buf, 0, ret, 0, offset);
    Arrays.fill(buf, ' ');

    return ret;
}
}

```

## HWLoginModule.java

以下は HWLoginModule.java のソースです。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
/*
 * =====
 * Licensed Materials - Property of IBM
 *
 * (C) Copyright IBM Corp. 2000 All Rights Reserved.
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * =====
 *
 * File: HWLoginModule.java
 */

package com.ibm.security;

import java.util.*;
import java.io.IOException;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;
import com.ibm.security.HWPrincipal;

/**
 * This LoginModule authenticates users with a password.
 *
 * This LoginModule only recognizes any user who enters
 * the required password: Go JAAS
 *
 * If the user successfully authenticates itself,
 * a HWPrincipal with the user name
 * is added to the Subject.
 *
 * This LoginModule recognizes the debug option.
 * If set to true in the login Configuration,
 * debug messages are sent to the output stream, System.out.
 *
 * @version 1.1, 09/10/99
 */
public class HWLoginModule implements LoginModule {

    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;

    // configurable option
    private boolean debug = false;

    // the authentication status
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    // user name and password
    private String user name;
    private char[] password;

    private HWPrincipal userPrincipal;

    /**
```



```

* Initialize this LoginModule.
*
* @param subject the Subject to be authenticated.
*
* @param callbackHandler a CallbackHandler for communicating
*       with the end user (prompting for user names and
*       passwords, for example).
*
* @param sharedState shared LoginModule state.
*
* @param options options specified in the login
*       Configuration for this particular
*       LoginModule.
*/
public void initialize(Subject subject, CallbackHandler callbackHandler,
    Map sharedState, Map options) {

    this.subject = subject;
    this.callbackHandler = callbackHandler;
    this.sharedState = sharedState;
    this.options = options;

    // initialize any configured options
    debug = "true".equalsIgnoreCase((String)options.get("debug"));
}

/**
 * Authenticate the user by prompting for a user name and password.
 *
 *
 * @return true in all cases since this LoginModule
 *         should not be ignored.
 *
 * @exception FailedLoginException if the authentication fails.
 *
 * @exception LoginException if this LoginModule
 *         is unable to perform the authentication.
 */
public boolean login() throws LoginException {

    // prompt for a user name and password
    if (callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
            "to garner authentication information from the user");

    Callback[] callbacks = new Callback[2];
    callbacks[0] = new NameCallback("\n\nHWModule user name: ");
    callbacks[1] = new PasswordCallback("HWModule password: ", false);

    try {
        callbackHandler.handle(callbacks);
        user name = ((NameCallback)callbacks[0]).getName();
        char[] tmpPassword = ((PasswordCallback)callbacks[1]).getPassword();
        if (tmpPassword == null) {
            // treat a NULL password as an empty password
            tmpPassword = new char[0];
        }
        password = new char[tmpPassword.length];
        System.arraycopy(tmpPassword, 0,
            password, 0, tmpPassword.length);
        ((PasswordCallback)callbacks[1]).clearPassword();
    } catch (java.io.IOException ioe) {
        throw new LoginException(ioe.toString());
    } catch (UnsupportedCallbackException uce) {
        throw new LoginException("Error: " + uce.getCallback().toString() +
            " not available to garner authentication information " +

```

```

        "from the user");
    }

// print debugging information
if (debug) {
    System.out.println("%n%t[HWLoginModule] " +
        "user entered user name: " +
        user name);
    System.out.print("%t[HWLoginModule] " +
        "user entered password: ");
    for (int i = 0; i < password.length; i++)
        System.out.print(password[i]);
    System.out.println();
}

// verify the password
if (password.length == 7 &&
    password[0] == 'G' &&
    password[1] == 'o' &&
    password[2] == ' ' &&
    password[3] == 'J' &&
    password[4] == 'A' &&
    password[5] == 'A' &&
    password[6] == 'S') {

    // authentication succeeded!!!
    if (debug)
        System.out.println("%n%t[HWLoginModule] " +
            "authentication succeeded");
    succeeded = true;
    return true;
} else {

    // authentication failed -- clean out state
    if (debug)
        System.out.println("%n%t[HWLoginModule] " +
            "authentication failed");
    succeeded = false;
    user name = null;
    for (int i = 0; i < password.length; i++)
        password[i] = ' ';
    password = null;
    throw new FailedLoginException("Password Incorrect");
}
}

/**
 * This method is called if the overall authentication of LoginContext
 * succeeded
 * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
 * succeeded).
 *
 * If this LoginModule authentication attempt
 * succeeded (checked by retrieving the private state saved by the
 * login method), then this method associates a
 * SolarisPrincipal
 * with the Subject located in the
 * LoginModule. If this LoginModule
 * authentication attempt failed, then this method removes
 * any state that was originally saved.
 *
 * @exception LoginException if the commit fails.
 *
 * @return true if the login and commit LoginModule
 * attempts succeeded, or false otherwise.
 */
public boolean commit() throws LoginException {

```

```

if (succeeded == false) {
    return false;
} else {
    // add a Principal (authenticated identity)
    // to the Subject

    // assume the user we authenticated is the HWPrincipal
    userPrincipal = new HWPrincipal(user name);
    final Subject s = subject;
    final HWPrincipal sp = userPrincipal;
    java.security.AccessController.doPrivileged
    (new java.security.PrivilegedAction() {
        public Object run() {
            if (!s.getPrincipals().contains(sp))
                s.getPrincipals().add(sp);
            return null;
        }
    });

    if (debug) {
        System.out.println("¥t[HWLoginModule] " +
            "added HWPrincipal to Subject");
    }

    // in any case, clean out state
    user name = null;
    for (int i = 0; i > password.length; i++)
        password[i] = ' ';
    password = null;

    commitSucceeded = true;
    return true;
}
}

/**
 * This method is called if the overall authentication of LoginContext
 * failed.
 * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
 * did not succeed).
 *
 * If this authentication attempt of LoginModule
 * succeeded (checked by retrieving the private state saved by the
 * login and commit methods),
 * then this method cleans up any state that was originally saved.
 *
 * @exception LoginException if the abort fails.
 *
 * @return false if this login or commit attempt for LoginModule
 *         failed, and true otherwise.
 */
public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        // login succeeded but overall authentication failed
        succeeded = false;
        user name = null;
        if (password != null) {
            for (int i = 0; i > password.length; i++)
                password[i] = ' ';
            password = null;
        }
        userPrincipal = null;
    } else {
        // overall authentication succeeded and commit succeeded,
        // but another commit failed
    }
}

```

```

        logout();
    }
    return true;
}

/**
 * Logout the user.
 *
 * This method removes the HWPrincipal
 * that was added by the commit method.
 *
 * @exception LoginException if the logout fails.
 *
 * @return true in all cases since this LoginModule
 *         should not be ignored.
 */
public boolean logout() throws LoginException {

    final Subject s = subject;
    final HWPrincipal sp = userPrincipal;
    java.security.AccessController.doPrivileged
        (new java.security.PrivilegedAction() {
            public Object run() {
                s.getPrincipals().remove(sp);
                return null;
            }
        });

    succeeded = false;
    succeeded = commitSucceeded;
    user name = null;
    if (password != null) {
        for (int i = 0; i < password.length; i++)
            password[i] = ' ';
        password = null;
    }
    userPrincipal = null;
    return true;
}
}

```

## HWPrincipal.java

以下は HWPrincipal.java のソースです。

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/*
 * =====
 * Licensed Materials - Property of IBM
 *
 * (C) Copyright IBM Corp. 2000 All Rights Reserved.
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 * =====
 *
 * File: HWPrincipal.java
 */

package com.ibm.security;

import java.security.Principal;

/**
 * This class implements the Principal interface

```

```

* and represents a HelloWorld tester.
*
* @version 1.1, 09/10/99
* @author D. Kent Soper
*/
public class HWPrincipal implements Principal, java.io.Serializable {

    private String name;

    /*
     * Create a HWPrincipal with the supplied name.
     */
    public HWPrincipal(String name) {
        if (name == null)
            throw new NullPointerException("illegal null input");

        this.name = name;
    }

    /*
     * Return the name for the HWPrincipal.
     */
    public String getName() {
        return name;
    }

    /*
     * Return a string representation of the HWPrincipal.
     */
    public String toString() {
        return("HWPrincipal: " + name);
    }

    /*
     * Compares the specified Object with the HWPrincipal for equality.
     * Returns true if the given object is also a HWPrincipal and the
     * two HWPrincipals have the same user name.
     */
    public boolean equals(Object o) {
        if (o == null)
            return false;

        if (this == o)
            return true;

        if (!(o instanceof HWPrincipal))
            return false;
        HWPrincipal that = (HWPrincipal)o;

        if (this.getName().equals(that.getName()))
            return true;
        return false;
    }

    /*
     * Return a hash code for the HWPrincipal.
     */
    public int hashCode() {
        return name.hashCode();
    }
}

```

## リンク集

[コードのサンプルに関する特記事項](#)

[コードのサンプルに関する特記事項](#)

## 例: JAAS SampleThreadSubjectLogin

この例は、SampleThreadSubjectLogin クラスの実装を示すものです。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
////////////////////////////////////
//
// 5722-JV1
// (C) Copyright IBM Corp. 2000
//
////////////////////////////////////
//
// File Name:   SampleThreadSubjectLogin.java
//
// Class:       SampleThreadSubjectLogin
//
////////////////////////////////////
//
// CHANGE ACTIVITY:
//
// END CHANGE ACTIVITY
//
////////////////////////////////////

import com.ibm.security.auth.ThreadSubject;

import com.ibm.as400.access.*;

import java.io.*;

import java.util.*;

import java.security.Principal;

import javax.security.auth.*;

import javax.security.auth.callback.*;

import javax.security.auth.login.*;

/**
 * This SampleThreadSubjectLogin application authenticates a single
 * user, swaps the OS thread identity to the authenticated user,
 * and then writes "Hello World" into a privately authorized
 * file, thread.txt, in the user's test directory.
 *
 * The user is requested to enter the user id and password to
 * authenticate.
 *
 * If successful, the user name and number of Credentials
 * are displayed.
 *
 *
 * Setup and run instructions:
 *
 * 1) Create a new user, JAAS13, by invoking
 * "CRTUSRPRF USRPRF(JAAS13) PASSWORD() TEXT('JAAS sample user id')"
 * with *USER class authority.
```

- 2) Allocate a dummy test file, "**yourTestDir**/thread.txt", and privately grant JAAS13 \*RWX authority to it for write access.
- 3) Copy SampleThreadSubjectLogin.java into your test directory.
- 4) Change the current directory to your test directory and compile the java source code.

Enter -

```
strqsh
```

```
cd 'yourTestDir'
```

```
javac -J-Djava.version=1.3
      -classpath /qibm/proddata/os400/java400/ext/jaas13.jar:
              /QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar:.
      -d ./classes
      *.java
```

- 5) Copy threadLogin.config, threadJaas.policy, and threadJava2.policy into your test directory.

- 6) If not already done, add the symbolic link to the extension directory for the jaas13.jar file. The extension class loader should normally load the JAR file.

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
      NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
```

- 7) If not already done to run this sample, add the symbolic link to the extension directory for the jt400.jar and jt400ntv.jar files. This causes these files to be loaded by the extension class loader. The application class loader can also load these files by including them in the CLASSPATH. If these files are loaded from the class path directory, do not add the symbolic link to the extension directory. The jaas13.jar file requires these JAR files for the credential implementation classes which are part of the IBM Toolbox for Java (5722-JC1) Licensed Program Product. (See the IBM Toolbox for Java topic for documentation on the credential classes found in the left frame under Security Classes => Authentication. Select the link to the ProfileTokenCredential class. At the top select 'This Package' for the entire com/ibm/as400/security/auth Java package. Javadoc for the authentication classes can also be found by selecting 'Javadoc' => 'Access Classes' on the left frame. Select 'All Packages' at the top and look for the com.ibm.as400.security.\* packages)

```
ADDLNK OBJ('/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar')
      NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400.jar')
```

```
ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
      NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')
```

```
////////////////////////////////////
IMPORTANT NOTES -
////////////////////////////////////
```

When updating the Java2 policy files for a real application remember to grant the appropriate permissions to the actual locations of the IBM Toolbox for Java JAR files. Even though they are symbolically linked to the extension directories previously listed which are granted java.security.AllPermission in the



`${java.home}/lib/security/java.policy` file, authorization is based on the actual location of the JAR files.

For example, to successfully use the credential classes in IBM Toolbox for Java, you would add the below to your application's Java2 policy file -

```
grant codeBase "file:/QIBM/ProdData/HTTP/Public/jt400/lib/jt400.jar"
{
    permission javax.security.auth.AuthPermission "modifyThreadIdentity";
    permission java.lang.RuntimePermission "loadLibrary.*";
    permission java.lang.RuntimePermission "writeFileDescriptor";
    permission java.lang.RuntimePermission "readFileDescriptor";
}
```

You also need to add these permissions for the application's codeBase since the operations performed by the IBM Toolbox for Java JAR files do not run in privileged mode.

This sample already grants these permissions to all java classes by omitting the codeBase parameter in the threadJava2.policy file.

8) Make sure the Host Servers are started and running. The ProfileTokenCredential classes which reside in IBM Toolbox for Java, i.e. jt400.jar, are used as the credentials that are attached to the authenticated subject by the SampleThreadSubjectLogin.java program. The IBM Toolbox for Java credential classes require access to the Host Servers.

9) Invoke SampleThreadSubjectLogin while signed on as a user that does not have access to '**yourTestDir**/thread.txt'.

10) Start the sample by entering the following CL commands =>

```
CHGCURDIR DIR('yourTestDir')
```

```
JAVA CLASS(SampleThreadSubjectLogin)
CLASSPATH('yourTestDir/classes')
PROP((java.version '1.3')
      (java.security.manager
       (java.security.auth.login.config
        'yourTestDir/threadLogin.config')
       (java.security.policy
        'yourTestDir/threadJava2.policy')
       (java.security.auth.policy
        'yourTestDir/threadJaas.policy'))
```

Enter the user id and password when prompted from step 1.

11) Check **yourTestDir**/thread.txt for the "Hello World" entry.

```
*
**/
```

```
public class SampleThreadSubjectLogin {
/**
 * Attempt to authenticate the user.
 *
 * @param args
 *     Input arguments for this application (ignored).
 */
    public static void main(String[] args) {

        // use the configured LoginModules for the "AS400ToolboxApp" entry
        LoginContext lc = null;
```

```

try {
    // if provided, the same subject is used for multiple login attempts
    lc = new LoginContext("AS400ToolboxApp",
        new Subject(),
        new SampleCBHandler());
} catch (LoginException le) {
    le.printStackTrace();
    System.exit(-1);
}

// the user has 3 attempts to authenticate successfully
int i;
for (i = 0; i < 3; i++) {
    try {

        // attempt authentication
        lc.login();

        // if we return with no exception, authentication succeeded
        break;

    } catch (AccountExpiredException aee) {

        System.out.println("Your account has expired");
        System.exit(-1);

    } catch (CredentialExpiredException cee) {

        System.out.println("Your credentials have expired.");
        System.exit(-1);

    } catch (FailedLoginException fle) {

        System.out.println("Authentication Failed");
        try {
            Thread.currentThread().sleep(3000);
        } catch (Exception e) {
            // ignore
        }

    } catch (Exception e) {

        System.out.println("Unexpected Exception - unable to continue");
        e.printStackTrace();
        System.exit(-1);
    }
}

// did they fail three times?
if (i == 3) {
    System.out.println("Sorry authentication failed");
    System.exit(-1);
}

// display authenticated principals & credentials
System.out.println("Authentication Succeeded");

System.out.println("Principals:");

Iterator itr = lc.getSubject().getPrincipals().iterator();

while (itr.hasNext())
    System.out.println(itr.next());

itr = lc.getSubject().getPrivateCredentials().iterator();

```

```

while (itr.hasNext())
    System.out.println(itr.next());

itr = lc.getSubject().getPublicCredentials().iterator();

while (itr.hasNext())
    System.out.println(itr.next());

// let's do some Principal-based work:
ThreadSubject.doAsPrivileged(lc.getSubject(), new java.security.PrivilegedAction() {
    public Object run() {
        System.out.println("%nYour java.home property: "
            +System.getProperty("java.home"));
        System.out.println("%nYour user.home property: "
            +System.getProperty("user.home"));
        File f = new File("thread.txt");
        System.out.print("%nthread.txt does ");
        if (!f.exists()) System.out.print("not ");
        System.out.println("exist in your current directory");

        try {
            // write "Hello World number x" into thread.txt
            PrintStream ps = new PrintStream(new FileOutputStream("thread.txt", true), true);

            long flen = f.length();
            ps.println("Hello World number " +
                Long.toString(flen/22) +
                "%n");
            ps.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("%nOh, by the way, " + SampleThreadSubjectLogin.getCurrentUser());
        try {
            Thread.currentThread().sleep(2000);
        } catch (Exception e) {
            // ignore
        }
        System.out.println("%n%nHello World!%n");
        return null;
    }
}, null);

System.exit(0);

} // end main()

// Returns the current OS identity for the main thread of the application.
// (This routine uses classes from IBM Toolbox for Java)
// Note - Applications running on a secondary thread cannot use this API to determine the current user.
static public String getCurrentUser() {
    try {
        AS400 localSys = new AS400("localhost", "*CURRENT", "*CURRENT");

        int ccsid = localSys.getCcsid();
        ProgramCall qusrjobi = new ProgramCall(localSys);
        ProgramParameter[] parms = new ProgramParameter[6];

        int rLength = 100;
        parms[0] = new ProgramParameter(rLength);
        parms[1] = new ProgramParameter(new AS400Bin4().toBytes(rLength));
        parms[2] = new ProgramParameter(new AS400Text(8, ccsid, localSys).toBytes("JOBID0600"));
        parms[3] = new ProgramParameter(new AS400Text(26,ccsid, localSys).toBytes("*"));
    }
}

```

```

        parms[4] = new ProgramParameter(new AS400Text(16,ccsid, localSys).toBytes(""));
        parms[5] = new ProgramParameter(new AS400Bin4().toBytes(0));

        qusrjobi.setProgram(QSYSObjectPathName.toPath("QSYS", "QUSRJOB", "PGM"), parms);
        AS400Text uidText = new AS400Text(10, ccsid, localSys);

        // Invoke the QUSRJOB API
        qusrjobi.run();

        byte[] uidBytes = new byte[10];
        System.arraycopy((qusrjobi.getParameterList())[0].getOutputData(), 90, uidBytes, 0, 10);

        return ((String)(uidText.toObject(uidBytes))).trim();
    }

    catch (Exception e) {
        e.printStackTrace();
    }

    return "";
}

} //end SampleThreadSubjectLogin class

```

```

/**
 * A CallbackHandler is passed to underlying security
 * services so that they may interact with the application
 * to retrieve specific authentication data,
 * such as user names and passwords, or to display certain
 * information, such as error and warning messages.
 *
 * CallbackHandlers are implemented in an application
 * and platform-dependent fashion. The implementation decides
 * how to retrieve and display information depending on the
 * Callbacks passed to it.
 *
 * This class provides a sample CallbackHandler for applications
 * running in an i5/OS environment. However, it is not intended
 * to fulfill the requirements of production applications.
 * As indicated, the CallbackHandler is ultimately considered to
 * be application-dependent, as individual applications have
 * unique error checking, data handling, and user
 * interface requirements.
 *
 * The following callbacks are handled:
 *
 * • *
 *
 * • NameCallback *
 *
 * • PasswordCallback *
 *
 * • TextOutputCallback *
 *
 * For simplicity, prompting is handled interactively through
 * standard input and output. However, it is worth noting
 * that when standard input is provided by the console, this
 * approach allows passwords to be viewed as they are
 * typed. This should be avoided in production
 * applications.
 *
 * This CallbackHandler also allows a name and password
 * to be acquired through an alternative mechanism
 * and set directly on the handler to bypass the need for

```

```

* user interaction on the respective Callbacks.
*
*/
class SampleCBHandler implements CallbackHandler {
    private String name_ = null;
    private String password_ = null;
/**
* Constructs a new SampleCBHandler.
*
*/
public SampleCBHandler() {
    this(null, null);
}
/**
* Constructs a new SampleCBHandler.
*
* A name and password can optionally be specified in
* order to bypass the need to prompt for information
* on the respective Callbacks.
*
* @param name
*     The default value for name callbacks. A null
*     value indicates that the user should be
*     prompted for this information. A non-null value
*     cannot be zero length or exceed 10 characters.
*
* @param password
*     The default value for password callbacks. A null
*     value indicates that the user should be
*     prompted for this information. A non-null value
*     cannot be zero length or exceed 10 characters.
*/
public SampleCBHandler(String name, String password) {
    if (name != null)
        if ((name.length()==0) || (name.length())>10))
            throw new IllegalArgumentException("name");
        name_ = name;

    if (password != null)
        if ((password.length()==0) || (password.length())>10))
            throw new IllegalArgumentException("password");
        password_ = password;
}
/**
* Handle the given name callback.
*
* First check to see if a name has been passed in
* on the constructor. If so, assign it to the
* callback and bypass the prompt.
*
* If a value has not been preset, attempt to prompt
* for the name using standard input and output.
*
* @param c
*     The NameCallback.
*
* @exception java.io.IOException
*     If an input or output error occurs.
*/
private void handleNameCallback(NameCallback c) throws IOException {
    // Check for cached value
    if (name_ != null) {
        c.setName(name_);
        return;
    }
    // No preset value; attempt stdin/out

```

```

        c.setName(
            stdIOReadName(c.getPrompt(), 10));
    }
/**
 * Handle the given name callback.
 *
 * First check to see if a password has been passed
 * in on the constructor. If so, assign it to the
 * callback and bypass the prompt.
 *
 * If a value has not been preset, attempt to prompt
 * for the password using standard input and output.
 *
 * @param c
 *     The PasswordCallback.
 *
 * @exception java.io.IOException
 *     If an input or output error occurs.
 */
private void handlePasswordCallback(PasswordCallback c) throws IOException {
    // Check for cached value
    if (password_ != null) {
        c.setPassword(password_.toCharArray());
        return;
    }

    // No preset value; attempt stdin/out
    // Note - Not for production use.
    // Password is not concealed by standard console I/O
    if (c.isEchoOn())
        c.setPassword(
            stdIOReadName(c.getPrompt(), 10).toCharArray());
    else
    {

        // Note - Password is not concealed by standard console I/O
        c.setPassword(stdIOReadName(c.getPrompt(), 10).toCharArray());
    }
}
/**
 * Handle the given text output callback.
 *
 * If the text is informational or a warning,
 * text is written to standard output. If the
 * callback defines an error message, text is
 * written to standard error.
 *
 * @param c
 *     The TextOutputCallback.
 *
 * @exception java.io.IOException
 *     If an input or output error occurs.
 */
private void handleTextOutputCallback(TextOutputCallback c) throws IOException {
    if (c.getMessageType() == TextOutputCallback.ERROR)
        System.err.println(c.getMessage());
    else
        System.out.println(c.getMessage());
}
/**
 * Retrieve or display the information requested in the
 * provided Callbacks.
 *
 * The handle method implementation

```

```

* checks the instance(s) of the Callback
* object(s) passed in to retrieve or display the
* requested information.
*
* @param callbacks
*     An array of Callback objects provided
*     by an underlying security service which contains
*     the information requested to be retrieved or displayed.
*
* @exception java.io.IOException
*     If an input or output error occurs.
*
* @exception UnsupportedOperationException
*     If the implementation of this method does not support
*     one or more of the Callbacks specified in the
*     callbacks parameter.
*/
public void handle(Callback[] callbacks)
    throws IOException, UnsupportedOperationException
{
    for (int i=0; i<callbacks.length; i++) {
        Callback c = callbacks[i];

        if (c instanceof NameCallback)
            handleNameCallback((NameCallback)c);
        else if (c instanceof PasswordCallback)
            handlePasswordCallback((PasswordCallback)c);
        else if (c instanceof TextOutputCallback)
            handleTextOutputCallback((TextOutputCallback)c);
        else
            throw new UnsupportedOperationException
                (callbacks[i]);
    }
}
/**
* Displays the given string using standard output,
* followed by a space to separate from subsequent
* input.
*
* @param prompt
*     The text to display.
*
* @exception IOException
*     If an input or output error occurs.
*/
private void stdIOPrompt(String prompt) throws IOException {
    System.out.print(prompt + ' ');
    System.out.flush();
}
/**
* Reads a String from standard input, stopped at
* maxLength or by a newline.
*
* @param prompt
*     The text to display to standard output immediately
*     prior to reading the requested value.
*
* @param maxLength
*     Maximum length of the String to return.
*
* @return
*     The entered string. The value returned does
*     not contain leading or trailing whitespace
*     and is converted to uppercase.
*

```



```

* @exception IOException
*     If an input or output error occurs.
*
*/
private String stdIOReadName(String prompt, int maxLength) throws IOException {
    stdIOPrompt(prompt);
    String s =
        (new BufferedReader
         (new InputStreamReader(System.in))).readLine().trim();
    if (s.length() < maxLength)
        s = s.substring(0,maxLength);
    return s.toUpperCase();
}

} //end SampleCBHandler class

```

## リンク集

コードのサンプルに関する特記事項

## サンプル: IBM JGSS 非 JAAS クライアント・プログラム

サンプル・クライアント・プログラムの使用に関して詳しくは、サンプル・プログラムのダウンロードおよび実行を参照してください。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// IBM JGSS 1.0 Sample Client Program
```

```

package com.ibm.security.jgss.test;
import org.ietf.jgss.*;
import com.ibm.security.jgss.Debug;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * A JGSS sample client;
 * to be used in conjunction with the JGSS sample server.
 * The client first establishes a context with the server
 * and then sends wrapped message followed by a MIC to the server.
 * The MIC is calculated over the plain text that was wrapped.
 * The client requires to server to authenticate itself
 * (mutual authentication) during context establishment.
 * It also delegates its credentials to the server.
 *
 * It sets the JAVA variable
 * javax.security.auth.useSubjectCredsOnly to false
 * so that JGSS will not acquire credentials through JAAS.
 *
 * The client takes input parameters, and complements it
 * with information from the jgss.ini file; any required input not
 * supplied on the command line is taking from the jgss.ini file.
 *
 * Usage: Client [options]
 *
 * The -? option produces a help message including supported options.
 *
 * This sample client does not use JAAS.
 * The client can be run against the JAAS sample client and server.
 * See {@link JAASClient JAASClient} for a sample client that uses JAAS.
 */

```

```

class Client
{
    private Util testUtil      = null;
    private String myName     = null;
    private GSSName gssName   = null;
    private String serverName = null;
    private int servicePort   = 0;
    private GSSManager mgr    = GSSManager.getInstance();
    private GSSName service   = null;
    private GSSContext context = null;
    private String program    = "Client";
    private String debugPrefix = "Client: ";
    private TCPComms tcp      = null;
    private String data       = null;
    private byte[] dataBytes  = null;
    private String serviceHostname = null;
    private GSSCredential gssCred = null;

    private static Debug debug      = new Debug();

    private static final String usageString =
        "%t[-?] [-d | -n name] [-s serverName]"
        + "%n%t[-h serverHost [:port]] [-p port] [-m msg]"
        + "%n"
        + "%n -?%t%t%thelp; produces this message"
        + "%n -n name%t%tthe client's principal name (without realm)"
        + "%n -s serverName%t%tthe server's principal name (without realm)"
        + "%n -h serverHost[:port]%t%tthe server's hostname"
        + " " (and optional port number)"
        + "%n -p port%t%tthe port on which the server will be listening"
        + "%n -m msg%t%tmessage to send to the server";

    // Caller must call initialize (may need to call processArgs first).
    public Client (String programName) throws Exception
    {
        testUtil = new Util();
        if (programName != null)
        {
            program = programName;
            debugPrefix = programName + ": ";
        }
    }

    // Caller must call initialize (may need to call processArgs first).
    Client (String programName, boolean useSubjectCredsOnly) throws Exception
    {
        this(programName);
        setUseSubjectCredsOnly(useSubjectCredsOnly);
    }

    public Client(GSSCredential myCred,
                 String serverNameWithoutRealm,
                 String serverHostname,
                 int serverPort,
                 String message)
        throws Exception
    {
        testUtil = new Util();

        if (myCred != null)
        {
            gssCred = myCred;
        }
        else
        {
            throw new GSSEException(GSSEException.NO_CRED, 0,

```

```

        "Null input credential");
    }

    init(serverNameWithoutRealm, serverHostname, serverPort, message);
}

void setUseSubjectCredsOnly(boolean useSubjectCredsOnly)
{
    final String subjectOnly = useSubjectCredsOnly ? "true" : "false";
    final String property = "javax.security.auth.useSubjectCredsOnly";

    String temp = (String)java.security.AccessController.doPrivileged(
        new sun.security.action.GetPropertyAction(property));

    if (temp == null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "setting useSubjectCredsOnly property to "
            + useSubjectCredsOnly);

        // Property not set. Set it to the specified value.

        java.security.AccessController.doPrivileged(
            new java.security.PrivilegedAction() {
                public Object run() {
                    System.setProperty(property, subjectOnly);
                    return null;
                }
            });
    }
    else
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "useSubjectCredsOnly property already set "
            + "in JVM to " + temp);
    }
}

private void init(String myNameWithoutRealm,
    String serverNameWithoutRealm,
    String serverHostname,
    int serverPort,
    String message) throws Exception
{
    myName = myNameWithoutRealm;
    init(serverNameWithoutRealm, serverHostname, serverPort, message);
}

private void init(String serverNameWithoutRealm,
    String serverHostname,
    int serverPort,
    String message) throws Exception
{
    // peer's name
    if (serverNameWithoutRealm != null)
    {
        this.serverName = serverNameWithoutRealm;
    }
    else
    {
        this.serverName = testUtil.getDefaultServicePrincipalWithoutRealm();
    }

    // peer's host
    if (serverHostname != null)
    {
        this.serviceHostname = serverHostname;
    }
}

```

```

    }
    else
    {
        this.serviceHostname = testUtil.getDefaultServiceHostname();
    }

    // peer's port
    if (serverPort > 0)
    {
        this.servicePort = serverPort;
    }
    else
    {
        this.servicePort = testUtil.getDefaultServicePort();
    }

    // message for peer
    if (message != null)
    {
        this.data = message;
    }
    else
    {
        this.data = "The quick brown fox jumps over the lazy dog";
    }

    this.dataBytes = this.data.getBytes();

    tcp = new TCPComms(serviceHostname, servicePort);
}

void initialize() throws Exception
{
    Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");

    if (gssCred == null)
    {
        if (myName != null)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "creating GSSName USER_NAME for "
                + myName);

            gssName = mgr.createName(
                myName,
                GSSName.NT_USER_NAME,
                krb5MechanismOid);

            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                + "Canonicalized GSSName=" + gssName);
        }
        else
            gssName = null; // for default credentials

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "creating"
            + ((gssName == null)? " default " : " ")
            + "credential");

        gssCred = mgr.createCredential(
            gssName,
            GSSCredential.DEFAULT_LIFETIME,
            (Oid)null,
            GSSCredential.INITIATE_ONLY);

        if (gssName == null)
        {

```

```

        gssName = gssCred.getName();
        myName = gssName.toString();
        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "default credential principal=" + myName);
    }
}

debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + gssCred);

debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
    + "creating canonicalized GSSName for serverName " + serverName);

service = mgr.createName(serverName,
    GSSName.NT_HOSTBASED_SERVICE,
    krb5MechanismOid);

debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
    + "Canonicalized server name = " + service);

debug.out(Debug.OPTS_CAT_APPLICATION,
    debugPrefix + "Raw data=" + data);
}

```

```

void establishContext(BitSet flags) throws Exception
{
    try {
        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "creating GSScontext");

        Oid defaultMech = null;
        context = mgr.createContext(service, defaultMech, gssCred,
            GSSContext.INDEFINITE_LIFETIME);

        if (flags != null)
        {
            if (flags.get(Util.CONTEXT_OPTS_MUTUAL))
            {
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                    + "requesting mutualAuthn");

                context.requestMutualAuth(true);
            }

            if (flags.get(Util.CONTEXT_OPTS_INTEG))
            {
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                    + "requesting integrity");

                context.requestInteg(true);
            }

            if (flags.get(Util.CONTEXT_OPTS_CONF))
            {
                context.requestConf(true);
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                    + "requesting confidentiality");
            }

            if (flags.get(Util.CONTEXT_OPTS_DELEG))
            {
                context.requestCredDeleg(true);
                debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                    + "requesting delegation");
            }
        }
    }
}

```

```

    }

    if (flags.get(Util.CONTEXT_OPTS_REPLAY))
    {
        context.requestReplayDet(true);
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "requesting replay detection");
    }

    if (flags.get(Util.CONTEXT_OPTS_SEQ))
    {
        context.requestSequenceDet(true);
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "requesting out-of-sequence detection");
    }
    // Add more later!
}

byte[] response = null;
byte[] request = null;
int len = 0;
boolean done = false;
do {
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
        + "Calling initSecContext");

    request = context.initSecContext(response, 0, len);

    if (request != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "Sending initial context token");

        tcp.send(request);
    }
    done = context.isEstablished();

    if (!done)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "Receiving response token");

        byte[] temp = tcp.receive();
        response = temp;
        len = response.length;
    }
} while(!done);

debug.out(Debug.OPTS_CAT_APPLICATION,
    debugPrefix + "context established with acceptor");

} catch (Exception exc) {
    exc.printStackTrace();
    throw exc;
}
}

void doMIC() throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "generating MIC");
    byte[] mic = context.getMIC(dataBytes, 0, dataBytes.length, null);

    if (mic != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "sending MIC");
        tcp.send(mic);
    }
}

```

```

        else
            debug.out(Debug.OPTS_CAT_APPLICATION,
                      debugPrefix + "getMIC Failed");
    }

void doWrap() throws Exception
{
    MessageProp mp = new MessageProp(true);
    mp.setPrivacy(context.getConfState());

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "wrapping message");

    byte[] wrapped = context.wrap(dataBytes, 0, dataBytes.length, mp);

    if (wrapped != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION,
                  debugPrefix + "sending wrapped message");

        tcp.send(wrapped);
    }
    else
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "wrap Failed");
}

void printUsage()
{
    System.out.println(program + usageString);
}

void processArgs(String[] args) throws Exception
{
    String port          = null;
    String myName        = null;
    int servicePort      = 0;
    String serviceHostname = null;

    String sHost = null;
    String msg = null;

    GetOptions options = new GetOptions(args, "?h:p:m:n:s:");
    int ch = -1;
    while ((ch = options.getopt()) != options.optEOF)
    {
        switch(ch)
        {
            case '?':
                printUsage();
                System.exit(1);

            case 'h':
                if (sHost == null)
                {
                    sHost = options.optArgGet();
                    int p = sHost.indexOf(':');
                    if (p != -1)
                    {
                        String temp1 = sHost.substring(0, p);
                        if (port == null)
                            port = sHost.substring(p+1, sHost.length()).trim();
                        sHost = temp1;
                    }
                }
                continue;

            case 'p':
                if (port == null)

```



```

        port = options.optArgGet();
        continue;

    case 'm':
        if (msg == null)
            msg = options.optArgGet();
        continue;

    case 'n':
        if (myName == null)
            myName = options.optArgGet();
        continue;

    case 's':
        if (serverName == null)
            serverName = options.optArgGet();
        continue;
    }
}

if ((port != null) && (port.length() > 0))
{
    int p = -1;
    try {
        p = Integer.parseInt(port);
    } catch (Exception exc) {
        System.out.println("Bad port input: "+port);
    }

    if (p != -1)
        servicePort = p;
}

if ((sHost != null) && (sHost.length() > 0)) {
    serviceHostname = sHost;
}

init(myName, serverName, serviceHostname, servicePort, msg);
}

void interactWithAcceptor(BitSet flags) throws Exception
{
    establishContext(flags);
    doWrap();
    doMIC();
}

void interactWithAcceptor() throws Exception
{
    BitSet flags = new BitSet();
    flags.set(Util.CONTEXT_OPTS_MUTUAL);
    flags.set(Util.CONTEXT_OPTS_CONF);
    flags.set(Util.CONTEXT_OPTS_INTEG);
    flags.set(Util.CONTEXT_OPTS_DELEG);
    interactWithAcceptor(flags);
}

void dispose() throws Exception
{
    if (tcp != null)
    {
        tcp.close();
    }
}

public static void main(String args[]) throws Exception
{

```

```

System.out.println(debug.toString()); // XXXXXXX
String programName = "Client";
Client client = null;
try {
    client = new Client(programName,
                        false); // don't use Subject creds.
    client.processArgs(args);
    client.initialize();
    client.interactWithAcceptor();
} catch (Exception exc) {
    debug.out(Debug.OPTS_CAT_APPLICATION,
              programName + " Exception: " + exc.toString());
    exc.printStackTrace();
    throw exc;
} finally {
    try {
        if (client != null)
            client.dispose();
    } catch (Exception exc) {}
}

debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": done");
}
}

```

## リンク集

サンプル・プログラムのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

## サンプル: IBM JGSS 非 JAAS サーバー・プログラム

サンプル・サーバー・プログラムの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// IBM JGSS 1.0 Sample Server Program
```

```
package com.ibm.security.jgss.test;
```

```
import org.ietf.jgss.*;
```

```
import com.ibm.security.jgss.Debug;
```

```
import java.io.*;
```

```
import java.net.*;
```

```
import java.util.*;
```

```
/**
```

```
 * A JGSS sample server; to be used in conjunction with a JGSS sample client.
```

```
 *
```

```
 * It continuously listens for client connections,
```

```
 * spawning a thread to service an incoming connection.
```

```
 * It is capable of running multiple threads concurrently.
```

```
 * In other words, it can service multiple clients concurrently.
```

```
 *
```

```
 * Each thread first establishes a context with the client
```

```
 * and then waits for a wrapped message followed by a MIC.
```

```
 * It assumes that the client calculated the MIC over the plain
```

```
 * text wrapped by the client.
```

```
 *
```

```
 * If the client delegates its credential to the server, the delegated
```

```
 * credential is used to communicate with a secondary server.
```

```

*
* Also, the server can be started to act as a client as well as
* a server (using the -b option). In this case, the first
* thread spawned by the server uses the server principal's own credential
* to communicate with the secondary server.
*
* The secondary server must have been started prior to the (primary) server
* initiating contact with it (the secondary server).
* In communicating with the secondary server, the primary server acts as
* a JGSS initiator (i.e., client), establishing a context and engaging in
* wrap and MIC per-message exchanges with the secondary server.
*
* The server takes input parameters, and complements it
* with information from the jgss.ini file; any required input not
* supplied on the command line is taken from the jgss.ini file.
* Built-in defaults are used if there is no jgss.ini file or if a particular
* variable is not specified in the ini file.
*
* Usage: Server [options]
*
* The -? option produces a help message including supported options.
*
* This sample server does not use JAAS.
* It sets the JAVA variable
* javax.security.auth.useSubjectCredsOnly to false
* so that JGSS will not acquire credentials through JAAS.
* The server can be run against the JAAS sample clients and servers.
* See {@link JAASServer JAASServer} for a sample server that uses JAAS.
*/

```

```

class Server implements Runnable
{

```

```

    /*
    * NOTES:
    * This class, Server, is expected to be run in concurrent
    * multiple threads. The static variables consist of variables
    * set from command-line arguments and variables (such as
    * the server's own credentials, gssCred) that are set once during
    * during initialization. These variables do not change
    * once set and are shared between all running threads.
    *
    * The only static variable that is changed after being set initially
    * is the variable 'beenInitiator' which is set 'true'
    * by the first thread to run the server as initiator using
    * the server's own creds. This ensures the server is run as an initiator
    * once only. Querying and modifying 'beenInitiator' is synchronized
    * between the threads.
    *
    * The variable 'tcp' is non-static and is set per thread
    * to represent the socket on which the client being serviced
    * by the thread connected.
    */

```

```

    private static Util testUtil          = null;
    private static int myPort             = 0;
    private static Debug debug            = new Debug();
    private static String myName          = null;
    private static GSSCredential gssCred  = null;
    private static String serviceNameNoRealm = null;
    private static String serviceHost     = null;
    private static int servicePort        = 0;
    private static String serviceMsg      = null;
    private static GSSManager mgr         = null;
    private static GSSName gssName        = null;
    private static String program          = "Server";
    private static boolean clientServer    = false;
    private static boolean primaryServer   = true;

```



```

// Property not set. Set it to the specified value.

java.security.AccessController.doPrivileged(
    new java.security.PrivilegedAction() {
        public Object run() {
            System.setProperty(property, subjectOnly);
            return null;
        }
    });
}
else
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
        + "useSubjectCredsOnly property already set "
        + "in JVM to " + temp);
}
}

private void init(boolean primary,
    String myNameWithoutRealm,
    int port,
    String serverNameWithoutRealm,
    String serverHostname,
    int serverPort,
    String message,
    boolean clientServer)
    throws Exception
{
    primaryServer = primary;
    this.clientServer = clientServer;

    myName = myNameWithoutRealm;

    // my port
    if (port > 0)
    {
        myPort = port;
    }
    else if (primary)
    {
        myPort = testUtil.getDefaultServicePort();
    }
    else
    {
        myPort = testUtil.getDefaultService2Port();
    }

    if (primary)
    {
        ///// peer's name
        if (serverNameWithoutRealm != null)
        {
            serviceNameNoRealm = serverNameWithoutRealm;
        }
        else
        {
            serviceNameNoRealm =
                testUtil.getDefaultService2PrincipalWithoutRealm();
        }

        // peer's host
        if (serverHostname != null)
        {
            if (serverHostname.equalsIgnoreCase("localhost"))
            {
                serverHostname = InetAddress.getLocalHost().getHostName();
            }
        }
    }
}

```

```

        }
        serviceHost = serverHostname;
    }
    else
    {
        serviceHost = testUtil.getDefaultService2Hostname();
    }

    // peer's port
    if (serverPort > 0)
    {
        servicePort = serverPort;
    }
    else
    {
        servicePort = testUtil.getDefaultService2Port();
    }

    // message for peer
    if (message != null)
    {
        serviceMsg = message;
    }
    else
    {
        serviceMsg = "Hi there! I am a server."
            + "But I can be a client, too";
    }
}

String temp = debugPrefix + "details"
    + "\n\tPrimary:\t" + primary
    + "\n\tName:\t\t" + myName
    + "\n\tPort:\t\t" + myPort
    + "\n\tClient+server:\t" + clientServer;
if (primary)
{
    temp += "\n\tOther Server:"
        + "\n\t\tName:\t" + serviceNameNoRealm
        + "\n\t\tHost:\t" + serviceHost
        + "\n\t\tPort:\t" + servicePort
        + "\n\t\tMsg:\t" + serviceMsg;
}

debug.out(Debug.OPTS_CAT_APPLICATION, temp);
}

void initialize() throws GSSEException
{
    debug.out(Debug.OPTS_CAT_APPLICATION,
        debugPrefix + "creating GSSManager");

    mgr = GSSManager.getInstance();

    int usage = clientServer ? GSSCredential.INITIATE_AND_ACCEPT
        : GSSCredential.ACCEPT_ONLY;

    if (myName != null)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "creating GSSName for " + myName);

        gssName = mgr.createName(myName,
            GSSName.NT_HOSTBASED_SERVICE);
    }
}

```

```

        Oid krb5MechanismOid = new Oid("1.2.840.113554.1.2.2");
        gssName.canonicalize(krb5MechanismOid);

        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "Canonicalized GSSName=" + gssName);
    }
    else
        gssName = null;

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "creating"
        + ((gssName == null)? " default " : " ")
        + "credential");

    gssCred = mgr.createCredential(
        gssName, GSSCredential.DEFAULT_LIFETIME,
        (Oid)null, usage);
    if (gssName == null)
    {
        gssName = gssCred.getName();
        myName = gssName.toString();

        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "default credential principal=" + myName);
    }
}

```

```

void processArgs(String[] args) throws Exception
{
    String port    = null;
    String name    = null;
    int  iport     = 0;

    String sport   = null;
    int  isport    = 0;
    String sname   = null;
    String shost   = null;
    String smessage = null;

    boolean primary = true;
    String status   = null;

    boolean defaultPrinc = false;
    boolean clientServer = false;

    GetOptions options = new GetOptions(args, "?#:p:n:P:s:h:m:b");
    int ch = -1;
    while ((ch = options.getopt()) != options.optEOF)
    {
        switch(ch)
        {
            case '?':
                printUsage();
                System.exit(1);

            case '#':
                if (status == null)
                    status = options.optArgGet();
                continue;

            case 'p':
                if (port == null)
                    port = options.optArgGet();
                continue;

            case 'n':

```



```

        if (name == null)
            name = options.optArgGet();
        continue;

    case 'b':
        clientServer = true;
        continue;

    /////// The other server

    case 'P':
        if (sport == null)
            sport = options.optArgGet();
        continue;

    case 'm':
        if (smessage == null)
            smessage = options.optArgGet();
        continue;

    case 's':
        if (sname == null)
            sname = options.optArgGet();
        continue;

    case 'h':
        if (shost == null)
        {
            shost = options.optArgGet();
            int p = shost.indexOf(':');
            if (p != -1)
            {
                String temp1 = shost.substring(0, p);
                if (sport == null)
                    sport = shost.substring
                        (p+1, shost.length()).trim();
                shost = temp1;
            }
        }
        continue;
    }
}

if (defaultPrinc && (name != null))
{
    System.out.println(
        "ERROR: '-d' and '-n ' options are mutually exclusive");
    printUsage();
    System.exit(1);
}

if (status != null)
{
    int p = -1;
    try {
        p = Integer.parseInt(status);
    } catch (Exception exc) {
        System.out.println( "Bad status input: "+status);
    }

    if (p != -1)
    {
        primary = (p == 1);
    }
}

if (port != null)

```

```

    {
        int p = -1;
        try {
            p = Integer.parseInt(port);
        } catch (Exception exc) {
            System.out.println( "Bad port input: "+port);
        }
        if (p != -1)
            iport = p;
    }

    if (sport != null)
    {
        int p = -1;
        try {
            p = Integer.parseInt(sport);
        } catch (Exception exc) {
            System.out.println( "Bad server port input: "+port);
        }
        if (p != -1)
            isport = p;
    }

    init(primary, // first or second server
        name, // my name
        iport, // my port
        sname, // other server's name
        shost, // other server's hostname
        isport, // other server's port
        smessage, // msg for other server
        clientServer); // whether to run as initiator with own creds
}

void processRequests() throws Exception
{
    ServerSocket ssocket = null;
    Server server = null;
    try {
        ssocket = new ServerSocket(myPort);
        do {
            debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "listening on port " + myPort + " ...");
            Socket csocket = ssocket.accept();

            debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "incoming connection on " + csocket);

            server = new Server(csocket); // set client socket per thread
            Thread thread = new Thread(server);
            thread.start();
            if (!thread.isAlive())
                server.dispose(); // close the client socket
        } while(true);
    } catch (Exception exc) {
        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "*** ERROR processing requests ***");
        exc.printStackTrace();
    } finally {
        try {
            if (ssocket != null)
                ssocket.close(); // close the server socket
            if (server != null)
                server.dispose(); // close the client socket
        } catch (Exception exc) {}
    }
}
}

```

```

void dispose()
{
    try {
        if (tcp != null)
        {
            tcp.close();
            tcp = null;
        }
    } catch (Exception exc) {}
}

boolean establishContext(GSSContext context) throws Exception
{
    byte[] response = null;
    byte[] request = null;

    debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "establishing context");

    do {
        request = tcp.receive();
        if (request == null || request.length == 0)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                      + "Received no data; perhaps client disconnected");

            return false;
        }

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "accepting");
        if ((response = context.acceptSecContext
            (request, 0, request.length)) != null)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION,
                      debugPrefix + "sending response");
            tcp.send(response);
        }
    } while(!context.isEstablished());

    debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "context established - " + context);

    return true;
}

byte[] unwrap(GSSContext context, byte[] msg) throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "unwrapping");

    MessageProp mp = new MessageProp(true);
    byte[] unwrappedMsg = context.unwrap(msg, 0, msg.length, mp);

    debug.out(Debug.OPTS_CAT_APPLICATION,
               debugPrefix + "unwrapped msg is:");
    debug.out(Debug.OPTS_CAT_APPLICATION, unwrappedMsg);

    return unwrappedMsg;
}

void verifyMIC (GSSContext context, byte[] mic, byte[] raw) throws Exception
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "verifying MIC");

    MessageProp mp = new MessageProp(true);
    context.verifyMIC(mic, 0, mic.length, raw, 0, raw.length, mp);

    debug.out(Debug.OPTS_CAT_APPLICATION,

```

```

        debugPrefix + "successfully verified MIC");
    }

void useDelegatedCred(GSSContext context) throws Exception
{
    GSSCredential delCred = context.getDelegCred();
    if (delCred != null)
    {
        if (primaryServer)
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                "Primary server received delegated cred; using it");
            runAsInitiator(delCred); // using delegated creds
        }
        else
        {
            debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
                "Non-primary server received delegated cred; "
                + "ignoring it");
        }
    }
}
else
{
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
        "ERROR: null delegated cred");
}
}

public void run()
{
    byte[] response = null;
    byte[] request = null;
    boolean unwrapped = false;
    GSSContext context = null;

    try {
        Thread currentThread = Thread.currentThread();
        String threadName = currentThread.getName();

        debugPrefix = program + " " + threadName + ": ";

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "servicing client ...");

        debug.out(Debug.OPTS_CAT_APPLICATION,
            debugPrefix + "creating GSSContext");

        context = mgr.createContext(gssCred);

        // First establish context with the initiator.
        if (!establishContext(context))
            return;

        // Then process messages from the initiator.
        // We expect to receive a wrapped message followed by a MIC.
        // The MIC should have been calculated over the plain
        // text that we received wrapped.
        // Use delegated creds if any.
        // Then run as initiator using own creds if necessary; only
        // the first thread does this.

        do {
            debug.out(Debug.OPTS_CAT_APPLICATION,
                debugPrefix + "receiving per-message request");

            request = tcp.receive();

```

```

    if (request == null || request.length == 0)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
            + "Received no data; perhaps client disconnected");

        return;
    }

    // Expect wrapped message first.
    if (!unwrapped)
    {
        response = unwrap(context, request);
        unwrapped = true;
        continue; // get next request
    }

    // Followed by a MIC.
    verifyMIC(context, request, response);

    // Impersonate the initiator if it delegated its creds to us.
    if (context.getCredDelegState())
        useDelegatedCred(context);

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
        + "clientServer=" + clientServer
        + ", beenInitiator=" + beenInitiator);

    // If necessary, run as initiator using our own creds.
    if (clientServer)
        runAsInitiatorOnce(currentThread);

    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "done");
    return;
} while(true);
} catch (Exception exc) {
    debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix + "ERROR");
    exc.printStackTrace();

    // Squelch per-thread exceptions so we don't bring
    // the server down because of exceptions in
    // individual threads.
    return;
} finally {
    if (context != null)
    {
        try {
            context.dispose();
        } catch (Exception exc) {}
    }
}
}

synchronized void runAsInitiatorOnce(Thread thread)
    throws InterruptedException
{
    if (!beenInitiator)
    {
        // set flag true early to prevent subsequent threads
        // from attempting to runAsInitiator.
        beenInitiator = true;

        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix +
            "About to run as initiator with own creds ...");

        //thread.sleep(30*1000, 0);
    }
}

```

```

        runAsInitiator();
    }
}

void runAsInitiator(GSSCredential cred)
{
    Client client = null;
    try {
        client = new Client(cred,
                           serviceNameNoRealm,
                           serviceHost,
                           servicePort,
                           serviceMsg);

        client.initialize();

        BitSet flags = new BitSet();
        flags.set(Util.CONTEXT_OPTS_MUTUAL);
        flags.set(Util.CONTEXT_OPTS_CONF);
        flags.set(Util.CONTEXT_OPTS_INTEG);

        client.interactWithAcceptor(flags);
    } catch (Exception exc) {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                 + "Exception running as initiator");

        exc.printStackTrace();
    } finally {
        try {
            client.dispose();
        } catch (Exception exc) {}
    }
}

void runAsInitiator()
{
    if (clientServer)
    {
        debug.out(Debug.OPTS_CAT_APPLICATION,
                 debugPrefix + "running as initiator with own creds");

        runAsInitiator(gssCred); // use own creds;
    }
    else
    {
        debug.out(Debug.OPTS_CAT_APPLICATION, debugPrefix
                 + "Cannot run as initiator with own creds "
                 + "%nbecause not running as both initiator and acceptor.");
    }
}

void printUsage()
{
    System.out.println(program + usageString);
}

public static void main(String[] args) throws Exception
{
    System.out.println(debug.toString()); // XXXXXXXX
    String programName = "Server";
    try {
        Server server = new Server(programName,
                                   false); // don't use creds from Subject
        server.processArgs(args);
        server.initialize();
    }
}

```

```

        server.processRequests();
    } catch (Exception exc) {
        debug.out(Debug.OPTS_CAT_APPLICATION, programName + ": EXCEPTION");
        exc.printStackTrace();
        throw exc;
    }
}
}
}

```

## リンク集

IBM JGSS サンプルのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

## サンプル: IBM JGSS JAAS 使用可能クライアント・プログラム

サンプル・クライアント・プログラムの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

// IBM Java GSS 1.0 sample JAAS-enabled client program

package com.ibm.security.jgss.test;
import com.ibm.security.jgss.Debug;
import com.ibm.security.auth.callback.Krb5CallbackHandler;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.security.PrivilegedExceptionAction;

/**
 * A Java GSS sample client that uses JAAS.
 *
 * It does a JAAS login and operates within the JAAS login context so created.
 *
 * It does not set the JAVA variable
 * javax.security.auth.useSubjectCredsOnly, leaving
 * the variable to default to true
 * so that Java GSS acquires credentials from the JAAS Subject
 * associated with login context (created by the client).
 *
 * The JAASClient is equivalent to its superclass {@link Client Client}
 * in all other respects, and it
 * can be run against the non-JAAS sample clients and servers.
 */

class JAASClient extends Client
{
    JAASClient(String programName) throws Exception
    {
        // Do not set useSubjectCredsOnly. Set only the program name.
        // useSubjectCredsOnly default to "true" if not set.
        super(programName);
    }

    static class JAASClientAction implements PrivilegedExceptionAction
    {
        private JAASClient client;

        public JAASClientAction(JAASClient client)
        {
            this.client = client;
        }
    }
}

```



```

    }

    public Object run () throws Exception
    {
        client.initialize();
        client.interactWithAcceptor();
        return null;
    }
}

public static void main(String args[]) throws Exception
{
    String programName = "JAASClient";
    JAASClient client = null;
    Debug dbg = new Debug();

    System.out.println(dbg.toString()); // XXXXXXX

    try {
        client = new JAASClient(programName);//use Subject creds
        client.processArgs(args);

        LoginContext loginCtxt = new LoginContext("JAASClient",
            new Krb5CallbackHandler());

        loginCtxt.login();

        dbg.out(Debug.OPTS_CAT_APPLICATION,
            programName + ": Kerberos login OK");

        Subject subject = loginCtxt.getSubject();

        PrivilegedExceptionAction jaasClientAction
            = new JAASClientAction(client);

        Subject.doAsPrivileged(subject, jaasClientAction, null);

    } catch (Exception exc) {
        dbg.out(Debug.OPTS_CAT_APPLICATION,
            programName + " Exception: " + exc.toString());
        exc.printStackTrace();
        throw exc;
    } finally {
        try {
            if (client != null)
                client.dispose();
        } catch (Exception exc) {}
    }

    dbg.out(Debug.OPTS_CAT_APPLICATION,
        programName + ": Done ...");
}
}

```

## リンク集

IBM JGSS サンプルのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

## サンプル: IBM JGSS JAAS 使用可能サーバー・プログラム

サンプル・サーバー・プログラムの使用に関して詳しくは、IBM JGSS サンプルのダウンロードおよび実行を参照してください。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
// IBM Java GSS 1.0 sample JAAS-enabled server program

package com.ibm.security.jgss.test;
import com.ibm.security.jgss.Debug;
import com.ibm.security.auth.callback.Krb5CallbackHandler;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.security.PrivilegedExceptionAction;

/**
 * A Java GSS sample server that uses JAAS.
 *
 * It does a JAAS login and operates within the JAAS login context so created.
 *
 * It does not set the JAVA variable
 * javax.security.auth.useSubjectCredsOnly, leaving
 * the variable to default to true
 * so that Java GSS acquires credentials from the JAAS Subject
 * associated with login context (created by the server).
 *
 * The JAASServer is equivalent to its superclass {@link Server Server}
 * in all other respects, and it
 * can be run against the non-JAAS sample clients and servers.
 */

class JAASServer extends Server
{
    JAASServer(String programName) throws Exception
    {
        super(programName);
    }

    static class JAASServerAction implements PrivilegedExceptionAction
    {
        private JAASServer server = null;

        JAASServerAction(JAASServer server)
        {
            this.server = server;
        }

        public Object run() throws Exception
        {
            server.initialize();
            server.processRequests();

            return null;
        }
    }

    public static void main(String[] args) throws Exception
    {
        String programName    = "JAASServer";
        Debug dbg              = new Debug();

        System.out.println(dbg.toString()); // XXXXXXXX

        try {
            // Do not set useSubjectCredsOnly.
            // useSubjectCredsOnly defaults to "true" if not set.

            JAASServer server = new JAASServer(programName);

            server.processArgs(args);
        }
    }
}
```

```

LoginContext loginCtxt = new LoginContext(programName,
                                           new Krb5CallbackHandler());

dbg.out(Debug.OPTS_CAT_APPLICATION, programName + ": Login in ...");

loginCtxt.login();

dbg.out(Debug.OPTS_CAT_APPLICATION, programName +
        ": Login successful");

Subject subject = loginCtxt.getSubject();

JAASServerAction serverAction = new JAASServerAction(server);

Subject.doAsPrivileged(subject, serverAction, null);
} catch (Exception exc) {
    dbg.out(Debug.OPTS_CAT_APPLICATION, programName + " EXCEPTION");
    exc.printStackTrace();
    throw exc;
}
}
}

```

## リンク集

IBM JGSS サンプルのダウンロードおよび実行

このトピックでは、サンプル javadoc のダウンロードおよび実行の手順を記載しています。

コードのサンプルに関する特記事項

## 例: IBM Java Secure Sockets Extension

JSSE の例では、クライアントおよびサーバーがネイティブ iSeries JSSE プロバイダーを使用して、安全な通信を可能にするコンテキストを作成する方法を示しています。

注: いずれの例でも、java.security ファイルの指定するプロパティーにかかわらず、ネイティブ iSeries JSSE プロバイダーを使用します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

304 ページの『例: SSLContext オブジェクトを使用する SSL クライアント』

このクライアント・プログラムの例では、"MY\_CLIENT\_APP" アプリケーション ID を使用するために初期化を行う、SSLContext オブジェクトを使用します。このプログラムでは、java.security ファイルにおける指定の有無にかかわらず、ネイティブ iSeries インプリメンテーションを使用します。

306 ページの『例: SSLContext オブジェクトを使用する SSL サーバー』

以下のサーバー・プログラムは、過去に作成された鍵ストア・ファイルによって初期化を行う、SSLContext オブジェクトを使用します。鍵ストア・ファイルの名前は /home/keystore.file であり、鍵ストア・パスワードは password です。

このサンプル・プログラムは、IbmISeriesKeyStore オブジェクトを作成するために鍵ストア・ファイルを必要とします。鍵ストア・オブジェクトはアプリケーション ID として MY\_SERVER\_APP を指定しなければなりません。

鍵ストア・ファイルを作成するために、以下のコマンドのいずれかを使用することができます。

- Qshell コマンド・プロンプトから:

```
java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
-storepass password -appid MY_SERVER_APP
```

Qshell で Java コマンドを使用することについては、iSeries Information Center 内の Qshell を参照してください。

- iSeries コマンド・プロンプトから:

```
RUNJAVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
'/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
```

## 例: java.lang.Runtime.exec() を使用して CL プログラムを呼び出す

この例では、Java プログラムから CL プログラムを実行する方法を示します。この例では、Java クラス CallCLPgm が CL プログラムを実行します。

CL プログラムは Java プログラム表示 (DSPJVAPGM) コマンドを使用して、Hello クラス・ファイルと関連しているプログラムを表示します。この例では、CL プログラムはコンパイル済みであり、JAVSAMPLIB と呼ばれるライブラリーに存在していることが前提となっています。CL プログラムからの出力は、QSYSPRT スプール・ファイルにあります。

Java プログラムから CL コマンドを呼び出す方法の例については、CL コマンドを呼び出すを参照してください。

**注:** JAVSAMPLIB は、IBM Developer Kit ライセンス・プログラム (LP) (番号 5722-JV1) のインストール・プロセスの一部としては作成されません。このライブラリーは明示的に作成する必要があります。

### 例 1: CallCLPgm クラス

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallCLPgm
{
    public static void main(String[] args)
    {
        try
        {
            Process theProcess =
                Runtime.getRuntime().exec("/QSYS.LIB/JAVSAMPLIB.LIB/DSPJVA.PGM");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }
    } // end main() method
} // end class
```

### 例 2: Java CL プログラムの表示

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
OUTPUT(*PRINT)
ENDPGM
```

背景情報については、java.lang.Runtime.exec() を使用するを参照してください。

## 例: java.lang.Runtime.exec() を使用して CL コマンドを呼び出す

この例では、Java プログラムから制御言語 (CL) コマンドを実行する方法を示します。

この例では、Java クラスが CL コマンドを実行します。CL コマンドは、「Java プログラムの表示 (DSPJVAPGM)」CL コマンドを使用して、Hello クラス・ファイルと関連しているプログラムを表示します。CL コマンドからの出力は、QSYSPRT スプール・ファイルにあります。

os400.runtime.exec システム・プロパティを (デフォルトの) EXEC に設定する場合、Runtime.getRuntime().exec() 関数に渡すコマンドは次のフォーマットになります。

```
Runtime.getRuntime().exec("system CLCOMMAND");
```

ここで、*CLCOMMAND* は、実行しようとしている CL コマンドです。

注: os400.runtime.exec を QSHELL に設定する場合は、スラッシュと引用符 (¥) を追加する必要があります。たとえば、上のコマンドならば、次のようになります。

```
Runtime.getRuntime().exec("system ¥"CLCOMMAND¥");
```

os400.runtime.exec と、それが java.lang.Runtime.exec() の使用に与える影響については、以下のページを参照してください。

java.lang.Runtime.exec() を使用する

Java システム・プロパティのリスト

## 例: CL コマンドを呼び出すためのクラス

以下のコードでは、os400.runtime.exec システム・プロパティにデフォルト値の EXEC を使用していることを想定しています。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallCLCom
{
    public static void main(String[] args)
    {
        try
        {
            Process theProcess =
                Runtime.getRuntime().exec("system DSPJVAPGM CLSF('/com/ibm/as400/system/Hello.class')
                OUTPUT(*PRINT)");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }
    } // end main() method
} // end class
```

背景情報については、java.lang.Runtime.exec() を使用するを参照してください。

## 例: java.lang.Runtime.exec() を使用して別の Java プログラムを呼び出す

この例では、java.lang.Runtime.exec() を使用して別の Java プログラムを呼び出す方法を示します。このクラスは、IBM Developer Kit for Java の一部として配布される Hello プログラムを呼び出します。Hello クラスが System.out に書き込みを行うときに、このプログラムは、ストリームへのハンドルを取得し、そこから読み取りを行うことができます。

注: プログラムを呼び出すには、Qshell インタープリターを使用します。

### 例 1: CallHelloPgm クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallHelloPgm
{
    public static void main(String args[])
    {
        Process theProcess = null;
        BufferedReader inStream = null;

        System.out.println("CallHelloPgm.main() invoked");

        // call the Hello class
        try
        {
            theProcess = Runtime.getRuntime().exec("java com.ibm.as400.system.Hello");
        }
        catch(IOException e)
        {
            System.err.println("Error on exec() method");
            e.printStackTrace();
        }

        // read from the called program's standard output stream
        try
        {
            inStream = new BufferedReader(
                new InputStreamReader( theProcess.getInputStream() ));
            System.out.println(inStream.readLine());
        }
        catch(IOException e)
        {
            System.err.println("Error on inStream.readLine()");
            e.printStackTrace();
        }

        } // end method
} // end class
```

背景情報については、java.lang.Runtime.exec() を使用するを参照してください。

## 例: C から Java を呼び出す

次に示すのは、system() 関数を使用して Java Hello プログラムを呼び出す C プログラムの例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
#include <stdlib.h>

int main(void)
{
    int result;

    /* The system function passes the given string to the CL command processor
    for processing. */

    result = system("JAVA CLASS('com.ibm.as400.system.Hello')");
}

```

## 例: RPG から Java を呼び出す

次に示すのは、QCMDEXC API を使用して Java Hello プログラムを呼び出す RPG プログラムの例です。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
D*             DEFINE THE PARAMETERS FOR THE QCMDEXC API
D*
DCMDSTRING    S             25    INZ('JAVA CLASS(''com.ibm.as400.system.Hello''))
DCMDLENGTH    S             15P 5 INZ(25)
D*           NOW THE CALL TO QCMDEXC WITH THE 'JAVA' CL COMMAND
C             CALL 'QCMDEXC'
C             PARM                                CMDSTRING
C             PARM                                CMDLENGTH
C*           This next line displays 'DID IT' after you exit the
C*           Java Shell via F3 or F12.
C 'DID IT'    DSPLY
C*           Set On LR to exit the RPG program
C             SETON                                LR
C
```

## 例: プロセス間通信に入出力ストリームを使用する

この例では、Java から C プログラムを呼び出し、プロセス間通信に入出力ストリームを使用する方法を示します。

C プログラムは、その標準出力ストリームにストリングを書き込み、Java プログラムは、このストリングを読み取り、表示します。この例では、JAVSAMPLIB というライブラリーが作成されていることと、その中で CSAMP1 プログラムが作成されていることを前提としています。

注: JAVSAMPLIB は、IBM Developer Kit ライセンス・プログラム (LP) (番号 5722-JV1) のインストール・プロセスの一部としては作成されません。明示的にそれを作成しなければなりません。

### 例 1: CallPgm クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.io.*;

public class CallPgm
{
    public static void main(String args[])
    {
        Process theProcess = null;
        BufferedReader inStream = null;

        System.out.println("CallPgm.main() invoked");
    }
}

```



```

// call the CSAMP1 program
try
{
    theProcess = Runtime.getRuntime().exec(
        "/QSYS.LIB/JAVSAMPLIB.LIB/CSAMP1.PGM");
}
catch(IOException e)
{
    System.err.println("Error on exec() method");
    e.printStackTrace();
}

// read from the called program's standard output stream
try
{
    inStream = new BufferedReader(new InputStreamReader
        (theProcess.getInputStream()));
    System.out.println(inStream.readLine());
}
catch(IOException e)
{
    System.err.println("Error on inStream.readLine()");
    e.printStackTrace();
}

} // end method

} // end class

```

## 例 2: CSAMP1 C プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* args[])
{
    /* Convert the string to ASCII at compile time */
    #pragma convert(819)
    printf("Program JAVSAMPLIB/CSAMP1 was invoked\n");
    #pragma convert(0)
    /* Stdout may be buffered, so flush the buffer */
    fflush(stdout);
}

```

詳細については、プロセス間通信に入出力ストリームを使用するを参照してください。

## 例: Java 呼び出し API

この例は、標準の呼び出し API パラダイムに従っています。

これは以下を実行します。

- JNI\_CreateJavaVM を使用して Java 仮想マシンを作成する。
- Java 仮想マシンを使用して、実行したいクラス・ファイルを検索する。
- クラスの main メソッドの methodID を検索する。
- クラスの main メソッドを呼び出す。
- 例外が発生した場合に、エラーを報告する。

プログラムを作成する際は、QJVAJNI または QJVAJNI64 サービス・プログラムが、JNI\_CreateJavaVM 呼び出し API 機能を提供します。JNI\_CreateJavaVM は Java 仮想マシンを作成します。

注: QJVAJNI64 は、teraspace/LLP64 ネイティブ・メソッドと呼び出し API のサポートのための新しいサービス・プログラムです。

これらのサービス・プログラムは、システム・バインディング・ディレクトリーにあり、制御言語 (CL) 作成コマンドで明示的に示す必要はありません。たとえば、前述のサービス・プログラムを「プログラム作成 (CRTPGM)」コマンドや「サービス・プログラムの作成 (CRTSRVPGM)」コマンドを使用する際に明示的に示すことはしません。

このプログラムを実行する方法の 1 つは、以下の制御言語 (CL) コマンドを使用することです。

```
SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)
```

Java 仮想マシンを作成するジョブは、マルチスレッド対応でなければなりません。主プログラムからの出力と、プログラムからのすべての出力は、最終的に QPRINT スプール・ファイルに送られます。「投入されたジョブの処理 (WRKSBMJOB)」制御言語 (CL) コマンドを使用し、「ジョブの投入 (SBMJOB)」CL コマンドで開始したジョブを表示すると、これらのスプール・ファイルを見ることができます。

## 例: Java 呼び出し API を使用する

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
#define OS400_JVM_12
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <jni.h>

/* Specify the pragma that causes all literal strings in the
 * source code to be stored in ASCII (which, for the strings
 * used, is equivalent to UTF-8)
 */

#pragma convert(819)

/* Procedure: Oops
 *
 * Description: Helper routine that is called when a JNI function
 * returns a zero value, indicating a serious error.
 * This routine reports the exception to stderr and
 * ends the JVM abruptly with a call to FatalError.
 *
 * Parameters: env -- JNIEnv* to use for JNI calls
 * msg -- char* pointing to error description in UTF-8
 *
 * Note: Control does not return after the call to FatalError
 * and it does not return from this procedure.
 */

void Oops(JNIEnv* env, char *msg) {
    if ((*env)->ExceptionOccurred(env)) {
        (*env)->ExceptionDescribe(env);
    }
    (*env)->FatalError(env, msg);
}

/* This is the program's "main" routine. */
int main (int argc, char *argv[])
```

```

{
    JavaVMInitArgs initArgs; /* Virtual Machine (VM) initialization structure, passed by
        * reference to JNI_CreateJavaVM(). See jni.h for details
        */
    JavaVM* myJVM;          /* JavaVM pointer set by call to JNI_CreateJavaVM */
    JNIEnv* myEnv;         /* JNIEnv pointer set by call to JNI_CreateJavaVM */
    char*   myClasspath;   /* Changeable classpath 'string' */
    jclass  myClass;       /* The class to call, 'NativeHello'. */
    jmethodID mainID;      /* The method ID of its 'main' routine. */
    jclass  stringClass;   /* Needed to create the String[] arg for main */
    jobjectArray args;     /* The String[] itself */
    JavaVMOption options[1]; /* Options array -- use options to set classpath */
    int     fd0, fd1, fd2; /* file descriptors for IO */

    /* Open the file descriptors so that IO works. */
    fd0 = open("/dev/null1", O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IROTH);
    fd1 = open("/dev/null2", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);
    fd2 = open("/dev/null3", O_CREAT|O_TRUNC|O_WRONLY, S_IWUSR|S_IWOTH);

    /* Set the version field of the initialization arguments for J2SDK v1.3. */
    initArgs.version = 0x00010002;
    /* To use J2SDK v1.4, set initArgs.version = 0x00010004; */
    /* To use J2SDK v1.5, set initArgs.version = 0x00010005; */

    /* Now, you want to specify the directory for the class to run in the classpath.
     * with Java2, classpath is passed in as an option.
     * Note: You must specify the directory name in UTF-8 format. So, you wrap
     *       blocks of code in #pragma convert statements.
     */
    options[0].optionString="-Djava.class.path=/CrtJvmExample";
    /*To use J2SDK v1.4 or v1.5, replace the '1.3' with '1.4' or '1.5'.
    options[1].optionString="-Djava.version=1.3" */

    initArgs.options=options; /* Pass in the classpath that has been set up. */
    initArgs.nOptions = 2;    /* Pass in classpath and version options */

    /* Create the JVM -- a nonzero return code indicates there was
     * an error. Drop back into EBCDIC and write a message to stderr
     * before exiting the program.
     */
    if (JNI_CreateJavaVM("myJVM, (void **)myEnv, (void *)initArgs)) {
#pragma convert(0)
        fprintf(stderr, "Failed to create the JVM\n");
#pragma convert(819)
        exit(1);
    }

    /* Use the newly created JVM to find the example class,
     * called 'NativeHello'.
     */
    myClass = (*myEnv)->FindClass(myEnv, "NativeHello");
    if (! myClass) {
        Oops(myEnv, "Failed to find class 'NativeHello'");
    }

    /* Now, get the method identifier for the 'main' entry point
     * of the class.
     * Note: The signature of 'main' is always the same for any
     *       class called by the following java command:
     *       "main", "([Ljava/lang/String;)V"
     */
    mainID = (*myEnv)->GetStaticMethodID(myEnv, myClass, "main",
        "([Ljava/lang/String;)V");
    if (! mainID) {
        Oops(myEnv, "Failed to find jmethodID of 'main'");
    }
}

```

```

}

/* Get the jclass for String to create the array
 * of String to pass to 'main'.
 */
stringClass = (*myEnv)->FindClass(myEnv, "java/lang/String");
if (! stringClass) {
    Oops(myEnv, "Failed to find java/lang/String");
}

/* Now, you need to create an empty array of strings,
 * since main requires such an array as a parameter.
 */
args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0);
if (! args) {
    Oops(myEnv, "Failed to create args array");
}

/* Now, you have the methodID of main and the class, so you can
 * call the main method.
 */
(*myEnv)->CallStaticVoidMethod(myEnv,myClass,mainID,args);

/* Check for errors. */
if ((*myEnv)->ExceptionOccurred(myEnv)) {
    (*myEnv)->ExceptionDescribe(myEnv);
}

/* Finally, destroy the JavaVM that you created. */
(*myJVM)->DestroyJavaVM(myJVM);

/* All done. */
return 0;
}

```

詳細については、Java 呼び出し API を参照してください。

## 例: Java 用の IBM i5/OS PASE ネイティブ・メソッド

この Java 用の IBM i5/OS PASE ネイティブ・メソッドの例では、Java ネイティブ・インターフェース (JNI) を使用して Java コードにコールバックを行うネイティブ C メソッドのインスタンスを呼び出します。この例では、Java コードから直接ストリングにアクセスするのではなく、JNI を通して Java にコールバックを行ってストリング値を取得するネイティブ・メソッドを呼び出します。

このソース・ファイル例の HTML 版を表示するには、以下のリンクを使用してください。

**注:** サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

- [PaseExample1.java](#)
- [PaseExample1.c](#)

この i5/OS PASE ネイティブ・メソッド例を実行するには、まず以下のタスクを完了する必要があります。

1. 使用している AIX ワークステーションへのソース・コード例のダウンロード
2. ソース・コード例の準備をする
3. iSeries サーバーの準備

## i5/OS PASE native method for Java 例の実行

上記のタスクを完了したら、この例を実行することができます。このプログラム例を実行するには、以下のコマンドのいずれかを使用します。

- iSeries サーバーのコマンド・プロンプトから:

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- Qshell コマンド・プロンプトまたは i5/OS PASE 端末セッションから:

```
cd /home/example
java PaseExample1
```

### 例: PaseExample1.java

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
////////////////////////////////////
//
// This example program loads the native method library 'PaseExample1'.
// The source code for the native method is contained in PaseExample1.c
// The printString method in this Java program uses a native method,
// getStringNative to retrieve the value of the String. The native method
// simply calls back into the getStringCallback method of this class.
//
////////////////////////////////////
```

```
public class PaseExample1 {
    public static void main(String args[]) {
        PaseExample1 pe1 = new PaseExample1("String for PaseExample1");
        pe1.printString();
    }

    String str;

    PaseExample1(String s) {
        str = s;
    }

    //-----
    public void printString() {
        String result = getStringNative();
        System.out.println("Value of str is '" + result + "'");
    }

    // This calls getStringCallback through JNI.
    public native String getStringNative();

    // Called by getStringNative via JNI.
    public String getStringCallback() {
        return str;
    }

    //-----
    static {
        System.loadLibrary("PaseExample1");
    }
}
```

#### リンク集

コードのサンプルに関する特記事項

## 例: PaseExample1.c

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
/*
 *
 * This native method implements the getStringNative method of class
 * PaseExample1. It uses the JNI function CallObjectMethod to call
 * back to the getStringCallback method of class PaseExample1.
 *
 * Compile this code in AIX to create module 'libPaseExample1.so'.
 */

#include "PaseExample1.h"
#include <stdlib.h>

/*
 * Class:    PaseExample1
 * Method:   getStringNative
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_PaseExample1_getStringNative(JNIEnv* env, jobject obj) {
    char* methodName = "getStringCallback";
    char* methodSig = "()Ljava/lang/String;";
    jclass clazz = (*env)->GetObjectClass(env, obj);
    jmethodID methodID = (*env)->GetMethodID(env, clazz, methodName, methodSig);
    return (*env)->CallObjectMethod(env, obj, methodID);
}
```

### リンク集

コードのサンプルに関する特記事項

## 例: 使用している AIX ワークステーションへのソース・コード例のダウンロード

Java 用の IBM i5/OS PASE ネイティブ・メソッドの例を実行する前に、ソース・コードを含む圧縮ファイルをダウンロードする必要があります。圧縮ファイルを AIX ワークステーションにダウンロードするには、以下のステップを完成させます。

1. AIX ワークステーションに、ソース・ファイルを入れる一時ディレクトリーを作成します。
2. i5/OS PASE ソース・コード例を一時ディレクトリーにダウンロードします。
3. 例のファイルを一時ディレクトリーに unzip します。

Java 用の IBM i5/OS PASE ネイティブ・メソッドの例についての詳細は、以下のトピックを参照してください。

- 例: Java 用の IBM i5/OS PASE ネイティブ・メソッド
- 例: ソース・コード例の準備
- 例: iSeries サーバーの準備

### リンク集

i5/OS PASE ソース・コード例のダウンロード

例: Java 用の IBM i5/OS PASE ネイティブ・メソッド

この Java 用の IBM i5/OS PASE ネイティブ・メソッドの例では、Java ネイティブ・インターフェース (JNI) を使用して Java コードにコールバックを行うネイティブ C メソッドのインスタンスを呼び出します。この例では、Java コードから直接ストリングにアクセスするのではなく、JNI を通して Java にコールバックを行ってストリング値を取得するネイティブ・メソッドを呼び出します。

例: ソース・コード例の準備

例: iSeries サーバーの準備

## 例: ソース・コード例の準備

Java 用の IBM i5/OS PASE ネイティブ・メソッドを自分の iSeries サーバーに移動する前に、ソース・コードをコンパイルし、C インクルード・ファイルを作成し、共用ライブラリー・オブジェクトを作成する必要があります。

例には、以下の C および Java ソース・ファイルが含まれています。

- **PaseExample1.c:** getStringNative() のインプリメンテーションを含む C ソース・コード・ファイル。
- **PaseExample1.java:** C プログラムでネイティブ getStringNative メソッドを呼び出す Java ソース・コード・ファイル。

コンパイルした Java .class ファイルを使用して、C インクルード・ファイル PaseExample1.h を作成します。ここでは、C ソース・コードに含まれる getStringNative メソッドの機能プロトタイプが含まれています。

AIX ワークステーションでのソース・コード例を準備するには、以下のステップを完成させます。

1. 次のコマンドを使用して、Java ソース・コードをコンパイルします。

```
javac PaseExample1.java
```

2. 次のコマンドを使用して、ネイティブ・メソッド・プロトタイプを含む C インクルード・ファイルを作成します。

```
javah -jni PaseExample
```

新しい C インクルード・ファイル (PaseExample1.h) には、getStringNative メソッドの機能プロトタイプが含まれます。C ソース・コード例 (PaseExample1.c) には、getStringNative メソッドを使用するために、C インクルード・ファイルからコピーして変更した情報が含まれています。JNI の使用の詳細は、Sun Web サイトの、Java Native Interface tutorial を参照してください。

3. 次のコマンドを使用して、C ソース・コードをコンパイルし、共用ライブラリー・オブジェクトを作成します。

```
xlc -G -I/usr/local/java/J1.3.0/include PaseExample1.c -o libPaseExample1.so
```

新しい共用ライブラリー・オブジェクト・ファイル (libPaseExample1.so) には、例で使用されるネイティブ・メソッド・ライブラリー "PaseExample1" が含まれます。

**注:** それぞれの AIX システムで、正しい Java ネイティブ・メソッド・インクルード・ファイル (たとえば、jni.h) を含むディレクトリーを示すために、-I オプションを変更しなければならない場合があります。

Java 用の IBM i5/OS PASE ネイティブ・メソッドの例についての詳細は、以下のトピックを参照してください。

- 例: Java 用の IBM i5/OS PASE ネイティブ・メソッド
- 例: 使用している AIX ワークステーションへのソース・コード例のダウンロード
- 例: iSeries サーバーの準備

### リンク集

Java Native Interface tutorial



例: Java 用の IBM i5/OS PASE ネイティブ・メソッド

この Java 用の IBM i5/OS PASE ネイティブ・メソッドの例では、Java ネイティブ・インターフェース (JNI) を使用して Java コードにコールバックを行うネイティブ C メソッドのインスタンスを呼び出します。この例では、Java コードから直接ストリングにアクセスするのではなく、JNI を通して Java にコールバックを行ってストリング値を取得するネイティブ・メソッドを呼び出します。

例: 使用している AIX ワークステーションへのソース・コード例のダウンロード

例: iSeries サーバーの準備

## 例: iSeries サーバーの準備

Java 用の IBM i5/OS PASE ネイティブ・メソッド例を実行する前に、iSeries サーバーで例を実行できるよう準備する必要があります。サーバーを準備する場合、ファイルをサーバーにコピーし、例を実行するのに必要な環境変数を追加する必要があります。

サーバーを準備するには、以下のステップを完成させます。

1. サーバーに、例のファイルを含めるための、次の統合ファイル・システム・ディレクトリーを作成します。たとえば、次の制御言語 (CL) コマンドを使用して、/home/example というディレクトリーを作成します。

```
mkdir /home/example
```

2. 以下のファイルを新しいディレクトリーにコピーします。

- PaseExample1.class
- libPaseExample1.so

3. iSeries コマンド・プロンプトで、以下の制御言語 (CL) コマンドを使用して、必要な環境変数を追加します。

```
addenvvar PASE_THREAD_ATTACH 'Y'  
addenvvar PASE_LIBPATH '/home/example'  
addenvvar QIBM_JAVA_PASE_STARTUP '/usr/lib/start32'
```

注: i5/OS PASE 端末セッションから PASE ネイティブ・メソッドを使用する場合、32 ビットの PASE 環境はすでに開始されています。この場合、PASE\_THREAD\_ATTACH を Y にセットし、PASE\_LIBPATH を PASE ネイティブ・メソッド・ライブラリーのパスにセットすることだけを行います。この状態では、QIBM\_JAVA\_PASE\_STARTUP を定義すると、JVM は正常に始動しません。

追加された環境変数についての詳細は、以下のトピックを参照してください。

- IBM i5/OS PASE 例の環境変数

Java 用の IBM i5/OS PASE ネイティブ・メソッドの例についての詳細は、以下のトピックを参照してください。

- 例: Java 用の IBM i5/OS PASE ネイティブ・メソッド
- 例: 使用している AIX ワークステーションへのソース・コード例のダウンロード
- 例: ソース・コード例の準備

### リンク集

IBM i5/OS PASE 例の環境変数

IBM i5/OS PASE native methods for Java 例を使用するには、環境変数を設定する必要があります。

例: Java 用の IBM i5/OS PASE ネイティブ・メソッド

この Java 用の IBM i5/OS PASE ネイティブ・メソッドの例では、Java ネイティブ・インターフェース (JNI) を使用して Java コードにコールバックを行うネイティブ C メソッドのインスタンスを呼び

出します。この例では、Java コードから直接ストリングにアクセスするのではなく、JNI を通して Java にコールバックを行ってストリング値を取得するネイティブ・メソッドを呼び出します。

例: 使用している AIX ワークステーションへのソース・コード例のダウンロード

例: ソース・コード例の準備

## 例: ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用する

このサンプル・プログラムは、“Hello, World” と表示するために C ネイティブ・メソッドを使用する、単純な Java ネイティブ・インターフェース (JNI) の例です。NativeHello.h ファイルを作成するには、NativeHello クラス・ファイルに対して javah ツールを使用します。この例では、NativeHello の C での実装が、NATHELLO と呼ばれるサービス・プログラムの一部であると想定しています。

注: このサンプルが実行されるためには、NATHELLO サービス・プログラムの存在するライブラリーがライブラリー・リストに入っていないと仮定していません。

### 例 1: NativeHello クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
public class NativeHello {

    // Declare a field of type 'String' in the NativeHello object.
    // This is an 'instance' field, so every NativeHello object
    // contains one.
    public String theString;           // instance variable

    // Declare the native method itself. This native method
    // creates a new string object, and places a reference to it
    // into 'theString'
    public native void setTheString(); // native method to set string

    // This 'static initializer' code is called before the class is
    // first used.
    static {

        // Attempt to load the native method library. If you do not
        // find it, write a message to 'out', and try a hardcoded path.
        // If that fails, then exit.
        try {

            // System.loadLibrary uses the iSeries library list in JDK 1.1,
            // and uses the java.library.path property or the LIBPATH environment
            // variable in JDK1.2
            System.loadLibrary("NATHELLO");
        }

        catch (UnsatisfiedLinkError e1) {

            // Did not find the service program.
            System.out.println
                ("I did not find NATHELLO *SRVPGM.");
            System.out.println ("I will try a hardcoded path");

            try {

                // System.load takes the full integrated file system form path.
                System.load ("/qsys.lib/jniexample.lib/nathello.srvpgm");
            }
        }
    }
}
```

```

        catch (UnsatisfiedLinkError e2) {

            // If you get to this point, then you are done! Write the message
            // and exit.
            System.out.println
                ("<sigh> I did not find NATHELLO *SRVPGM anywhere. Goodbye");
            System.exit(1);
        }
    }

// Here is the 'main' code of this class. This is what runs when you
// enter 'java NativeHello' on the command line.
public static void main(String argv[]){

    // Allocate a new NativeHello object now.
    NativeHello nh = new NativeHello();

    // Echo location.
    System.out.println("(Java) Instantiated NativeHello object");
    System.out.println("(Java) string field is '" + nh.theString + "'");
    System.out.println("(Java) Calling native method to set the string");

    // Here is the call to the native method.
    nh.setTheString();

    // Now, print the value after the call to double check.
    System.out.println("(Java) Returned from the native method");
    System.out.println("(Java) string field is '" + nh.theString + "'");
    System.out.println("(Java) All done...");
}
}

```

## 例 2: 生成された NativeHello.h ヘッダー・ファイル

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class NativeHello */

#ifndef _Included_NativeHello
#define _Included_NativeHello
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      NativeHello
 * Method:    setTheString
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_NativeHello_setTheString
    (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif

```

次に示されている NativeHello.C の例は、ネイティブ・メソッドを C で実装したものです。この例は、Java をネイティブ・メソッドにリンクする方法を示すものです。ただし、iSeries サーバーが内部的に拡張 2 進化 10 進コード (EBCDIC) マシンであることから生じる複雑さも示しています。また、現在、JNI に本当の意味での国際化要素が欠けていることから生じる複雑さも示しています。

このような状況は JNI で初めて生じたものではありませんが、これらの理由から、作成する C コードには iSeries サーバー固有の違いがあります。STDOUT や STDERR への書き込み、あるいは STDIN からの読み取りを行う場合には、データがおそらく EBCDIC 形式でエンコードされることに留意しなければなりません。

C コードでは、大部分のリテラル・ストリング (7 ビット文字だけを含むもの) を、JNI によって必要とされる UTF-8 形式に簡単に変換することができます。これを行うには、リテラル・ストリングをコード・ページ変換プラグマで囲みます。ただし、C コードから情報を直接 STDOUT または STDERR に書き込むことが必要な場合があるため、一部のリテラルを EBCDIC のまま残しておくこともできます。

注: #pragma convert(0) ステートメントは、文字データを EBCDIC に変換します。#pragma convert(819) ステートメントは、文字データを ASCII コードに変換します。これらのステートメントは、C プログラム内の文字データをコンパイル時に変換します。

### 例 3: NativeHello Java クラスの NativeHello.c ネイティブ・メソッドの実装

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
#include <stdlib.h>      /* malloc, free, and so forth */
#include <stdio.h>       /* fprintf(), and so forth */
#include <qtqiconv.H>    /* iconv() interface */
#include <string.h>      /* memset(), and so forth */
#include "NativeHello.h" /* generated by 'javah-jni' */

/* All literal strings are ISO-8859-1 Latin 1 code page (and with 7-bit
characters, they are also automatically UTF-8). */
#pragma convert(819) /* handle all literal strings as ASCII */

/* Report and clear a JNI exception. */
static void HandleError(JNIEnv*);

/* Print an UTF-8 string to stderr in the coded character */
set identifier (CCSID) of the current job. */
static void JobPrint(JNIEnv*, char*);

/* Constants describing which direction to convert: */
#define CONV_UTF2JOB 1
#define CONV_JOB2UTF 2

/* Convert a string from the CCSID of the job to UTF-8, or vice-versa. */
int StringConvert(int direction, char *sourceStr, char *targetStr);

/* Native method implementation of 'setTheString()'. */

JNIEXPORT void JNICALL Java_NativeHello_setTheString
(JNIEnv *env, jobject javaThis)
{
    jclass thisClass; /* class for 'this' object */
    jstring stringObject; /* new string, to be put in field in 'this' */
    jfieldID fid; /* field ID required to update field in 'this' */
    jthrowable exception; /* exception, retrieved using ExceptionOccurred */

    /* Write status to console. */
    JobPrint(env, "( C ) In the native method\n");

    /* Build the new string object. */
    if (! (stringObject = (*env)->NewStringUTF(env, "Hello, native world!")))
    {
        /* For nearly every function in the JNI, a null return value indicates
        that there was an error, and that an exception had been placed where it
        could be retrieved by 'ExceptionOccurred()'. In this case, the error
        would typically be fatal, but for purposes of this example, go ahead
        and catch the error, and continue. */
    }
}
```

```

        HandleError(env);
        return;
    }

    /* get the class of the 'this' object, required to get the fieldID */
    if (! (thisClass = (*env)->GetObjectClass(env,javaThis))
    {
        /* A null class returned from GetObjectClass indicates that there
        was a problem. Instead of handling this problem, simply return and
        know that the return to Java automatically 'throws' the stored Java
        exception. */
        return;
    }

    /* Get the fieldID to update. */
    if (! (fid = (*env)->GetFieldID(env,
        thisClass,
        "theString",
        "Ljava/lang/String;")))
    {
        /* A null fieldID returned from GetFieldID indicates that there
        was a problem. Report the problem from here and clear it.
        Leave the string unchanged. */
        HandleError(env);
        return;
    }

    JobPrint(env, "( C ) Setting the field%n");

    /* Make the actual update.
    Note: SetObjectField is an example of an interface that does
    not return a return value that can be tested. In this case, it
    is necessary to call ExceptionOccurred() to see if there
    was a problem with storing the value */
    (*env)->SetObjectField(env, javaThis, fid, stringObject);

    /* Check to see if the update was successful. If not, report the error. */
    if ((*env)->ExceptionOccurred(env)) {

        /* A non-null exception object came back from ExceptionOccurred,
        so there is a problem and you must report the error. */
        HandleError(env);
    }

    JobPrint(env, "( C ) Returning from the native method%n");
    return;
}

static void HandleError(JNIEnv *env)
{
    /* A simple routine to report and handle an exception. */
    JobPrint(env, "( C ) Error occurred on JNI call: ");
    (*env)->ExceptionDescribe(env); /* write exception data to the console */
    (*env)->ExceptionClear(env); /* clear the exception that was pending */
}

static void JobPrint(JNIEnv *env, char *str)
{
    char *jobStr;
    char buf[512];
    size_t len;

    len = strlen(str);

    /* Only print non-empty string. */
    if (len) {

```

```

        jobStr = (len >= 512) ? malloc(len+1) : &buf;
        if (! StringConvert(CONV_UTF2JOB, str, jobStr))
            (*env)->FatalError
                (env,"ERROR in JobPrint: Unable to convert UTF2JOB");
        fprintf(stderr, jobStr);
        if (len >= 512) free(jobStr);
    }
}

int StringConvert(int direction, char *sourceStr, char *targetStr)
{
    QtqCode_T source, target; /* parameters to instantiate iconv */
    size_t sStrLen, tStrLen; /* local copies of string lengths */
    iconv_t ourConverter; /* the actual conversion descriptor */
    int iconvRC; /* return code from the conversion */
    size_t originalLen; /* original length of the sourceStr */

    /* Make local copies of the input and output sizes that are initialized
    to the size of the input string. The iconv() requires the
    length parameters to be passed by address (that is as int*). */
    originalLen = sStrLen = tStrLen = strlen(sourceStr);

    /* Initialize the parameters to the QtqIconvOpen() to zero. */
    memset(&source,0x00,sizeof(source));
    memset(&target,0x00,sizeof(target));

    /* Depending on direction parameter, set either SOURCE
    or TARGET CCSID to ISO 8859-1 Latin. */
    if (CONV_UTF2JOB == direction) {
        source.CCSID = 819;
    }
    else {
        target.CCSID = 819;
    }

    /* Create the iconv_t converter object. */
    ourConverter = QtqIconvOpen(&target,&source);

    /* Make sure that you have a valid converter, otherwise return 0. */
    if (-1 == ourConverter.return_value) return 0;

    /* Perform the conversion. */
    iconvRC = iconv(ourConverter,
                    (char**) &sourceStr,
                    &sStrLen,
                    &targetStr,
                    &tStrLen);

    /* If the conversion failed, return a zero. */
    if (0 != iconvRC) return 0;

    /* Close the conversion descriptor. */
    iconv_close(ourConverter);

    /* The targetStr returns pointing to the character just
    past the last converted character, so set the null
    there now. */
    *targetStr = '\0';

    /* Return the number of characters that were processed. */
    return originalLen-tStrLen;
}

#pragma convert(0)

```

背景情報については、ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用するを参照してください。

## リンク集

コードのサンプルに関する特記事項

コードのサンプルに関する特記事項

リテラル・ストリング

リテラル・ストリングが 7 ビットの情報交換用米国標準コード (ASCII) 表記の文字で構成されている場合、ストリングを UTF-8 でエンコードすることは比較的簡単です。

コードのサンプルに関する特記事項

ネイティブ・メソッドのために Java ネイティブ・インターフェースを使用する

ネイティブ・メソッドは、Pure Java ではプログラミングの要件を満たすことができない場合にのみ使用してください。

## 例: プロセス間通信のためにソケットを使用する

この例では、ソケットを使用して Java プログラムと C プログラムとの間で通信します。

最初に、ソケット上で聴取する C プログラムを開始してください。Java プログラムがソケットに接続された後は、ソケット接続を使用して C プログラムがそれにストリングを送ります。C プログラムから送られるストリングは、コード・ページ 819 の ASCII コードのストリングです。

Qshell インタープリターのコマンド行か、または他の Java プラットフォームで、コマンド `java TalkToC xxxxx nnnn` を使用して Java プログラムを開始しなくてはなりません。または iSeries コマンド行に `JAVA TALKTOC PARM(yyyyy nnnn)` を入力することにより、Java プログラムを開始します。yyyyy は、C プログラムが実行されているシステムのドメイン・ネームまたはインターネット・プロトコル (IP) アドレスです。nnnn は、C プログラムが使用するソケットのポート番号です。このポート番号は、C プログラムを呼び出すときに最初に渡すパラメーターとして指定する必要があります。

### 例 1: TalkToC クライアント・クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
import java.net.*;
import java.io.*;

class TalkToC
{
    private String host = null;
    private int port = -999;
    private Socket socket = null;
    private BufferedReader inStream = null;

    public static void main(String[] args)
    {
        TalkToC caller = new TalkToC();
        caller.host = args[0];
        caller.port = new Integer(args[1]).intValue();
        caller.setUp();
        caller.converse();
        caller.cleanup();

    } // end main() method

    public void setUp()
```



```

{
    System.out.println("TalkToC.setUp() invoked");

    try
    {
        socket = new Socket(host, port);
        inStream = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
    }
    catch(UnknownHostException e)
    {
        System.err.println("Cannot find host called: " + host);
        e.printStackTrace();
        System.exit(-1);
    }
    catch(IOException e)
    {
        System.err.println("Could not establish connection for " + host);
        e.printStackTrace();
        System.exit(-1);
    }
} // end setUp() method

public void converse()
{
    System.out.println("TalkToC.converse() invoked");

    if (socket != null && inStream != null)
    {
        try
        {
            System.out.println(inStream.readLine());
        }
        catch(IOException e)
        {
            System.err.println("Conversation error with host " + host);
            e.printStackTrace();
        }
    }

    } // end if

} // end converse() method

public void cleanUp()
{
    try
    {
        if(inStream != null)
        {
            inStream.close();
        }
        if(socket != null)
        {
            socket.close();
        }
    } // end try
    catch(IOException e)
    {
        System.err.println("Error in cleanup");
        e.printStackTrace();
        System.exit(-1);
    }
} // end cleanUp() method

} // end TalkToC class

```

SockServ.C は、ポート番号のパラメーターを渡すことによって開始します。たとえば、CALL SockServ '2001' とします。

## 例 2: SockServ.C サーバー・プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <unistd.h>
#include <sys/time.h>

void main(int argc, char* argv[])
{
    int    portNum = atoi(argv[1]);
    int    server;
    int    client;
    int    address_len;
    int    sendrc;
    int    bndrc;
    char*  greeting;
    struct sockaddr_in local_Address;
    address_len = sizeof(local_Address);

    memset(&local_Address,0x00,sizeof(local_Address));
    local_Address.sin_family = AF_INET;
    local_Address.sin_port = htons(portNum);
    local_Address.sin_addr.s_addr = htonl(INADDR_ANY);

    #pragma convert (819)
    greeting = "This is a message from the C socket server.";
    #pragma convert (0)

    /* allocate socket */
    if((server = socket(AF_INET, SOCK_STREAM, 0))<0)
    {
        printf("failure on socket allocation\n");
        perror(NULL);
        exit(-1);
    }

    /* do bind */
    if((bndrc=bind(server,(struct sockaddr*)&local_Address, address_len))<0)
    {
        printf("Bind failed\n");
        perror(NULL);
        exit(-1);
    }

    /* invoke listen */
    listen(server, 1);

    /* wait for client request */
    if((client = accept(server,(struct sockaddr*)NULL, 0))<0)
    {
        printf("accept failed\n");
        perror(NULL);
        exit(-1);
    }
}
```

```

/* send greeting to client */
if((sendrc = send(client, greeting, strlen(greeting),0))<0)
{
    printf("Send failed\n");
    perror(NULL);
    exit(-1);
}

close(client);
close(server);
}

```

詳細については、プロセス間通信のためにソケットを使用するを参照してください。

## 例: Java パフォーマンス・データ・コンバーターを実行する

Java パフォーマンス・データ・コンバーター (JPDC) を実行するには、iSeries コマンド行または Qshell 環境のいずれかを使用できます。

### iSeries コマンド行の使用:

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

1. iSeries コマンド行に「Java プログラムの実行 (RUNJVA)」コマンドまたは JAVA コマンドを入力する。
2. クラス・パラメーター行で `com.ibm.as400.jpdc.JPDC` を入力する。
3. パラメーター行で `general pexdfn mydir/myfile myrdbdire` を入力する。
4. クラスパスのパラメーター行で `'/QIBM/ProdData/Java400/ext/JPDC.jar'` を入力する。

注: `'/QIBM/ProdData/Java400/ext/JPDC.jar'` スtringが CLASSPATH 環境変数で存在する場合には、クラスパスは省略してください。CLASSPATH 環境変数にこのStringを追加するには、「環境変数の追加 (ADDENVVAR)」コマンド、「環境変数の変更 (CHGENVVAR)」コマンド、または「環境変数の処理 (WRKENVVAR)」コマンドのいずれかを使用できます。

### Qshell 環境の使用:

1. 「Qshell の開始 (STRQSH)」コマンドを入力して、Qshell インタープリターを開始する。
2. コマンド行で次のように入力します。

```

java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC
jinsight pexdfn mydir/myfile myrdbdire

```

注: `'/QIBM/ProdData/Java400/ext/JPDC.jar'` スtringが現行の環境に追加される場合には、クラスパスは省略してください。ADDENVVAR コマンド、CHGENVVAR、または WRKENVVAR コマンドのいずれかを使用して、現行の環境にこのStringを追加できます。

背景情報については、Java パフォーマンス・データ・コンバーターを実行するを参照してください。

## 例: SQL ステートメントを Java アプリケーションに組み込む

以下の SQLJ アプリケーション例、App.sqlj は、静的 SQL を使用して更新データを DB2 サンプル・データベースの EMPLOYEE テーブルから検索します。

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator App_Cursor1 (String empno, String firstnme) ; // 1
#sql iterator App_Cursor2 (String) ;

class App
{
    /*****
    ** Register Driver **
    *****/

    static
    {
        try
        {
            Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    /*****
    ** Main **
    *****/

    public static void main(String argv[])
    {
        try
        {
            App_Cursor1 cursor1;
            App_Cursor2 cursor2;

            String str1 = null;
            String str2 = null;
            long count1;

            // URL is jdbc:db2:dbname
            String url = "jdbc:db2:sample";

            DefaultContext ctx = DefaultContext.getDefaultContext();
            if (ctx == null)
            {
                try
                {
                    // connect with default id/password
                    Connection con = DriverManager.getConnection(url);
                    con.setAutoCommit(false);
                    ctx = new DefaultContext(con);
                }
                catch (SQLException e)
                {
                    System.out.println("Error: could not get a default context");
                    System.err.println(e) ;
                    System.exit(1);
                }
                DefaultContext.setDefaultContext(ctx);
            }

            // retrieve data from the database
            System.out.println("Retrieve some data from the database.");
            #sql cursor1 = {SELECT empno, firstnme FROM employee}; // 2
        }
    }
}

```

```

// display the result set
// cursor1.next() returns false when there are no more rows
System.out.println("Received results:");
while (cursor1.next()) // 3
{
    str1 = cursor1.empno(); // 4
    str2 = cursor1.firstnme();

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
}
cursor1.close(); // 9

// retrieve number of employee from the database
#sql { SELECT count(*) into :count1 FROM employee }; // 5
if (1 == count1)
    System.out.println ("There is 1 row in employee table");
else
    System.out.println ("There are " + count1
        + " rows in employee table");

// update the database
System.out.println("Update the database.");
#sql { UPDATE employee SET firstnme = 'SHILI' WHERE empno = '000010' };

// retrieve the updated data from the database
System.out.println("Retrieve the updated data from the database.");
str1 = "000010";
#sql cursor2 = {SELECT firstnme FROM employee WHERE empno = :str1}; // 6

// display the result set
// cursor2.next() returns false when there are no more rows
System.out.println("Received results:");
while (true)
{
    #sql { FETCH :cursor2 INTO :str2 }; // 7
    if (cursor2.endFetch()) break; // 8

    System.out.print (" empno= " + str1);
    System.out.print (" firstname= " + str2);
    System.out.println("");
}
cursor2.close(); // 9

// rollback the update
System.out.println("Rollback the update.");
#sql { ROLLBACK work };
System.out.println("Rollback done.");
}
catch( Exception e )
{
    e.printStackTrace();
}
}
}

```

<sup>1</sup> 反復子を宣言する。このセクションでは、次の 2 種類の反復子を宣言します。

- App\_Cursor1: 列データのタイプおよび名前を宣言して、列名 (列に結び付けられた名前) に応じた列の値を戻します。
- App\_Cursor2: 列データのタイプを宣言して、列位置 (列に結び付けられた定位置) に応じた列の値を戻します。

<sup>2</sup> 反復子を初期設定する。反復子オブジェクト `cursor1` が照会の結果を使用して初期設定されます。照会は結果を `cursor1` に格納します。

<sup>3</sup> 反復子を次の行に進める。 `cursor1.next()` メソッドは、検索する行がなくなった場合にブール値の偽を戻します。

<sup>4</sup> データを移動する。名前付きアクセス機構メソッド `empno()` は、現在の行にある `empno` という名前の列の値を戻します。名前付きアクセス機構メソッド `firstnme()` は、現在の行にある `firstnme()` という名前の列の値を戻します。

<sup>5</sup> データをホスト変数に `SELECT` する。 `SELECT` ステートメントは、テーブル内の行数をホスト変数 `count1` に渡します。

<sup>6</sup> 反復子を初期設定する。反復子オブジェクト `cursor2` が照会の結果を使用して初期設定されます。照会は結果を `cursor2` に格納します。

<sup>7</sup> データを検索する。 `FETCH` ステートメントは、結果テーブルから `ByPos` カーソル内で宣言された最初の列の現行値を、ホスト変数 `str2` に戻します。

<sup>8</sup> `FETCH.INTO` ステートメントが成功したかを検査する。 `endFetch()` メソッドは、反復子が行に位置していない場合、つまり行を取り出す前回の試行が失敗した場合に、ブール値の真を戻します。 `endFetch()` メソッドは、行を取り出す前回の試行が成功した場合に、偽を戻します。 `DB2` は `next()` メソッドが呼び出されたときに行の取り出しを試行します。 `FETCH...INTO` ステートメントは、暗黙的に `next()` メソッドを呼び出します。

<sup>9</sup> 反復子をクローズする。 `close()` メソッドは、反復子が保持しているリソースを解放します。反復子を明示的にクローズして、システム・リソースが適時に解放されるようにしてください。

この例に関する背景情報は、SQL ステートメントを Java アプリケーションに組み込むを参照してください。

## 例: クライアントのソケット・ファクトリーを使用するように Java コードを変更する

以下の例は、`simpleSocketClient` という単純なソケット・クラスを変更し、ソケット・ファクトリーを使用してすべてのソケットを作成できるようにする方法を示しています。1 つ目の例は、ソケット・ファクトリーのない `simpleSocketClient` クラスを示しています。2 つ目の例は、ソケット・ファクトリーのある `simpleSocketClient` クラスを示しています。2 つ目の例では、`simpleSocketClient` が `factorySocketClient` に名前変更されています。

### 例 1: ソケット・ファクトリーのないソケット・クライアント・プログラム

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
/* Simple Socket Client Program */

import java.net.*;
import java.io.*;

public class simpleSocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;
```

```

if (args.length < 1) {
    System.out.println("java simpleSocketClient serverHost serverPort");
    System.out.println("serverPort defaults to 3000 if not specified.");
    return;
}
if (args.length == 2)
    serverPort = new Integer(args[1]).intValue();

System.out.println("Connecting to host " + args[0] + " at port " +
    serverPort);

// Create the socket and connect to the server.
Socket s = new Socket(args[0], serverPort);
.
.
.

// The rest of the program continues on from here.

```

**例 2:** ソケット・ファクトリーのある単純なソケット・クライアント・プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/* Simple Socket Factory Client Program */

// Notice that javax.net.* is imported to pick up the SocketFactory class.
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java factorySocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
            return;
        }
        if (args.length == 2)
            serverPort = new Integer(args[1]).intValue();

        System.out.println("Connecting to host " + args[0] + " at port " +
            serverPort);

        // Change the original simpleSocketClient program to create a
        // SocketFactory and then use the socket factory to create sockets.

        SocketFactory socketFactory = SocketFactory.getDefault();

        // Now the factory creates the socket. This is the last change
        // to the original simpleSocketClient program.

        Socket s = socketFactory.createSocket(args[0], serverPort);
        .
        .
        .

        // The rest of the program continues on from here.
    }
}

```

背景情報については、ソケット・ファクトリーを使用するための Java コードの変更を参照してください。



## 例: サーバーのソケット・ファクトリーを使用するように Java コードを変更する

以下の例は、`simpleSocketServer` という単純なソケット・クラスを変更し、ソケット・ファクトリーを使用してすべてのソケットを作成できるようにする方法を示しています。1 つ目の例は、ソケット・ファクトリーのない `simpleSocketServer` クラスを示しています。2 つ目の例は、ソケット・ファクトリーのある `simpleSocketServer` クラスを示しています。2 つ目の例では、`simpleSocketServer` が `factorySocketServer` に名前変更されています。

### 例 1: ソケット・ファクトリーのないソケット・サーバー・プログラム

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
/* File simpleSocketServer.java*/

import java.net.*;
import java.io.*;

public class simpleSocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        ServerSocket serverSocket =
            new ServerSocket(serverPort);

        // a real server would handle more than just one client like this...

        Socket s = serverSocket.accept();
        BufferedInputStream is = new BufferedInputStream(s.getInputStream());
        BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it...

        byte buffer[] = new byte[4096];

        int bytesRead;

        // read until "eof" returned
        while ((bytesRead = is.read(buffer)) > 0) {
            os.write(buffer, 0, bytesRead); // write it back
            os.flush(); // flush the output buffer
        }

        s.close();
        serverSocket.close();
    } // end main()
} // end class definition
```

### 例 2: ソケット・ファクトリーのある単純なソケット・サーバー・プログラム

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
/* File factorySocketServer.java */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        // Change the original simpleSocketServer to use a
        // ServerSocketFactory to create server sockets.
        ServerSocketFactory serverSocketFactory =
            ServerSocketFactory.getDefault();
        // Now have the factory create the server socket. This is the last
        // change from the original program.
        ServerSocket serverSocket =
            serverSocketFactory.createServerSocket(serverPort);

        // a real server would handle more than just one client like this...

        Socket s = serverSocket.accept();
        BufferedInputStream is = new BufferedInputStream(s.getInputStream());
        BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it...

        byte buffer[] = new byte[4096];

        int bytesRead;

        while ((bytesRead = is.read(buffer)) > 0) {
            os.write(buffer, 0, bytesRead);
            os.flush();
        }

        s.close();
        serverSocket.close();
    }
}
```

背景情報については、ソケット・ファクトリーを使用するための Java コードの変更を参照してください。

## 例: Secure Sockets Layer を使用するように Java クライアントを変更する

以下の例は、factorySocketClient という 1 つのクラスを変更して、Secure Sockets Layer (SSL) を使用できるようにする方法を示しています。1 つ目の例は、SSL を使用しない factorySocketClient クラスを示しています。2 つ目の例は、同じクラスで SSL を使用するものを示しており、名前が factorySSLSocketClient に変更されています。

### 例 1: SSL を使用しない単純な factorySocketClient クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```
/* Simple Socket Factory Client Program */

import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java factorySocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
            return;
        }
        if (args.length == 2)
            serverPort = new Integer(args[1]).intValue();

        System.out.println("Connecting to host " + args[0] + " at port " +
            serverPort);

        SocketFactory socketFactory = SocketFactory.getDefault();

        Socket s = socketFactory.createSocket(args[0], serverPort);
        .
        .
        .

        // The rest of the program continues on from here.
    }
}
```

### 例 2: SSL を使用する単純な factorySocketClient クラス

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
// Notice that we import javax.net.ssl.* to pick up SSL support
import javax.net.ssl.*;
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySSLSocketClient {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java factorySSLSocketClient serverHost serverPort");
            System.out.println("serverPort defaults to 3000 if not specified.");
            return;
        }
    }
}
```

```

if (args.length == 2)
    serverPort = new Integer(args[1]).intValue();

System.out.println("Connecting to host " + args[0] + " at port " +
    serverPort);

// Change this to create an SSLSocketFactory instead of a SocketFactory.
SocketFactory socketFactory = SSLSocketFactory.getDefault();

// We do not need to change anything else.
// That's the beauty of using factories!
Socket s = socketFactory.createSocket(args[0], serverPort);
.
.
.

// The rest of the program continues on from here.

```

背景情報については、Secure Sockets Layer を使用するように Java コードを変更するを参照してください。

## 例: Secure Sockets Layer を使用するように Java サーバーを変更する

以下の例は、factorySocketServer という 1 つのクラスを変更して、Secure Sockets Layer (SSL) を使用できるようにする方法を示しています。

1 つ目の例は、SSL を使用しない factorySocketServer クラスを示しています。2 つ目の例は、同じクラスで SSL を使用するものを示しており、名前が factorySSLSocketServer に変更されています。

### 例 1: SSL を使用しない単純な factorySocketServer クラス

注: サンプル・コードをご使用の場合は、560 ページの『コードに関する特記事項』に同意していただいているものとします。

```

/* File factorySocketServer.java */
// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        // Change the original simpleSocketServer to use a
        // ServerSocketFactory to create server sockets.
        ServerSocketFactory serverSocketFactory =
            ServerSocketFactory.getDefault();
        // Now have the factory create the server socket. This is the last
        // change from the original program.
        ServerSocket serverSocket =
            serverSocketFactory.createServerSocket(serverPort);

        // a real server would handle more than just one client like this...

```

```

Socket s = serverSocket.accept();
BufferedInputStream is = new BufferedInputStream(s.getInputStream());
BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

// This server just echoes back what you send it.

byte buffer[] = new byte[4096];

int bytesRead;

while ((bytesRead = is.read(buffer)) > 0) {
    os.write(buffer, 0, bytesRead);
    os.flush();
}

s.close();
serverSocket.close();
}
}

```

## 例 2: SSL を使用する単純な factorySocketServer クラス

注: 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```

/* File factorySocketServer.java */

// need to import javax.net to pick up the ServerSocketFactory class
import javax.net.*;
import java.net.*;
import java.io.*;

public class factorySocketServer {
    public static void main (String args[]) throws IOException {

        int serverPort = 3000;

        if (args.length < 1) {
            System.out.println("java simpleSocketServer serverPort");
            System.out.println("Defaulting to port 3000 since serverPort not specified.");
        }
        else
            serverPort = new Integer(args[0]).intValue();

        System.out.println("Establishing server socket at port " + serverPort);

        // Change the original simpleSocketServer to use a
        // ServerSocketFactory to create server sockets.
        ServerSocketFactory serverSocketFactory =
            ServerSocketFactory.getDefault();
        // Now have the factory create the server socket. This is the last
        // change from the original program.
        ServerSocket serverSocket =
            serverSocketFactory.createServerSocket(serverPort);

        // a real server would handle more than just one client like this...

        Socket s = serverSocket.accept();
        BufferedInputStream is = new BufferedInputStream(s.getInputStream());
        BufferedOutputStream os = new BufferedOutputStream(s.getOutputStream());

        // This server just echoes back what you send it.

        byte buffer[] = new byte[4096];

        int bytesRead;

```

```

while ((bytesRead = is.read(buffer)) > 0) {
    os.write(buffer, 0, bytesRead);
    os.flush();
}

s.close();
serverSocket.close();
}
}

```

背景情報については、Secure Sockets Layer を使用するように Java コードを変更するを参照してください。

---

## IBM Developer Kit for Java のトラブルシューティング

このトピックでは、ジョブ・ログを検索する方法と、Java プログラムの分析データを収集する方法を説明します。ここでは、プログラム一時修正 (PTF) の説明や、IBM Developer Kit for Java のサポートを受ける方法も示します。

プログラムの実行時間が長くなるとパフォーマンスが低下する場合は、誤ってメモリー・リークがコーディングされている可能性があります。iSeries iDoctor のコンポーネントである JavaWatcher を使用して、プログラムをデバッグし、メモリー・リークを見つけることができます。詳しくは、JavaWatcher を参照してください。

### 制限

このセクションでは、IBM Developer Kit for Java の既知の制限、制約事項、または固有の動きをリストします。

- あるクラスのロード時に、そのスーパークラスが見つからないと、元のクラスが見つからなかったことを示すエラーが出されます。たとえば、クラス B がクラス A を拡張する場合、クラス B のロード時にクラス A が見つからないと、実際に見つからなかったのはクラス A であるにもかかわらず、クラス B が見つからなかったことを示すエラーが出されます。あるクラスが見つからないというエラーが表示された場合は、そのクラスと、そのすべてのスーパークラスが CLASSPATH に入っているかどうかを確認してください。このことは、ロード対象のクラスによって実装されるインターフェースにも適用されます。
- ガーベッジ・コレクション・ヒープは、240 GB に制限されます。
- 構成するオブジェクトの数に明確な限度はありません。
- iSeries サーバーでの java.net バックログ・パラメーターの動きは、他のプラットフォームと異なる場合があります。以下に例を示します。
  - Listen バックログ 0、1
    - Listen(0) と指定すると、接続を 1 つ保留にすることができます。ソケットは使用禁止になります。
    - Listen(1) と指定すると、Listen(0) と同じ効果があるほか、注記を 1 つ保留にすることができます。
  - Listen バックログ > 1
    - これにより、listen 待ち行列に、保留中の多数の要求を残すことが可能になります。新しい接続要求が到着し、待ち行列が限界に達すると、保留中の要求が 1 つ削除されます。
- マルチスレッドを使用できる (つまり、スレッド・セーフな) 環境では、使用する JDK のバージョンに関係なく、1 つの Java 仮想マシンだけを使用できます。iSeries サーバーはスレッド・セーフです

が、ファイル・システムの中にはそうではないものもあります。スレッド・セーフではないファイル・システムのリストについては、統合ファイル・システムを参照してください。

- インターネット・プロトコル バージョン 6 (IPv6) のサポートは完全には実装されておらず、なんらかの副次作用が起こる可能性があります。詳しくは、ソケットを参照してください。

## Java の問題分析用のジョブ・ログを検索する

Java コマンドを実行したジョブのジョブ・ログおよび Java プログラムが実行されたバッチ即時 (BCI) ジョブ・ログを使用して、Java の障害の原因を分析してください。どちらのジョブ・ログにも重要なエラー情報が含まれている可能性があります。

BCI ジョブのジョブ・ログを検索するには、2 つの方法があります。Java コマンドを実行したジョブのジョブ・ログに利用記録がとられている BCI ジョブの名前を、検索することができます。そして、そのジョブ名を使用して、BCI ジョブのジョブ・ログを検索します。

また、以下のステップに従って、BCI ジョブのジョブ・ログを検索することができます。

1. iSeries コマンド行に「投入されたジョブの処理 (WRKSBMJOB)」コマンドを入力します。
2. リストの一番後ろへ移動します。
3. リスト中から、QJVACMDSRV という最後のジョブを見つけます。
4. そのジョブに対して、オプション 8 (スプール・ファイルの処理) を入力します。
5. QPJOBLOG というファイルが、表示されます。
6. F11 を押して、スプール・ファイルのビュー 2 を参照します。
7. 日時が、障害が発生したときの日時と一致するかを確認します。

日時がサインオフした日時と一致しない場合は、投入されたジョブのリストをさらに調べてください。サインオフした日時に一致する日時がある QJVACMDSRV ジョブ・ログを検索してください。

BCI ジョブのジョブ・ログを見つけられない場合は、そのジョブ・ログは作成されなかった可能性があります。これは、QDFTJOB D ジョブ記述の ENDSEP 値の設定が高すぎる場合、または QDFTJOB D ジョブ記述の LOG 値が \*NOLIST を指定する場合に起こります。これらの値をチェックして、BCI ジョブのジョブ・ログが作成されるようにその値を変更してください。

「Java プログラムの実行 (RUNJVA)」コマンドを実行したジョブのジョブ・ログを作成するには、以下のことを行ってください。

1. SIGNOFF \*LIST を入力します。
2. 次に、再びサインオンします。
3. iSeries コマンド行に「スプール・ファイルの処理 (WRKSPLF)」コマンドを入力します。
4. リストの一番後ろへ移動します。
5. QPJOBLOG というファイルを検索します。
6. F11 を押します。
7. 日時が、サインオフ・コマンドを入力した日時と一致することを確認します。

日時がサインオフした日時と一致しない場合は、投入されたジョブのリストをさらに調べてください。サインオフした日時に一致する日時がある QJVACMDSRV ジョブ・ログを検索してください。



## Java の問題分析用のデータを収集する

プログラム診断依頼書 (APAR) に記載するデータを収集するには、以下のことを行います。

1. 問題の完全な記述を含めます。
2. 実行中に問題の原因となった Java クラス・ファイルを保管します。
3. SAV コマンドを使用して、統合ファイル・システムからオブジェクトを保管します。このプログラムの実行に必要な他のクラス・ファイルを保管しなければならない場合があります。また、必要であれば、IBM が問題を再現するときに使用するディレクトリー全体を保管して送ることもできます。以下に、ディレクトリー全体を保管する方法の例を示します。

**例:** ディレクトリーを保管する

**注:** 法律上の重要な情報に関しては、コードの特記事項情報をお読みください。

```
SAV DEV('/QSYS.LIB/TAP01.DEVD') OBJ('/mydir')
```

可能であれば、問題に関連している Java クラスのソース・ファイルを保管します。これは、IBM が問題を再現して分析するときに役立ちます。

4. プログラムの実行に必要なネイティブ・メソッドを含む、すべてのサービス・プログラムを保管します。
5. Java プログラムの実行に必要な、すべてのデータ・ファイルを保管します。
6. 問題を再現する方法についての完全な記述を追加します。

これには次のものが含まれます。

- CLASSPATH 環境変数の値。
  - 実行された Java コマンドの記述。
  - プログラムによって要求されるどんな入力にでも応答する方法の記述。
7. 障害が起きたところに発生したすべての垂直ライセンス内部コード (VLIC) ログを含めます。
  8. Java 仮想マシンが実行していた対話式ジョブおよび BCI ジョブからジョブ・ログを追加します。

## プログラム一時修正を適用する

i5/OS V5R4 から、「Java 仮想マシンのジョブの表示 (DSPJVMJOB)」CL コマンドを使用して、システムがアクティブである間も JVM ジョブを管理して PTF を適用できるようになりました。

多くの Java プログラム一時修正 (PTF) は、JVM に影響を与えるため、JVM ジョブの実行中にコードが適用された場合、PTF の適用は予測不能の結果を引き起こす可能性があります。かつては、一部の Java PTF の適用は、システム上で実行中の JVM ジョブがないことを確実にするために、初期プログラム・ロード (IPL) がシステム上で実行できるようになるまで遅延されていました。ただし、多くのユーザーにとってこれは不便なことでした。JVM 前提条件が追加され、それらの遅延されていた PTF の多くは、アクティブな JVM がシステム上になければ、即時に適用可能になりました。DSPJVMJOB コマンドにより、どのジョブで JVM が実行中かを確認することができます。PTF を適用するために IPL を待つ代わりに、この情報を使用して、PTF を適用する前にアクティブな JVM を含むジョブを適切に終了させることができます。

DSPJVMJOB コマンドの詳細を確認するには、CL トピック内の Java 仮想マシンのジョブを表示するを参照してください。

### 関連情報

i5/OS および関連ソフトウェアの保守管理

## IBM Developer Kit for Java のサポート

IBM Developer Kit for Java のサポート・サービスは、iSeries ソフトウェア・プロダクトの通常の期間と条件のもとで提供されています。サポート・サービスには、プログラム・サービス、音声サポート、およびコンサルティング・サービスが含まれます。

詳細については、IBM iSeries ホーム・ページのトピック「Support」で提供されているオンライン情報を使用してください。IBM Support Services for 5722-JV1 (IBM Developer Kit for Java) を使用してください。または、ローカル IBM 担当員に連絡してください。

継続してプログラム・サービスを受けるには、IBM の指示により、より最近のレベルの IBM Developer Kit for Java が必要になることがあります。詳細については、複数の Java Development Kit (JDK) のサポートを参照してください。

IBM Developer Kit for Java プログラムの欠陥の解決は、プログラム・サービスまたは音声サポートによってサポートされています。アプリケーションのプログラミングまたはデバッグに関する問題の解決は、コンサルティング・サービスによってサポートされています。

IBM Developer Kit for Java アプリケーション・プログラム・インターフェース (API) 呼び出しは、以下の場合を除いてコンサルティング・サービスによりサポートされています。

1. 比較的単純なプログラムで再び発生することにより示されるような、明らかに Java API の欠陥である場合。
2. 資料の説明を求める質問である場合。
3. サンプルまたは資料の入手先についての質問の場合。

プログラミングに関するすべてのサポートは、コンサルティング・サービスにより提供されます。IBM Developer Kit for Java ライセンス・プログラム (LP) 製品に付随するプログラム・サンプルもそのサービスに含まれます。追加のサンプルは、インターネット上の IBM iSeries ホーム・ページで入手できる場合もありますが、これらはサポート対象外です。

IBM Developer Kit for Java LP により、問題の解決に関する情報が提供されます。IBM Developer Kit for Java API に潜在的な欠陥があると思われる場合は、エラーを示す簡単なプログラムが必要です。

---

## IBM Developer Kit for Java の関連情報

以下は、IBM Developer Kit for Java に関連した Javadoc 参照情報です。

### Javadoc

- iSeries 固有の JAAS Javadoc
- JAAS API 仕様
- Java 2 Platform, Standard Edition API 仕様

以下は、IBM Developer Kit for Java に関連した参照情報です。

## Java Naming and Directory Interface

Java Naming and Directory Interface (JNDI) は、JavaSoft のプラットフォーム・アプリケーション・プログラム・インターフェース (API) の一部です。JNDI により、複数の命名およびディレクトリー・サービス

にシームレスに接続することができます。このインターフェースを使用すると、強力で可搬性のある、ディレクトリーが使用可能な Java アプリケーションを作成することができます。

JavaSoft は、IBM、SunSoft、Novell、Netscape、および Hewlett-Packard Co など、業界のリーダー企業と共同で JNDI の仕様を開発しました。

注: IBM Developer Kit for Java が提供する i5/OS Java ランタイム環境 (JRE) および Java 2 Platform, Software Development Kit (J2SDK) のバージョンには、Sun LDAP プロバイダーが組み込まれています。i5/OS Java サポートには Sun LDAP プロバイダーが組み込まれているため、このサポートには `ibmjndi.jar` ファイルは組み込まれていません。 `ibmjndi.jar` ファイルは、旧バージョンの J2SDK 用の IBM 開発 LDAP サービス・プロバイダーを提供していました。

JNDI について詳しくは、Java Naming and Directory interface by Sun Microsystems, Inc. を参照してください。

### リンク集

Java Naming and Directory interface by Sun Microsystems, Inc.

## JavaMail

JavaMail API は、電子メール (E メール) システムをモデル化する抽象クラスのセットを提供します。この API はメールの読み取りおよび送信に関する一般的なメール機能を提供します。サービス・プロバイダーはプロトコルをインプリメントしなければなりません。

サービス・プロバイダーは、特定のプロトコルをインプリメントします。たとえば、Simple Mail Transfer Protocol (SMTP) は Eメールの送信用の転送プロトコルです。Post Office Protocol 3 (POP3) は Eメールの受信用の標準プロトコルです。Internet Message Access Protocol (IMAP) は POP3 の代替プロトコルです。

JavaMail は、プレーン・テキストではないメールの内容を処理するために、サービス・プロバイダーのほかに JavaBeans Activation Framework (JAF) を必要とします。これには、Multipurpose Internet Mail Extensions (MIME)、Uniform Resource Locator (URL) ページ、およびファイルの添付が含まれます。

すべての JavaMail コンポーネントが、IBM Developer Kit for Java の一部として配送されます。これらのコンポーネントには、以下のものが含まれます。

- **mail.jar** この JAR ファイルには、JavaMail API、SMTP サービス・プロバイダー、POP3 サービス・プロバイダー、および IMAP サービス・プロバイダーが含まれます。
- **activation.jar** この JAR ファイルには、JavaBeans Activation Framework が含まれています。

詳細については、Sun Microsystems, Inc. JavaMail 資料を参照してください。

### リンク集

JavaMail

## Java 印刷サービス

Java 印刷サービス (JPS) API はすべての Java プラットフォームで印刷ができるようにします。Java 1.4 および後続のバージョンは、Java ランタイム環境およびサード・パーティーが、PDF、Postscript、および高機能印刷™ (AFP™) など印刷のためのさまざまなフォーマットを作成するストリーム生成プラグインを提供できるように、フレームワークを提供します。これらのプラグインは 2 次元 (2D) グラフィック・コールから出力フォーマットを作成します。

iSeries 印刷サービスは、i5/OS の「デバイス記述作成 (プリンター) (CRTDEVPRT)」コマンドを使用して iSeries 上に構成された印刷装置を表示します。印刷装置を作成する際は、情報公開パラメーターを指定してください。それにより、iSeries 印刷サービスがサポートする印刷サービス属性の数が増加します。

プリンターが Simple Network Management Protocol (SNMP) をサポートしている場合は、iSeries 上にプリンターを構成します。CRTDEVPRT コマンドのシステム・ドライバー・プログラム・パラメーターの値として \*IBMSNMPDRV を指定してください。印刷サービスは SNMP を使用して、構成されたプリンターについての特定の情報 (プリンター・サービス属性) を検索します。

iSeries がサポートする Doc Flavor には、\*AFPDS、\*SCS、\*USERASCII - (PCL)、\*USERASCII - (ポストスクリプト)、および \*USERASCII - (PDF) が含まれます。CRTDEVPRT コマンドの「情報公開 (Publishing Information)」の中の「サポートされるデータ・ストリーム (Data Streams Supported)」パラメーターに、プリンターがサポートする Doc Flavor を指定してください。

アプリケーションが印刷サービスを使用して iSeries 上のジョブ (文書) を印刷すると、印刷サービスはその文書をスプール・ファイル内のその印刷装置と同名の (また、PrinterName 属性で指定された名前と同名の) 出力待ち行列に置きます。文書が印刷装置で印刷される前に、コマンド STRPRTWTR で印刷装置書き出しプログラムを始動してください。

Java 印刷サービス仕様で定義された属性に加えて、iSeries 印刷サービスは、すべての Doc Flavor の以下の属性をサポートします。

- PrinterFile (スプール・ファイルの作成時に使用されるプリンター・ファイル、名前、およびライブラリーを指定します)
- SaveSpooledFile (スプール・ファイルを保管するかどうかを指定します)
- UserData (10 文字のユーザー定義データのストリング)
- JobHold (スプール・ファイルを保持するかどうかを指定します)
- SourceDrawer (出力メディア用に使用するソース・ドロワーを指定します)

## | JDK 1.5 の使用時に JPS を使用できるようにする方法

| 以下は、Java 印刷サービスを使用可能にするためにセットアップする必要のあるシンボリック・リンクです。

```
| ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjps.jar')  
|     NEWLNK('/QIBM/ProdData/Java400/jdk15/lib/ext/ibmjps.jar')  
|     LNKTYPE(*SYMBOLIC)  
|  
| ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')  
|     NEWLNK('/QIBM/ProdData/Java400/jdk15/lib/ext/jt400Native.jar')  
|     LNKTYPE(*SYMBOLIC)
```

| 詳細については、Sun Microsystems の Java Print Service 資料を参照してください。

## コードに関する特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用权を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

- | 強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。
  
- | IBM、そのプログラム開発者、または供給者は、いかなる場合においてもその予見の有無を問わず、以下に対する責任を負いません。
  - | 1. データの喪失、または損傷。
  - | 2. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
  - | 3. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用
  
- | 国または地域によっては、法律の強行規定により、上記の責任の制限が適用されない場合があります。





---

## 付録. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032  
東京都港区六本木 3-2-31  
IBM World Trade Asia Corporation  
Licensing

**以下の保証は、国または地域の法律に沿わない場合は、適用されません。** IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとしします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation  
Software Interoperability Coordinator, Department YBWA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。



- | 本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム
- | 契約の契約条項、IBM プログラムのご使用条件、IBM 機械コードのご使用条件、またはそれと同等の条項
- | に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性があります。その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売り価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. \_年を入れる\_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

---

## プログラミング・インターフェース情報

本書「IBM Developer Kit for Java」には、プログラムを作成するユーザーが IBM Developer のサービスを使用するためのプログラミング・インターフェースが記述されています。

---

## 商標

以下は、IBM Corporation の商標です。

- | Advanced Function Presentation
- | AFP
- | AIX
- | AT
- | C/400
- | DB2
- | DB2 Universal Database
- | Distributed Relational Database Architecture
- | DRDA
- | i5/OS
- | IBM
- | Integrated Language Environment
- | iSeries
- | PowerPC
- | VisualAge
- | WebSphere

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

---

## 使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

**個人使用:** これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

**商業的使用:** これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、第三者の権利の不侵害の保証、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。





Printed in Japan