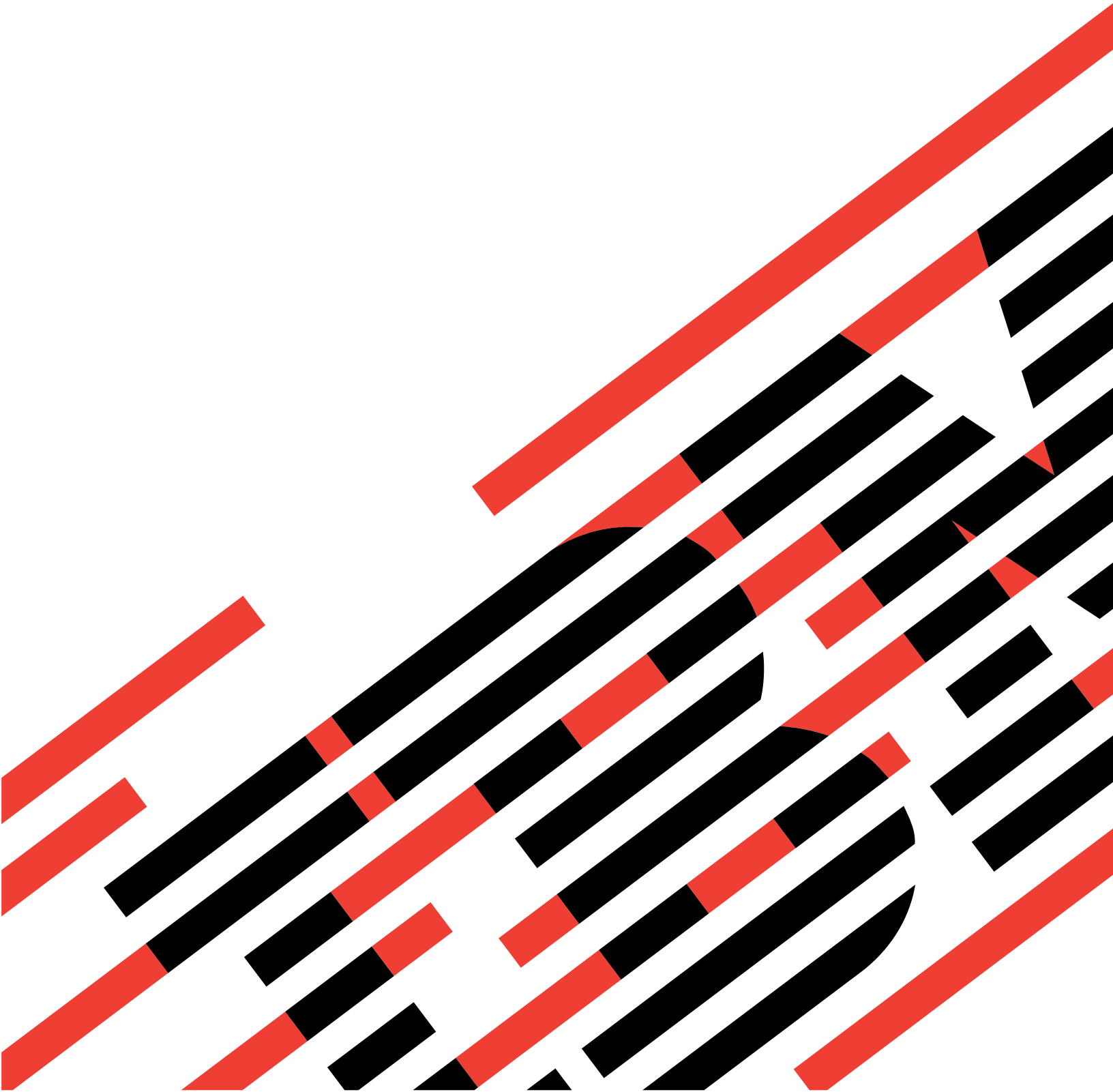
IBM

eserver

iSeries

IBM Developer Kit for Java

版本 6



IBM

@server

iSeries

IBM Developer Kit for Java

版本 6

目录

第 1 章 IBM Developer Kit for Java

为了用于 iSeries(TM) 服务器环境，已优化 IBM(R)Developer Kit for Java(TM)。它使用 Java 编程和用户接口的兼容性，所以您可以开发自己的 iSeries 服务器应用程序。

IBM Developer Kit for Java 允许您在 iSeries 服务器上创建和运行 Java 程序。因为 IBM Developer Kit for Java 是与 Sun Microsystems 的 Java 技术相兼容的实现，所以我们假定您熟悉他们的 Java Development Kit（JDK）文档。为了便于您使用他们的和我们的信息，我们提供了指向 Sun Microsystems 信息的链接。

如果由于任何原因导致指向 Sun Microsystems 的 Java Development Kit 文档的链接不起作用，则请参考他们的 HTML 参考文档以获取您所需的信息。您可以在万维网上找到此信息，网址为 The Source for Java Technology java.sun.com 。

请选择下列任何主题来了解更多有关如何使用 IBM Developer Kit for Java 的详细信息:

- 打印本主题提供了有关如何下载可打印 PDF 文件或 IBM Developer Kit for Java HTML 文件的压缩包的详细信息。
- V5R2 中的新增内容着重说明最新的产品和信息更新。
- 入门提供了有关以下各方面的信息: 安装、配置、如何创建和运行简单的 Hello World Java 程序、下载和安装以及发行版间的兼容性。
- 定制提供了有关如何在服务器上定制时区配置、系统属性和类路径的指示信息。
- 兼容性提供了关于发行版之间的 Java 类文件兼容性的信息。
- 数据库访问说明 IBM Developer Kit for Java 如何允许 Java 程序访问 iSeries 数据库文件。
- Java 与其它编程语言显示了如何通过使用 “Java 本机接口”（JNI）、java.lang.Runtime.exec()、进程间通信和 Java “调用” API 来调用使用非 Java 语言编写的代码。
- Java 平台描述了用于开发和管理 Java applet 和应用程序的环境，此平台由 Java 语言、Java 包和 Java 虚拟机组成。
- 高级主题提供了有关如何在批处理作业中运行 Java 的指示信息，并描述了要显示、运行或调试 Java 程序而在集成文件系统中需要的文件权限。
- 在不带 GUI 的主机上运行包含有关如何使用 Remote Abstract Window Toolkit（AWT）、Class Broker for Java 或 Native Abstract Windowing Toolkit（NAWT）来设置和运行 Java 程序的信息。
- 安全性提供了有关沿用权限的详细信息，并说明了如何使用 SSL 来在 Java 应用程序中保护套接字流。
- 性能提供了有关如何调整 Java 性能的信息。
- 命令和工具提供了有关如何使用 Java 命令和 Java 工具的详细信息。
- 可选包列示了您在开发 Java 应用程序时可选择使用的包，如 JavaMail 和 “Java 命名和目录服务”（JNDI）。
- 调试说明了如何调试 Java 程序。
- 故障诊断显示了如何查找作业记录和收集数据以进行 Java 程序分析。此主题还提供了有关程序临时性修订（PTF）以及获取对 IBM Developer Kit for Java 的支持的信息。
- 代码示例直接链接至本信息中的所有代码示例。

- 参考直接链接至所有 Javadoc 和 API 参考信息。

IBM Developer Kit for Java 的 V5R2 的新增功能

此主题着重说明 V5R2 中对 IBM Developer Kit for Java(TM) 的更改。我们分别地着重说明特定于 Java Development Kit（JDK）1.1.8 和 Java 2 Software Development Kit（J2SDK）Standard Edition V1.4 的更改。在 V5R2 的一般发行版之后的更新出现在以下列表的底部。

入门

- 多 JDK 支持包含有关 IBM 支持的每个 JDK 的信息。
- 将网络驱动器映射至 iSeries 服务器和在 iSeries 服务器上创建目录已移至“入门”。
- 创建、编译和运行 Hello World Java 程序包含若干处更改。

定制

- 有一些新的系统属性，包括高速缓存类装入程序的属性。

数据库访问

- JDBC 部分已作了大量修订。
- 添加了 Java 存储过程和 Java 用户定义标量函数部分。

在不带 GUI 的主机上运行

- 请查看使用 Remote Abstract Window Toolkit 来运行 Java 程序主题以了解更新。

命令和工具

- 已将“分析 Java 虚拟机”（ANZJVM）命令添加至 CL 命令部分。
- 添加了 Java idlj 工具。
- 更改了 Java 支持的“iSeries 导航器”命令以反映对“iSeries 导航器”所作的若干项更改。

可选包

- 添加了 JNDI LDAP 提供程序编程指南。

调试

- 提供了新的对通过定制类装入程序装入的 Java 类的调试支持。

代码示例

- 添加了更多代码示例。

打印本主题

- 打印本主题包含 IBM Developer Kit for Java 信息的 PDF。

参考

- 添加了 IBM Developer Kit for Java 参考部分，此部分包含 Javadoc 和 API 参考信息。

对特定版本的更改

请单击下面的链接以了解特定于您选择的版本的信息:

- Java Development Kit V1.1.8
- Java 2 Software Development Kit Standard Edition V1.4

到 2002 年 9 月 26 日为止的新增内容

支持多个 Java Development Kit

此技术更新阐明当安装了多个 JDK 时，OS/400(R) 用来确定缺省 JDK 的优先次序。

从 Java 程序中调用控制语言命令

此技术更新添加当从 Java 程序调用控制语言（CL）命令时有关不同 JDK 所需要的定界符的信息。

到 2002 年 8 月 30 日为止的新增内容

Native Abstract Windowing Toolkit

对 Native Abstract Windowing Toolkit（NAWT）的更改的信息包括对 Java Development Kit V1.3 的支持和关于安装 NAWT 的指示信息的更新。安装指示信息现在反映了对 NAWT 使用的 PRPQ 5799-PTL 的增强。

Java 一般安全性服务

“Java 一般安全性服务”（JGSS）是一个新主题，它提供有关 IBM JGSS（认证和安全消息传递的一般接口）的信息。在此接口下，可以加入各种基于保密密钥、公用密钥或其它安全性技术的安全性机制。

如何了解哪些是新增内容或更改过的内容

为了帮助您了解哪些位置已作了技术更改，本信息使用:

- » 图像来标记新信息或更改过的信息的开始位置。
- « 图像来标记新信息或更改过的信息的结束位置。

要查找关于本发行版的新增内容或更改过的内容的其它信息，请查看用户备忘录  。

V5R2 中的 Java Development Kit（JDK）1.1.8 的新增内容

在 V5R2 中，没有特定于 Java(TM) Development Kit（JDK）1.1.8 的更改。

V5R2 中 Java 2 Software Development Kit（J2SDK）Standard Edition V1.4 的新增内容

本主题着重说明 V5R2 中对 IBM Developer Kit for Java(TM) 和对 Java 2 Software Development Kit（J2SDK）Standard Edition V1.4 所作的更改。

注意: 在本节中，我们只讨论在 J2SDK V1.4 中独特的或有趣的更改。Java Development Kit（JDK）1.1.8的新增内容主题中有关更新方面的一般信息也适用于 J2SDK V1.4。

定制

- 时区环境变量显示了在 iSeries 服务器上设置时区变量的独特方面。必须使用作为 Java 2 SDK（J2SDK）Standard Edition V1.3 和更高版本一部分的本机 **getSystemTimeZoneID()** 来完成此操作。
- 添加了新的 J2SDK 系统属性。

安全性

- Java 认证和授权服务（JAAS）是对 Java 2 Software Development Kit V1.3（JDK 1.3）和更高版本的标准扩展。当前，Java 2 提供了基于代码起源位置与代码签名人的访问控制（基于代码源的访问控制）。然而，它不包含强制实施基于代码运行者的附加访问控制的能力。JAAS 提供了一个框架，该框架使用这样的支持扩充了 Java 2 安全性模型。
- Java 密码术扩展（JCE）1.2 是对 Java 2 Software Development Kit（J2SDK）Standard Edition 的标准扩展。iSeries 服务器上的 JCE 实现与 Sun Microsystems 的实现相兼容。本文档阐述了 iSeries 实现的独特方面。我们假定您熟悉与 JCE 扩展相关的一般文档。

可选包

- Java 打印服务 API 允许在所有 Java 平台上打印。Java 运行时环境和第三方可以提供流生成器插件，用于生成各种可打印格式，如 PDF、Postscript 和 AFP。
- JavaMail API 是一组模仿电子（电子邮件）系统的抽象类。此 API 提供了独立于平台且独立于协议的框架，可用来构建基于 Java 的电子邮件和消息传递应用程序。

打印本主题

要查看或下载 PDF 版本，请选择 IBM Developer Kit for Java(TM)（大约 2159 KB 或 450 页）。

保存 PDF 文件

要将 PDF 保存在工作站上以便进行查看或打印:

1. 在浏览器中右键单击 PDF（右键单击上面的链接）。
2. 单击**目标另存为...**
3. 导航至要在其中保存该 PDF 的目录。
4. 单击**保存**。

下载 Adobe Acrobat Reader

如果您需要 Adobe Acrobat Reader 来查看或打印这些 PDF，则可从 Adobe Web 站点（www.adobe.com/products/acrobat/readstep.html）  下载一个副本。

IBM Developer Kit for Java 入门

如果您还没有使用过 IBM Developer Kit for Java(TM)，则应遵循下列步骤来安装和配置它，并练习运行简单的 Hello World Java 程序。

1. 如果您已经熟悉 IBM Developer Kit for Java 信息，则参见新增内容，以获取指向最新产品更新和信息的链接。
2. 安装 IBM Developer Kit for Java。
3. 配置系统。
4. 如果您不熟悉此信息，并且还没有使用过 IBM Developer Kit for Java，则参见运行您的第一个 Hello World Java 程序。本主题说明了两种使用 IBM Developer Kit for Java 来运行简单的 Hello World Java 程序的方法。这是检查 IBM Developer Kit for Java 的安装是否正确的一种简便方法。
5. 现在您已准备好创建、编译和运行您自己的 Hello World Java 程序了。有关执行步骤，参见创建、编译和运行 Hello World Java 程序。
6. 如果您有兴趣创建更多自己的 Java 应用程序，请阅读下列主题:
   - 创建和编辑 Java 源文件显示了创建和编辑 Java 源文件的三种不同方法。

- 在 iSeries 服务器上下载和安装 Java 包帮助您更有效地使用 Java 包。它提供了有关带有图形用户界面（GUI）的包、集成文件系统和区分大小写以及 ZIP 文件处理和 JAR 文件处理的详细信息。
- 发行版间的兼容性提供了关于从一个发行版到另一个发行版的兼容性的信息。

安装 IBM Developer Kit for Java

通过安装 IBM Developer Kit for Java(TM)，您可以在 iSeries 服务器上创建和运行 Java 程序。

要安装 IBM Developer Kit for Java，请执行下列步骤:

1. 在命令行上输入“转至许可程序”（GO LICPGM）命令。
2. 选择选项 11（安装许可程序）。
3. 对许可程序（LP）5722-JV1 *BASE 选择选项 1（安装），并选择与您想安装的 Java Development Kit（JDK）相匹配的选项。如果您想要安装的选项未显示在列表中，则可以通过在选项字段中输入选项 1（安装）将它添加到列表中。在许可程序字段中输入 5722JV1，并在产品选项字段中输入选项号。

   注意: 一次可以安装多个选项。

在 iSeries 服务器上安装 IBM Developer Kit for Java 之后，您就可以选择定制系统。

有关 IBM Developer Kit for Java 入门的信息，参见运行您的第一个 Hello World Java 程序。

使用“恢复许可程序”命令来安装许可程序

*安装许可程序*屏幕中列示的程序是一开始使用服务器时 LICPGM 安装所支持的那些程序。有时候，可使用一些新程序，但没有将它们作为服务器上的许可程序列示出来。如果要安装的程序发生这种情况，则必须使用“恢复许可程序”（RSTLICPGM）命令来进行安装。

要使用“恢复许可程序”（RSTLICPGM）命令来安装许可程序，请执行下列步骤:

1. 将包含许可程序的磁带或 CD-ROM 放入适当的驱动器。
2. 在 iSeries 命令行上，输入:

   RSTLICPGM

   并按**执行**键。

   将出现*恢复许可程序（RSTLICPGM）*屏幕。
3. 在*产品*字段中，输入要安装的许可程序的标识号。
4. 在*设备*字段中，指定安装设备。

   注意: 如果是从磁带机安装，则设备标识通常具有 **TAP*XX*** 格式，其中 ***XX*** 是编号，如 **01**。
5. 对*恢复许可程序*屏幕中的其它参数保留缺省设置。按**执行**键。
6. 将出现更多的参数。并保留这些缺省设置。按**执行**键。程序将开始安装。

当许可程序完成安装时，将再次出现*恢复许可程序*屏幕。

支持多个 Java Development Kit

iSeries 服务器支持多个 Java Development Kit（JDK）和 Java 2 SDK（J2SDK）Standard Edition。iSeries 服务器支持同时使用多个 JDK，但只能通过多个 Java 虚拟机使用。单个 Java 虚拟机只能运行一个指定的 JDK。

查找您正在使用或想要使用的 JDK，并选择要安装的协调选项。一次可以安装多个 JDK。java.version 系统属性确定要运行的 JDK。在启动并运行 Java 虚拟机之后，更改 java.version 系统属性便不起作用了。

注意: 在 V5R2 中，选项 1（JDK 1.1.6）和 2（JDK 1.1.7）不再可用。不能安装或使用它们。

选项	JDK	java.home	java.version
3	1.2	» /QIBM/ProdData/Java400/jdk12/ «	1.2
4	1.1.8	/QIBM/ProdData/Java400/jdk118/	1.1.8
5	1.3	/QIBM/ProdData/Java400/jdk13/	1.3
» 6	1.4	/QIBM/ProdData/Java400/jdk14/	1.4 «

» **注意**: V1.3 与 Java 2 SDK（J2SDK）Standard Edition V1.3 相同。 «

例如，以下是您安装的选项以及输入的命令所产生的结果。

安装	输入	结果
选项 3（1.2）	`java Hello`	将运行 J2SDK Standard Edition V1.2。
选项 4（1.1.8）	`java Hello`	由于只安装了一个 JDK，而那就是缺省 JDK，所以将运行 JDK 1.1.8。
选项 4（1.1.8）和选项 3（1.2）	`java Hello`	由于 J2SDK Standard Edition V1.2 具有较高的版本，所以将运行它。
» 安装了全部四个选项	`java Hello`	将运行 J2SDK Standard Edition V1.3。 «
选项 3（1.2）和选项 5（1.3）	`java Hello`	由于 J2SDK Standard Edition V1.3 具有较高的版本，所以将运行它。
选项 4（1.1.8）和选项 5（1.3）	`java -Djava.version=1.1.8 Hello`	由于指定了 JDK 1.1.8，所以将运行它。
» 选项 5（1.3）和选项 6（1.4）	`java Hello`	将运行 J2SDK Standard Edition V1.3。尽管 1.4 具有较高的编号，但 1.3 优先于 1.4。 «

注意: 如果仅安装一个 JDK，则缺省 JDK 就是所安装的 JDK。 » 如果安装多个 JDK，则以下优先顺序确定缺省 JDK:

1. 选项 5（1.3）
2. 选项 3（1.2）
3. 选项 6（1.4）
4. 选项 4（1.1.8） «

安装 IBM Developer Kit for Java 的扩展

扩展是可以用来扩展核心平台功能的 Java(TM) 类的包。扩展封装在一个或多个 ZIP 文件或 JAR 文件中，它们通过扩展类装入程序装入 Java 虚拟机。

扩展机制允许 Java 虚拟机按照虚拟机使用系统类的方式来使用扩展类。当 J2SDK V1.2 或更高版本或者“Java 2 运行时环境标准版” V1.2 和更高版本中尚未安装某些扩展时，扩展机制还提供了从指定的“统一资源定位器”（URL）检索这些扩展的方法。

» iSeries 服务器附带交付了一些扩展 JAR 文件。 « 如果您想要安装这些扩展之一，请输入以下命令:

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

其中，extensionToInstall.jar 是包含您想要安装的扩展的 ZIP 或 JAR 文件的名称。

注意: 不是由 IBM 提供的扩展的 JAR 文件可能放在 /QIBM/UserData/Java400/ext 目录中。

当创建指向 /QIBM/UserData/Java400/ext 目录中的扩展的链接或在该目录中添加文件时，对于*正在 iSeries 服务器上运行的每个 Java 虚拟机*而言，扩展类装入程序所搜索的文件列表都将发生更改。如果不想影响 iSeries 服务器上的其它 Java 虚拟机的扩展类装入程序，但是您仍想创建指向扩展的链接或安装不是由 IBM 随 iSeries 服务器交付的扩展，则请执行下列步骤:

1. 创建用来安装扩展的目录。
   从 iSeries 命令行使用“创建目录”（MKDIR）命令或者从 Qshell Interpreter 使用 mkdir 命令。
2. 将扩展 JAR 文件放到所创建的目录中。
3. 将新目录添加至 java.ext.dirs 属性。
   可以通过从 iSeries 命令行使用 JAVA 命令的 PROP 字段来将新目录添加至 java.ext.dirs 属性。

» 如果新目录的名称是 /home/username/ext，扩展文件的名称是 extensionToInstall.jar，并且 Java 程序的名称是 Hello，则输入的命令应该类似于:

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

CPY OBJ('/productA/extensionToInstall.jar') TODIR('/home/username/ext') or
copy the file to /home/username/ext using FTP (file transfer protocol).

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

在 iSeries 服务器上下载和安装 Java 包

要在 iSeries 服务器上更有效地下载、安装和使用 Java(TM) 包，请参见下列各项:

- 带有图形用户界面的包
- 区分大小写与集成文件系统
- ZIP 文件处理和 JAR 文件处理
- Java 扩展框架

带有图形用户界面的包

配合图形用户界面（GUI）使用的 Java 程序要求使用具有图形显示功能的显示设备。例如，可以使用个人计算机、技术工作站或网络计算机。iSeries 服务器提供了 Remote Abstract Window Toolkit（AWT）功能。这种功能通过使用适当显示设备上的功能全面的图形能力来在 iSeries 服务器上运行应用程序，而这些显示设备是通过“传输控制协议／网际协议”（TCP/IP）相连的。有关更特定的安装、设置和整体使用信息，参见设置 Remote Abstract Window Toolkit。

区分大小写与集成文件系统

集成文件系统提供了两种文件系统：区分文件名大小写的文件系统和不区分文件名大小写的文件系统。QOpenSys 便是集成文件系统中区分大小写文件系统的一个示例。而“根”（“/”）便是不区分大小写的文件系统的一个示例。有关集成文件系统的更多信息，参见集成文件系统主题中的“文件系统”信息。

» 尽管可以在不区分大小写的文件系统中定位 JAR 或类，但 Java 也仍然是区分大小写的语言。虽然 wrklnk '/home/Hello.class' 和 wrklnk '/home/hello.class' 将生成相同的结果，但 JAVA CLASS(Hello) 和 JAVA CLASS(hello) 却调用不同的类。 «

ZIP 文件处理和 JAR 文件处理

ZIP 文件和 JAR 文件都包含一组 Java 类。当对这些文件之一使用创建 Java 程序（CRTJVAPGM）命令时，将对类执行验证，将其转换成内部机器格式，并且，如果指定的话，还转换成 iSeries 机器代码。可以将 ZIP 文件和 JAR 文件与其它任何个别类文件一样对待。当某种内部机器格式与这些文件之一相关联时，这种格式就会保持与该文件相关联。内部机器格式在将来运行时用来代替类文件，以改进性能。如果您不肯定当前 Java 程序是否与类文件或 JAR 文件相关联，则请使用显示 Java 程序（DSPJVAPGM）命令来显示关于 iSeries 服务器上的 Java 程序的信息。

在 IBM Developer Kit for Java 的前发行版中，如果以任何方式更改了 JAR 文件或 ZIP 文件，则相连接的 Java 程序将变得不可使用，所以必须重建 Java 程序。这种情况不再存在。在许多情况下，即使您更改了 JAR 文件或 ZIP 文件，Java 程序也仍然有效，并且您不必重建它。如果作了部分更改，例如，更新了 JAR 文件中的单个类文件，则您只需要重建 JAR 文件中受影响的类文件。

在对 JAR 文件进行大多数典型更改之后，Java 程序将保持与该 JAR 文件连接。例如，在下列情况下，这些 Java 程序将保持与 JAR 文件相连:

- 使用 ajar 工具更改或重建 JAR 文件。
- 使用 jar 工具更改或重建 JAR 文件。
- 使用 0S/400 COPY 命令或 Qshell `cp` 实用程序替换 JAR 文件。

如果通过 iSeries Access for Windows 或从个人计算机（PC）上的映射驱动器访问集成文件系统中的 JAR 文件，则在下列情况下，Java 程序将保持与 JAR 文件相连接:

- 将另一个 JAR 文件拖放到现有集成文件系统 JAR 文件上。
- 通过使用 jar 工具更改或重建了集成文件系统 JAR 文件。
- 使用 PC 复制命令替换集成文件系统 JAR 文件。

当更改或替换 JAR 文件时，与之相连接的 Java 程序便不再是当前程序了。

有一种例外情况，在该情况下，Java 程序不保持与 JAR 文件相连接。如果使用文件传送协议（FTP）来替换 JAR 文件，则会破坏连接的 Java 程序。例如，如果使用 FTP `put` 命令来替换 JAR 文件，便会发生这种情况。

有关 JAR 文件的性能特征的详细信息，参见运行时性能。

Java 扩展框架

在 Java 2 SDK Standard Edition V1.2 和更高版本中，扩展是可以用来扩展核心平台功能的 Java 类包。扩展或应用程序封装在一个或多个 JAR 文件中。扩展机制允许 Java 虚拟机按照虚拟机使用系统类的方式来使用扩展类。当 Java Development Kit（JDK）或 “Java 2 运行时环境标准版” 中尚未安装扩展时，扩展机制还为您提供了从指定的 URL 检索这些扩展的方法。

有关安装扩展的信息，参见安装 IBM Developer Kit for Java 的扩展。

运行您的第一个 Hello World Java 程序

可以通过这些方法之一来启动并运行 Hello World Java(TM) 程序:

1. 可以仅仅是运行 IBM Developer Kit for Java 附带交付的 Hello World Java 程序。

   要运行包括的程序，请执行下列步骤:

   a. 输入 “转至许可程序”（GO LICPGM）命令检查是否已安装 IBM Developer Kit for Java。然后，选择选项 10（显示已安装的许可程序）。验证是否已将许可程序 5722-JV1 *BASE 和至少一个选项列示为 “已安装”。

   b. 在 “iSeries 主菜单” 命令行上输入 `java Hello`。按 “执行” 键来运行 Hello World Java 程序。

c. 如果已正确安装 IBM Developer Kit for Java，则 `Hello World` 将出现在“Java 外壳程序屏幕”中。按 F3 键（退出）或 F12 键（退出）以返回命令输入屏幕。

d. 如果 Hello World 类不运行，则请检查安装是否已成功完成，或查看获取 IBM Developer Kit for Java 支持以了解服务信息。

2. 您也可以运行自己的 Hello Java 程序。有关如何创建自己的 Hello Java 程序的更多信息，参见创建、编译和运行 Hello World Java 程序。

将网络驱动器映射至 iSeries 服务器

要将网络驱动器映射至 iSeries 服务器，请确保已在服务器上和工作站上安装 iSeries Access for Windows。有关如何安装和配置 iSeries Access for Windows 的更多信息，参见安装 iSeries Access for Windows。

在可以映射网络驱动器之前，必须为 iSeries 服务器配置连接。

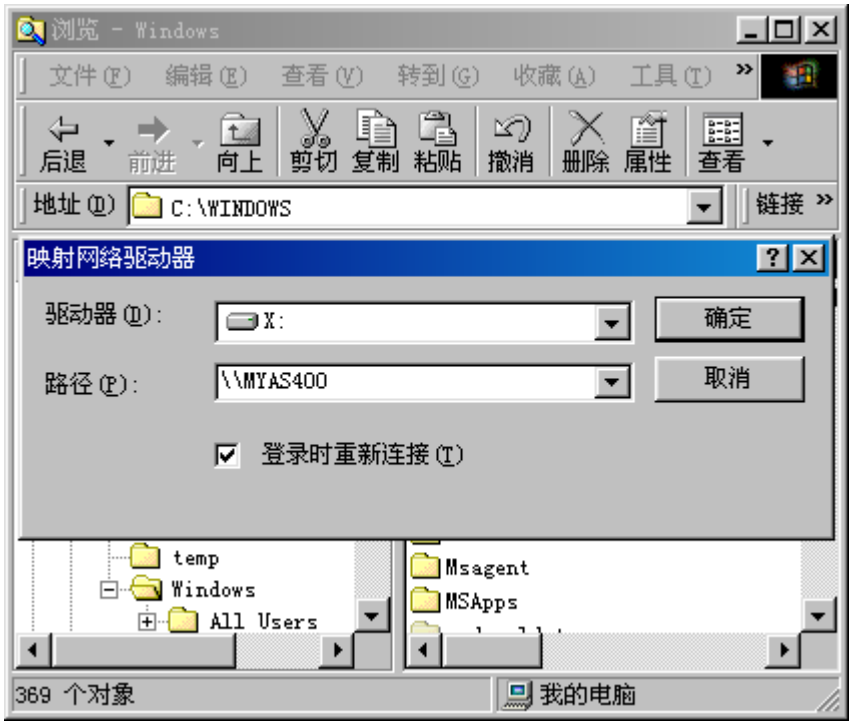
要映射网络驱动器，请遵循下面这些步骤:

1. Open Windows(R) 资源管理器:
   a. 右键单击 Windows 任务栏上的**开始**按钮。
   b. 单击菜单中的**资源管理器**。
2. 从**工具**菜单中选择**映射网络驱动器**。

3. 选择要用来连接至 iSeries 服务器的驱动器。
4. 输入服务器的路径名。例如:

   \\MYSERVER

   其中，***MYSERVER*** 是 iSeries 服务器的名称。



5. 如果**登录时重新连接**框是空白的，则选择此框。
6. 单击**确定**以完成。

映射的驱动器现在出现在“Windows 资源管理器”的**所有文件夹**部分中。 «

在 iSeries 服务器上创建目录

» 必须在 iSeries 服务器上创建可用来保存 Java(TM) 应用程序的目录。可以通过两种方法做到这一点:

- 使用 iSeries 导航器来创建目录
  如果已安装 iSeries Access for Windows，则选择此选项。如果计划使用“iSeries 导航器”来编译、优化和运行 Java 程序，则必须选择此选项以确保将程序保存在正确的位置中才能执行这些操作。
- 使用命令输入行来创建目录
  如果未安装 iSeries Access for Windows，则选择此选项。

有关“iSeries 导航器”的信息，包括安装信息，参见 iSeries 导航器入门。 «

使用命令输入行来创建目录

要在 iSeries 服务器上创建目录，请执行下列步骤:

1. 注册到 iSeries 服务器上。
2. 在命令行上，输入:

   ```
   CRTDIR DIR('/mydir')
   ```

   其中 *mydir* 是正在创建的目录的名称。

   按**执行**键。

   屏幕底部将出现一条消息，指出“**已创建目录**”。

使用 iSeries 导航器来创建目录

» 要在 iSeries 服务器上创建目录，请执行下列步骤:

1. 打开 “iSeries 导航器”。
2. 在**我的连接**窗口中双击服务器的名称以进行注册。
   如果您的服务器未列示在**我的连接**窗口中，则请执行下列步骤来添加它:
   a. 单击**文件 —> 添加连接...**。
   b. 在**系统**字段中输入服务器的名称。
   c. 单击**下一步**。
   d. 如果还没有输入的话，在**使用缺省用户标识，根据需要提示**字段中输入 “用户标识”。
   e. 单击**下一步**。
   f. 单击**验证连接**。这将确认是否可以连接至该服务器。
   g. 单击**完成**。
3. 在要使用的连接下面展开文件夹。定位名为**文件系统**的文件夹。如果您看不到此文件夹，则表示在 “iSeries 导航器” 的安装期间未选择用于安装 “文件系统” 的选项。必须通过选择**开始 —> 程序 —> IBM iSeries Access for Windows —> 选择性安装**来安装 “iSeries 导航器” 的 “文件系统” 选项。
4. 展开**文件系统**文件夹并定位**集成文件系统**文件夹。
5. 展开**集成文件系统**文件夹，然后展开**根**文件夹。通过展开**根**文件夹，您将看到与在 iSeries 命令行上执行 WRKLNK (’/’) 命令相同的结构。
6. 右键单击要在该处添加子目录的文件夹。选择**新建文件夹**并输入要创建的子目录的名称。«

创建、编译和运行 HelloWorld Java 程序

在您开始变得熟悉 IBM Developer Kit for Java 时，最好以创建简单的 Hello World Java(TM) 程序作为入门。

要创建、编译和运行您自己的 Hello World Java 程序，请执行下列步骤:

1. » 将网络驱动器映射至 iSeries 服务器。
2. 在 iSeries 服务器上为 Java 应用程序创建目录。«
3. 在集成文件系统中，将源文件创建为 “美国信息交换标准代码”（ASCII）文本文件。» 可以使用集成开发环境（IDE）产品或基于文本的编辑器（如 Windows(R) 的 “记事本”）来编码 Java 应用程序。«
   a. 将该文本文件命名为 `HelloWorld.java`。有关如何创建和编辑文件的更多信息，参见创建和编辑 Java 源文件。
   b. 确保该文件包含以下源代码:

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

4. 编译源文件。
   a. 输入 “使用环境变量”（WRKENVVAR）命令来检查 CLASSPATH 环境变量。如果 CLASSPATH 变量不存在，则添加它并将其设置为 “.”（当前目录）。如果 CLASSPATH 变量已存在，则确保 “.” 位于路径名列表的开头。有关 CLASSPATH 环境变量的详细信息，参见 Java 类路径。
   b. 输入 “启动 Qshell”（STRQSH）命令来启动 Qshell Interpreter。
   c. 使用更改目录（cd）命令来将当前目录更改为包含 `HelloWorld.java` 文件的集成文件系统目录。
   d. 输入 `javac`，后跟在磁盘上保存文件时所使用的名称。例如，输入 `javac HelloWorld.java`。

5. » 设置集成文件系统中的类文件的文件权限。
6. 优化 Java 应用程序。
   a. 在 *QSH 命令输入*行上，输入:

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

      其中，*mydir* 是用于保存 Java 应用程序的目录的路径名，*myclass* 是经过编译的 Java 应用程序的名称。

      注意: 最高可指定优化级别 40。优化级别 40 可提高 Java 应用程序的效率，但也会限制调试能力。在开发 Java 应用程序的早期阶段，您可能想将优化级别设置为 20 以便可以更容易地调试应用程序。有关更多信息，参见 CRTJVAPGM 命令和 OPTIMIZE 参数。
   b. 按**执行**键。

      将出现一条消息，指出已经为类创建了 Java 程序。«
7. 运行类文件。
   a. 确保 Java 类路径设置正确。
   b. 在 Qshell 命令行上，输入后面跟有 `HelloWorld` 的 `java` 以使用 Java 虚拟机来运行 `HelloWorld.class`。例如，输入 `java HelloWorld`。还可在 iSeries 服务器上使用“运行 Java”（RUNJVA）命令来运行 `HelloWorld.class`。
   c. 如果各项输入正确，则“`Hello World`”将打印在屏幕上。» 将出现外壳程序提示（缺省情况下为 $），指示 Qshell 已经为另一个命令作好准备。«
   d. 按 F3（退出）或 F12（断开连接）键以返回命令输入屏幕。

» 还可使用“iSeries 导航器”（这是一个用于在 iSeries 服务器上执行任务的图形用户界面）来很方便地编译、优化和运行 Java 应用程序，有关指示信息，参见通过使用 iSeries 导航器来使用 Java 应用程序。有关“iSeries 导航器”的更多信息，包括安装信息，参见 iSeries 导航器入门。 «

创建和编辑 Java 源文件

您可以通过许多种方法来创建和编辑 Java(TM) 源文件:

- 使用 iSeries Access for Windows。
- 在工作站上。
- 使用 EDTF。
- 使用源程序输入实用程序。

使用 iSeries Access for Windows

Java 源文件是 iSeries 服务器上的集成文件系统中的“美国信息交换标准代码”（ASCII）文本文件。

您可以使用 iSeries Access for Windows 和基于工作站的编辑器来创建和编辑 Java 源文件。

在工作站上

您可以在工作站上创建 Java 源文件。然后，通过使用文件传送协议（FTP）将该文件传送至集成文件系统。

要在工作站上创建和编辑 Java 源文件:

1. 使用您选择的编辑器来在工作站上创建 ASCII 文件。
2. 使用 FTP 连接至 iSeries 服务器。
3. 将源文件作为二进制文件传送至集成文件系统中的目录中，以使该文件保持 ASCII 格式。

使用 EDTF

可以使用 EDTF CL 命令来从任何文件系统编辑文件。这是一个与用于编辑流文件或数据库文件的“源程序输入实用程序”（SEU）相类似的编辑器。有关信息，参见 EDTF CL 命令。

使用源程序输入实用程序

您可以通过使用“源程序输入实用程序”（SEU）来将 Java 源文件作为文本文件来创建。

要使用 SEU 来将 Java 源文件作为文本文件来创建，请执行下列步骤:

1. 使用 SEU 来创建一个源文件成员。
2. 使用“复制为流文件”（CPYTOSTMF）命令来将该源文件成员复制为集成文件系统流文件，同时将数据转换为 ASCII。

如果需要对源代码作更改，可使用 SEU 来更改数据库成员，并再次复制该文件。

有关存储文件的信息，参见集成文件系统中的文件。

通过使用 iSeries 导航器来使用 Java 应用程序

“iSeries 导航器”允许通过简单的点击操作来编译、优化和运行 Java(TM) 应用程序。

设置需求

要使用“iSeries 导航器”来使用 Java 应用程序，应确保符合下列条件:

- 必须在工作站上安装“iSeries 导航器”（它是 iSeries Access for Windows 的一部分）。如果尚未安装 iSeries Access for Windows，则请查看 iSeries 导航器入门以了解与下载相关的信息。
- 必须将 Java 应用程序保存在 iSeries 服务器上的特定目录中。用于保存 Java 应用程序的正确路径如下:

  myserver -> 文件系统 -> 集成文件系统 -> 根 -> 主目录 -> mydir

  其中 **myserver** 是 iSeries 服务器的名称，而 **mydir** 是在其中保存 Java 应用程序的目录的名称。有关创建要在其中保存 Java 应用程序的目录的更详细信息，参见使用 iSeries 导航器来创建目录。

使用 iSeries 导航器来编译 Java 应用程序

要编译 Java 应用程序，请遵循下面这些步骤:

1. 右键单击 **myfile.java**，其中 **myfile** 是 Java 应用程序的名称。
2. 选择**编译 Java 文件**。
3. 在新窗口中，选择 JDK 版本。如果先前已根据本指南中给出的指示信息设置了类路径，则不必在此窗口中指定类路径。
4. 单击**确定**。

如果在程序中找到了任何错误，则将打开带有错误列表的窗口。否则，“iSeries 导航器”窗口的底部将出现一条消息，指出**已编译 Java 文件，第 1 个，共 1 个**。将创建名为 **myfile.class** 的新文件。

使用 iSeries 导航器来优化和运行 Java 应用程序

要优化和运行 Java 应用程序，请遵循下面这些步骤:

1. 右键单击 **myfile.class**。
2. » 选择**相关联的 Java 程序**，然后单击**运行...** 以运行 Java 程序。«
3. 单击**高级**。选择期望的优化级别。

注意: 最高可指定优化级别 40。优化级别 40 可提高 Java 应用程序的效率，但也会限制调试能力。在开发 Java 应用程序的早期阶段，您可能想将优化级别设置为 20 以便可以更容易地调试应用程序。有关优化的更多信息，参见优化级别。还有一个名为 JIT 的选项，JIT 表示“及时”编译器。JIT 根据您的需要编译代码，这比直接处理效率更高。有关 JIT 的更多信息，参见选择运行 Java 程序时要使用的方式。

4. 单击**确定**以关闭“高级选项”窗口。
5. 单击**确定**以运行 Java 程序。

程序的输出将显示在新窗口中。当程序运行完成时，将出现一条消息，指出 **Java 程序已完成**。

有关“iSeries 导航器”功能的更多信息，参见“iSeries 导航器”屏幕上的**帮助**菜单。

为 IBM Developer Kit for Java 定制 iSeries 服务器

在 iSeries 服务器上安装 IBM Developer Kit for Java(TM) 之后，您就可选择定制服务器。

时区配置

如果 Java 程序对每一天的时间敏感，则可能需要配置时区。

如果“世界协调时偏移”（QUTCOFFSET）系统值设置为其缺省值（+00:00），则 Java 使用 iSeries 400time 作为当前时间。user.timezone Java 系统属性设置的缺省值为 UTC。

在下列任何情况下都需要 QUTCOFFSET 系统值和更新过的语言环境:

- 如果未将 QUTCOFFSET 设置为其缺省值（因而对时区敏感）。
- 如果 Java 代码期望 user.timezone 系统属性缺省为不同于 UTC 的值。
- 如果在运行 `java` 命令时指定了 user.timezone Java 系统属性。

语言环境的 LOC_TOD 类别包含 tname 字段，必须将其设置成与您的时区相匹配的值。有关如何创建语言环境和格式化 tname 字段的详细信息，参见 OS/400 全球化。

系统属性配置

Java 系统属性确定 Java 程序的运行环境。它们与 OS/400 中的系统值或环境变量类似。许多属性是在 Java 虚拟机启动时设置的。您可以选择使用系统属性缺省值，也可以通过执行这些步骤之一来指定您自己的一组缺省属性值:

1. 在 /QIBM/UserData/Java400 中提供一个名为 SystemDefault.properties 的文件。您在此文件中指定的属性值将覆盖 IBM 提供的系统缺省值。此文件对 iSeries 服务器上运行的所有 Java 虚拟机设置缺省系统属性。
2. 或者，在您自己的 user.home 路径中放置一个 SystemDefault.properties 文件。此文件及其包含的属性将覆盖 /QIBM/UserData/Java400/SystemDefault.properties 中的属性。此文件对运行的所有 Java 虚拟机设置缺省系统属性。

有关 IBM Developer Kit for Java 入门的信息，参见运行您的第一个 Hello World Java 程序。

Java 类路径

在运行时，Java(TM) 虚拟机使用 Java 类路径来查找类。Java 命令和工具也使用类路径来定位类。缺省系统类路径、CLASSPATH 环境变量和类路径命令参数全都确定在查找特定类时要搜索的目录。

注意: 在 Java 2 Software Development Kit（J2SDK）Standard Edition V1.2 和更高版本中，java.ext.dirs 属性确定所装入的扩展的类路径。有关更多信息，参见安装 IBM Developer Kit for Java 的扩展。

缺省系统类路径由系统定义，不能更改。在 iSeries 服务器上，缺省类路径指定在何处查找作为 IBM Developer Kit 一部分的类、Remote Abstract Window Toolkit（AWT）以及其它系统类。

要查找系统上的任何其它类，必须使用 CLASSPATH 环境变量或类路径参数来指定要搜索的类路径。工具或命令上使用的类路径参数覆盖 CLASSPATH 环境变量中指定的值。

可通过使用“使用环境变量”（WRKENVVAR）命令来使用 CLASSPATH 环境变量。在 WRKENVVAR 屏幕中，可以添加或更改 CLASSPATH 环境变量。“添加环境变量”（ADDENVVAR）命令和“更改环境变量”（CHGENVVAR）命令可添加或更改 CLASSPATH 环境变量。

CLASSPATH 环境变量的值是路径名列表，这些路径名用冒号（:）分隔，在查找特定类时将要搜索这些路径。路径名是零个或多个目录名的序列。这些目录名后面跟随着要在集成文件系统中搜索的目录名、ZIP 文件或 JAR 文件。路径名的各个组件由斜杠（/）字符分隔。使用句点（.）来指示当前工作目录。

可以使用导出实用程序来在 Qshell 环境中设置 CLASSPATH 变量（该实用程序可以在 Qshell Interpreter 中使用）。

这些命令将 CLASSPATH 变量添加到 Qshell 环境并将它设置为值 ″.:/myclasses.zip:/Product/classes″。

- 此命令在 Qshell 环境中设置 CLASSPATH 变量:

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

- 此命令从命令行设置 CLASSPATH 变量:

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

J2SDK 搜索类的方式与 JDK 1.1.x 不同。JDK 1.1.x 首先搜索系统类路径，然后再搜索用户指定的任何类路径。J2SDK 首先搜索引导类路径，接着搜索扩充功能目录，然后再搜索类路径。

因此，当使用前面的示例代码时，JDK 1.1.x 的搜索次序为:

1. 缺省系统类路径,
2. 当前工作目录,
3. 位于“根”（/）文件系统中的 myclasses.zip 文件,
4. “根”（/）文件系统中“产品”目录中的类目录。

当使用以上示例时，J2SDK 的搜索次序为:

1. 引导类路径，它在 sun.boot.class.path 属性中,
2. 扩展目录，位于 java.ext.dirs 属性中,
3. 当前工作目录,
4. 位于“根”（/）文件系统中的 myclasses.zip 文件,
5. “根”（/）文件系统中“产品”目录中的类目录。

当进入 Qshell 环境时，CLASSPATH 变量便被设置为环境变量。类路径参数指定路径名列表。它的语法与 CLASSPATH 环境变量相同。这些工具和命令上有一个类路径参数:

- Qshell 中的 java 命令
- javac 工具
- javah 工具
- javap 工具
- javadoc 工具
- rmic 工具

- “运行 Java”（RUNJVA）命令

有关这些命令的更多信息，参见 IBM Developer Kit for Java 的命令和工具。如果将类路径参数配合任何这些命令或工具使用，则它忽略 CLASSPATH 环境变量。

可以通过使用 java.class.path 属性来覆盖 CLASSPATH 环境变量。通过使用 SystemDefault.properties 文件，可以更改 java.class.path 属性以及其它属性。SystemDefault.properties 文件中的值覆盖 CLASSPATH 环境变量。有关 SystemDefault.properties 文件的信息，参见 SystemDefault.properties 文件。

在 JDK 1.1.x 中，os400.class.path.system 属性也影响当查找类时搜索的目录。此属性可以包含这三个值之一：PRE、POST 或 NONE。在缺省情况下，此属性设置为 PRE，这导致在搜索路径之前搜索缺省系统类路径。此路径由 CLASSPATH 环境变量或类路径参数指定。如果将 os400.class.path.system 属性设置为 POST，则在搜索用户指定的任何类路径之后搜索缺省系统类路径。如果使用值 NONE，则完全不搜索缺省类路径，而只搜索用户指定的类路径。

» 在 J2SDK 中，可使用 -Xbootclasspath 选项，且效果相同。-Xbootclasspath/a:*path* 将 *path* 附加至缺省引导类路径，/p:*path* 将 *path* 附加至引导类路径的开头，/r:*path* 用 *path* 替换引导类路径。 «

注意：指定 NONE 或 POST 时请务必小心，这是因为如果系统类找不到或错误地被用户定义类替换，便会发生不可预测的结果。因此，应该允许在搜索用户指定的任何类路径之前搜索系统缺省类路径。

有关如何确定 Java 程序的运行环境的信息，参见 Java 系统属性。

有关更多信息，参见程序和 CL 命令 API 或集成文件系统。

Java 系统属性

Java(TM) 系统属性确定 Java 程序的运行环境。这些属性与 OS/400(R) 中的系统值或环境变量类似。许多属性是在 Java 虚拟机启动时设置的。

要查看此发行版支持的系统属性，请链接至您选择的版本以了解详细信息:

- Java Development Kit（JDK）1.1.8
- Java 2 Software Development Kit（J2SDK）Standard Edition » V1.2、1.3 和 1.4。 «

SystemDefault.properties 文件

SystemDefault.properties 文件是标准 Java(TM) 属性文件。可以在 SystemDefault.properties 文件中指定缺省属性。您的主目录中的 SystemDefault.properties 文件具有比 /QIBM/UserData/Java400 目录中的 SystemDefault.properties 文件更高的优先级。您的主目录中的 SystemDefault.properties 属性文件中设置的属性只用于由您或指定了 user.home = /YourUserHome/ 属性的用户启动的 Java 虚拟机。

可以在 SystemDefault.properties 文件中指定属性的值，就象在任何 Java 属性文件中所作的那样。

示例：SystemDefault.properties 文件

注意：请阅读代码示例不保证声明以了解重要的法律信息。

```
#Comments start with pound sign
#this means always run with JDK 1.3
java.version=1.3
#set my special property
myown.propname=6
```

Java Development Kit（JDK）1.1.8 的 Java 系统属性

Java(TM) 系统属性确定 Java 程序的运行环境。这些属性与 OS/400(R) 中的系统值或环境变量类似。许多属性是在 Java 虚拟机启动时设置的。

在 JDK 1.1.8 中，将系统属性设置为这些系统缺省值。

系统属性	系统缺省值
awt.toolkit	com.ibm.rawt.client.CToolkit
file.encoding.pkg	sun.io
file.separator	/（斜杠）
java.class.version	45.3
java.home	有关详细信息，参见支持多个 Java Development Kit（JDK）。
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os400.class.path.rawt	0
os400.class.path.security.check	20 有效值: 0 — 不执行安全性检查 10 — 等价于 RUNJVA CHKPATH(*IGNORE) 20 — 等价于 RUNJVA CHKPATH(*WARN) 30 — 等价于 RUNJVA CHKPATH(*SECURE)
os400.class.path.tools	0
os400.create.type	direct 有效值: interpret — 等价于 RUNJVA OPTIMIZE(*INTERPRET) 和 INTERPRET(*OPTIMIZE)，或者 INTERPRET(*YES) direct — 其它值
os400.defineClass.optLevel	20
os400.enbpfrcol	0 有效值: 0 — 等价于 CRTJVAPGM ENBPFRCOL(*NONE) 1 — 等价于 CRTJVAPGM ENBPFRCOL(*ENTRYEXIT) 7 — 等价于 CRTJVAPGM ENBPFRCOL(*FULL)
os400.interpret	0 有效值: 0 — 等价于 CRTJVAPGM INTERPRET(*NO) 1 — 等价于 CRTJVAPGM INTERPRET(*YES)

系统属性	系统缺省值
os400.optimization	10 有效值: 0 — 等价于 CRTJVAPGM OPTIMIZE(*INTERPRET) 10 — 等价于 CRTJVAPGM OPTIMIZE(10) 20 — 等价于 CRTJVAPGM OPTIMIZE(20) 30 — 等价于 CRTJVAPGM OPTIMIZE(30) 40 — 等价于 CRTJVAPGM OPTIMIZE(40)
os400.run.mode	program_create_type 有效值: interpret — 等价于 RUNJVA OPTIMIZE(*INTERPRET) 和 INTERPRET(*OPTIMIZE)，或者 INTERPRET(*YES) program_create_type — 其它情况
os400.stdin.allowed	0
os400.verify.checks.disable	65535 os400.class。此系统属性值是表示一个或多个数值之和的字符串。有关这些值的列表，参见 os400.verify.checks.disable 数值。
path.separator	:（冒号）

这组系统属性是根据其它系统信息设置的。

系统属性	描述
file.encoding	将 OS/400 作业 CCSID 映射至相对应的 ISO ASCII CCSID。并且，将 file.encoding 值设置为表示 ISO ASCII 编码字符集标识符（CCSID）的 Java 值。请查看 file.encoding 值和 iSeries CCSID 以获取一个表，该表显示了可能的 file.encoding 值与最匹配的 iSeries 编码字符集标识符（CCSID）之间的关系。
java.class.path	用来定位类的路径。缺省为缺省系统类路径，后面跟有用户指定的类路径。可以使用 os400.class.path.system 系统属性来更改 java.class.path 系统属性。
java.compiler	指定代码是使用“及时”（JIT）编译器（jitc）编译，还是既使用 JIT 编译器编译又直接处理（jitc_de）。
java.version	确定要运行的 Java Development Kit（JDK）。 如果指定了未安装的 JDK，则会发出错误消息。如果不指定 JDK，则最新的 JDK 变为缺省 JDK。如果只安装了一个 JDK，则它便是缺省 JDK。有关版本详细信息，参见支持多个 JDK。
os.version	从“检索产品信息”应用程序接口（API）获取 OS/400 发行版级别。
os400.CertificateContainer	指示 Java 安全套接字层（SSL）支持对启动的 Java 程序和指定的属性使用指定的证书容器。如果指定 os400.secureApplication 系统属性，则将忽略此系统属性。例如，输入 `-Dos400.certificateContainer=/home/username/mykeyfile.kdb` 或集成文件系统中的任何其它密钥文件。

系统属性	描述
os400.CertificateLabel	可以将此系统属性与 os400.CertificateContainer 系统属性一起指定。此属性允许您选择想要让安全套接字层（SSL）使用指定容器中的哪个证书。例如，输入 `-Dos400.certificateLabel=myCert`，其中 myCert 是您在创建或导入证书时通过“数字证书管理器”（DCM）对证书指定的标号名。
os400.child.stdio.convert	允许控制 Java 中标准输入、标准输出和标准错误的数据转换。在缺省情况下，Java 虚拟机中执行数据转换来在 ASCII 与 EBCDIC 之间来回转换数据。可以使用此属性来打开或关闭这些转换，这将影响此进程使用 runtime.exec() 方法启动的任何子进程。参见缺省值。
os400.class.path.system	PRE（此值导致在构造 java.class.path 系统属性时，将 os400 缺省系统类路径附加到类路径的用户指定部分的前面）。其它值为 POST（将系统缺省类附加到类路径的用户指定部分后面）和 NONE（只使用用户指定的类路径）。缺省值为 PRE。 此属性不区分大小写。例如，可以指定 NONE、none 或 noNe。然而，属性名区分大小写。例如，不能指定 OS400.CLASS.PATH.SYSTEM。为了避免潜在的问题，不应更改此系统属性。
» os400.file.create.auth, os400.dir.create.auth	这些属性指定对文件和目录指定的权限。指定不带任何值或带有不支持的值的属性将导致公用权限为 *NONE。 可指定 os400.file.create.auth=RWX 或 os400.dir.create.auth=RWX，其中 R = 读，W = 写，X = 执行。这些权限的任何组合都有效。 «
os400.file.io.mode	当您指定 TEXT 而不是缺省值 BINARY 时，如果文件的 CCSID 与 file.encoding 值不同，则转换该文件的 CCSID。
» os400.jit.mmi.threshold	设置在使用 JIT 编译器编译方法之前方法运行的次数。 «
» os400.pool.size	定义使多少空间（以千字节计）可用于线程本地堆中的每个堆池。 «
os400.runtime.exec	• EXEC（1.3 和更高版本的缺省值）— 使用 EXEC 接口来通过 runtime.exec() 调用函数。这与其它平台最为兼容。 • QSHELL（1.2 和更低版本的缺省值）— 使用 QSHELL Interpreter 来通过 runtime.exec() 调用函数。这样就允许使用变量替代和内置函数调用。
os400.secureApplication	将使用此系统属性（os400.secureApplication）时启动的 Java 程序与注册的安全应用程序名相关联。可以使用“数字证书管理器”（DCM）来查看已注册的安全应用程序名。
os400.stderr	允许将标准错误映射至文件或套接字。参见缺省值。
os400.stdin	允许将标准输入映射至文件或套接字。参见缺省值。

系统属性	描述
os400.stdio.convert	允许控制 Java 中标准输入、标准输出和标准错误的数据转换。在缺省情况下，Java 虚拟机中执行数据转换来在 ASCII 与 EBCDIC 之间来回转换数据。可以使用此属性来打开或关闭这些转换，这将影响当前 Java 程序。参见缺省值。
os400.stdout	允许将标准输出映射至文件或套接字。参见缺省值。
user.dir	使用 getcwd API 获取当前工作目录。
user.home	通过使用“获取” API（getpwnam）来检索初始工作目录。可以将 SystemDefault.properties 文件放在 user.home 路径中，以覆盖 /QIBM/UserData/Java400/SystemDefault.properties 中的缺省属性。可以定制 iSeries 服务器以指定您自己的缺省属性值集合。
user.language	Java 虚拟机用此系统属性来读取作业 LANGID 值，并使用此值来查找相对应的语言。
user.name	Java 虚拟机使用此系统属性来从“可信计算基础”（TCB）的“安全性”部分（Security.UserName）检索有效用户概要文件名。
user.region	Java 虚拟机使用此系统属性来读取作业 CNTRYID 值，并使用此值来确定用户区域。
user.timezone	Java 虚拟机此系统属性通过使用 QlgRetrieveLocalInformation API 获取时区名。如果没有时区信息可用，则将把 user.timezone 设置为“全球标准时间”（UTC）。

os400.stdio.convert 和 os400.child.stdio.convert 系统属性值

下表显示了 os400.stdio.convert 和 os400.child.stdio.convert 系统属性的系统值。

值	描述
N（缺省值）	在读或写期间不执行标准输入输出转换。
Y	在读或写期间，执行 file.encoding 值与作业 CCSID 相互之间的所有标准输入输出转换。
1	在读期间，只执行从作业 CCSID 到 file.encoding 的标准输入数据转换。
2	在写期间，只执行从 file.encoding 到作业 CCSID 的标准输出数据转换。
3	既执行标准输入转换又执行标准输出转换。
4	在写期间，只执行从 file.encoding 到作业 CCSID 的标准错误数据转换。
5	既执行标准输入转换又执行标准错误转换。
6	既执行标准输出转换又执行标准错误转换。
7	执行所有标准输入输出转换。

os400.stdin、os400.stdout 和 os400.stderr 系统属性值

下表显示了 os400.stdin、os400.stdout 和 os400.stderr 系统属性的系统值。

值	示例名	描述	示例
File	SomeFileName	SomeFileName 是绝对路径或相对于当前目录的路径。	file:/QIBM/UserData/Java400/Output.file

值	示例名	描述	示例
Port	HostName	端口地址	port:myhost:2000
Port	TCPAddress	端口地址	port:1.1.11.111:2000

os400.verify.checks.disable 数值

os400.verify.checks.disable 系统属性值是一个表示此列表中一个或多个数值之和的字符串:

值	描述
1	**绕过本地类的访问检查**: 指示您想让 Java(TM) 虚拟机绕过对从本地文件系统装入的类的专用和受保护字段和方法的访问检查。当传送包含内部类的应用程序，而这些内部类引用其封装类的专用和受保护方法和字段时，这非常有用。
2	**在早期装入期间抑制 NoClassDefFoundError**: 指定您想让 Java 虚拟机忽略 NoClassDefFoundError，这些错误是在类型强制转换以及字段或方法访问的早期验证检查期间发生的。
4	**允许绕过 LocalVariableTable 检查**: 指示当在类的 LocalVariableTable 中遇到错误时，该类操作起来就好象 LocalVariableTable 不存在一样。否则，LocaleVariableTable 中的错误会导致 ClassFormatError。
7	在运行时使用的值。

可以以十进制、十六进制或八进制格式指示此值。将忽略小于零的值。例如，要从列表中选择前两个值，请使用此 iSeries 命令语法:

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

Java 2 Software Development Kit（J2SDK）Standard Edition 的 Java 系统属性

Java(TM) 系统属性确定 Java 程序的运行环境。它们与 OS/400 中的系统值或环境变量类似。许多属性是在 Java 虚拟机启动时设置的。

» 在 Java 2 Software Development Kit（J2SDK）Standard Edition V1.4 中，将把系统属性设置为这些系统缺省值。在使用 JAVA 或 RUNJVA CL 命令时，许多系统属性的缺省值与使用“Java 本机接口”（JNI）的“调用”API 时的缺省值不同。下表反映了 API 的使用。 «

系统属性	系统缺省值
awt.toolkit	对于 JDK 1.1.x，缺省值是 com.ibm.rawt.client.CToolkit。对于 J2SDK，缺省值是 com.ibm.rawt2.ahost.java.awt.AHToolkit。
file.encoding.pkg	sun.io
file.separator	/（斜杠）
java.class.version	» 48.0 «
java.ext.dirs	» /QIBM/ProdData/Java400/jdk14/lib/ext:/QIBM/UserData/Java400/ext «
java.home	有关详细信息，参见支持多个 Java Development Kit（JDK）。
java.library.path	OS/400 库列表
java.policy	» /QIBM/ProdData/Java400/jdk14/lib/security/java.policy «
java.specification.name	Java Language Specification

系统属性	系统缺省值
java.specification.vendor	Sun Microsystems, Inc.
java.specification.version	» 1.4 «
» sun.boot.class.path «	Class_Path_Sys
java.use.policy	true
java.vendor	IBM Corporation
java.vendor.url	http://www.ibm.com
java.vm.name	» Classic VM «
java.vm.specification.name	Java Virtual Machine Specification
java.vm.specification.vendor	Sun Microsystems, Inc.
java.vm.specification.version	» 1.0 «
java.vm.vendor	IBM Corporation
java.vm.version	» 1.4 «
line.separator	\n
os.arch	PowerPC
os.name	OS/400
os400.class.path.rawt	0
os400.class.path.security.check	20 有效值: 0 — 不执行安全性检查 10 — 等价于 RUNJVA CHKPATH(*IGNORE) 20 — 等价于 RUNJVA CHKPATH(*WARN) 30 — 等价于 RUNJVA CHKPATH(*SECURE)
os400.class.path.tools	0
os400.create.type	» interpret « 有效值: interpret — 等价于 RUNJVA OPTIMIZE(*INTERPRET) 和 INTERPRET(*OPTIMIZE), 或者 INTERPRET(*YES) direct — 其它值
os400.defineClass.optLevel	20
os400.enbpfrcol	0 有效值: 0 — 等价于 CRTJVAPGM ENBPFRCOL(*NONE) 1 — 等价于 CRTJVAPGM ENBPFRCOL(*ENTRYEXIT) 7 — 等价于 CRTJVAPGM ENBPFRCOL(*FULL) 对于非零值, JIT 生成 *JVAENTRY、*JVAEXIT、*JVAPRECALL 和 *JVAPOSTCALL 事件。
os400.interpret	0 有效值: 0 — 等价于 CRTJVAPGM INTERPRET(*NO) 1 — 等价于 CRTJVAPGM INTERPRET(*YES)

系统属性	系统缺省值
os400.optimization	» 0 « 有效值: 0 — 等价于 CRTJVAPGM OPTIMIZE(*INTERPRET) 10 — 等价于 CRTJVAPGM OPTIMIZE(10) 20 — 等价于 CRTJVAPGM OPTIMIZE(20) 30 — 等价于 CRTJVAPGM OPTIMIZE(30) 40 — 等价于 CRTJVAPGM OPTIMIZE(40)
os400.run.mode	» jitc_de « 有效值: interpret — 等价于 RUNJVA OPTIMIZE(*INTERPRET) 和 INTERPRET(*OPTIMIZE), 或者 INTERPRET(*YES) program_create_type jitc_de — 其它情况
os400.stdin.allowed	0
os400.verify.checks.disable	65535 此系统属性值是表示一个或多个数值之和的字符串。有关这些值的列表, 参见 os400.verify.checks.disable 数值。
path.separator	:（冒号）

这组系统属性是根据其它系统信息设置的。

系统属性	描述
file.encoding	将 OS/400 作业 CCSID 映射至相对应的 ISO ASCII CCSID。并且, 将 file.encoding 值设置为表示 ISO ASCII 编码字符集标识符（CCSID）的 Java 值。请查看 file.encoding 值和 iSeries CCSID 以获取一个表, 该表显示了可能的 file.encoding 值与最匹配的 iSeries 编码字符集标识符（CCSID）之间的关系。
java.class.path	用来定位类的路径。缺省为用户指定的类路径。
java.compiler	指定代码是使用 “及时” （JIT）编译器（jitc）编译, 还是既使用 JIT 编译器编译又直接处理（jitc_de）。
java.version	确定要运行的 Java Development Kit（JDK）。 如果指定了未安装的 JDK, 则会发出错误消息。如果不指定 JDK, 则最新的 JDK 变为缺省 JDK。如果只安装了一个 JDK, 则它便是缺省 JDK。有关版本详细信息, 参见支持多个 JDK。
os.version	从 “检索产品信息” 应用程序接口（API）获取 OS/400 发行版级别。
os400.CertificateContainer	指示 Java 安全套接字层（SSL）支持对启动的 Java 程序和指定的属性使用指定的证书容器。如果指定 os400.secureApplication 系统属性, 则将忽略此系统属性。例如, 输入 `-Dos400.certificateContainer=/home/username/mykeyfile.kdb` 或集成文件系统中的任何其它密钥文件。

系统属性	描述
os400.CertificateLabel	可以将此系统属性与 os400.CertificateContainer 系统属性一起指定。此属性允许您选择想要让安全套接字层（SSL）使用指定容器中的哪个证书。例如，输入 `-Dos400.certificateLabel=myCert`，其中 myCert 是您在创建或导入证书时通过“数字证书管理器”（DCM）对证书指定的标号名。
os400.child.stdio.convert	允许控制 Java 中标准输入、标准输出和标准错误的数据转换。在缺省情况下，Java 虚拟机中执行数据转换来在 ASCII 与 EBCDIC 之间来回转换数据。可以使用此属性来打开或关闭这些转换，这将影响此进程使用 runtime.exec() 方法启动的任何子进程。参见缺省值。
os400.class.path.system	» 对于 J2SDK，忽略此系统属性。 «
» os400.define.class.cache.file	此属性指定 JAR 或 ZIP 文件的名称。缺省值是空。参见高速缓存类装入程序。 «
» os400.define.class.cache.hours	此属性是一个十进制值。缺省值是 168，最大十进制值是 9999。参见高速缓存类装入程序。 «
» os400.define.class.cache.maxpgms	此属性是一个十进制值。缺省值是 5000，最大十进制值是 40000。参见高速缓存类装入程序。 «
os400.exception.trace	指定此属性将导致在 JVM 退出时将最近的异常发送至标准输出。目前忽略对此属性指定的值，但将来可能会有所更改。此属性纯粹用于调试。
» os400.file.create.auth, os400.dir.create.auth	这些属性指定对文件和目录指定的权限。指定不带任何值或带有不支持的值的属性将导致公用权限为 *NONE。 可指定 os400.file.create.auth=RWX 或 os400.dir.create.auth=RWX，其中 R = 读，W = 写，X = 执行。这些权限的任何组合都有效。 «
os400.file.io.mode	当您指定 TEXT 而不是缺省值 BINARY 时，如果文件的 CCSID 与 file.encoding 值不同，则转换该文件的 CCSID。
» os400.jit.mmi.threshold	设置在使用 JIT 编译器编译方法之前方法运行的次数。 «
» os400.pool.size	定义使多少空间（以千字节计）可用于线程本地堆中的每个堆池。 «
os400.runtime.exec	• EXEC（1.3 和更高版本的缺省值）— 使用 EXEC 接口来通过 runtime.exec() 调用函数。这与其它平台最为兼容。 • QSHELL（1.2 和更低版本的缺省值）— 使用 QSHELL Interpreter 来通过 runtime.exec() 调用函数。这样就允许使用变量替代和内置函数调用。
os400.secureApplication	将使用此系统属性（os400.secureApplication）时启动的 Java 程序与注册的安全应用程序名相关联。可以使用“数字证书管理器”（DCM）来查看已注册的安全应用程序名。

系统属性	描述
os.400.security.properties	允许完全控制所使用的 java.security 文件。当指定此属性时，J2SDK 将不使用任何其它的 java.security 文件，包括特定于 J2SDK 的 java.security 缺省值。
os400.stderr	允许将标准错误映射至文件或套接字。参见缺省值。
os400.stdin	允许将标准输入映射至文件或套接字。参见缺省值。
os400.stdin.allowed	指定是允许标准输入（1）还是不允许标准输入（0）。如果调用程序正在运行批处理作业，则不应允许标准输入。缺省值为 0。
os400.stdio.convert	允许控制 Java 中标准输入、标准输出和标准错误的数据转换。在缺省情况下，Java 虚拟机中执行数据转换来在 ASCII 与 EBCDIC 之间来回转换数据。可以使用此属性来打开或关闭这些转换，这将影响当前 Java 程序。参见缺省值。
os400.stdout	允许将标准输出映射至文件或套接字。参见缺省值。
user.dir	使用 getcwd API 获取当前工作目录。
user.home	通过使用“获取” API（getpwnam）来检索初始工作目录。可以将 SystemDefault.properties 文件放在 user.home 路径中，以覆盖 /QIBM/UserData/Java400/SystemDefault.properties 中的缺省属性。可以定制 iSeries 服务器以指定您自己的缺省属性值集合。
user.language	Java 虚拟机用此系统属性来读取作业 LANGID 值，并使用此值来查找相对应的语言。
user.name	Java 虚拟机使用此系统属性来从“可信计算基础”（TCB）的“安全性”部分（Security.UserName）检索有效用户概要文件名。
user.region	Java 虚拟机使用此系统属性来读取作业 CNTRYID 值，并使用此值来确定用户区域。
user.timezone	Java 虚拟机此系统属性通过使用 QlgRetrieveLocalInformation API 获取时区名。如果没有时区信息可用，则将把 user.timezone 设置为“全球标准时间”（UTC）。

创建国际化的 Java 程序

如果需要针对世界上的特定地区定制 Java(TM) 程序，则可以借助 Java 语言环境创建国际化的 Java 程序。

要创建国际化的 Java 程序，请执行下列步骤:

1. 隔离语言环境敏感代码和数据。例如，程序中的字符串、日期和数字。
2. 使用 Locale 类来设置或获取语言环境。
3. 如果不想使用缺省语言环境，则对日期和数字进行格式化以指定语言环境。
4. 创建用来处理字符串和其它语言环境敏感数据的资源束。

要在您自己的 Java 程序中执行这些任务，请参考这些示例:

- 使用 java.util.DateFormat 类来使日期国际化

- 使用 java.util.NumberFormat 类来使数字显示国际化
- 使用 java.util.ResourceBundle 类来使特定于语言环境的数据国际化

有关国际化的更多信息，请单击下列链接中的任何之一:

- OS/400 全球化
- Sun Microsystems 的国际化

iSeries 服务器上的时区环境变量

对于 Java(TM) 2 SDK（J2SDK）Standard Edition V1.4，可使用本机方法 getSystemTimeZoneID() 来设置 “Java 虚拟机” 的时区。iSeries 服务器使用作为 *ENV 对象一部分的 *LOCALE 对象。将 *LOCALE 对象中的 tname 字段设置为适当的系统值。然后，从 getSystemTimeZoneID() 将该值作为相关联的 Java 字符串对象返回。

配置时区: JVM 要求设置 QUTCOFFSET 系统值以及当前作业的 LOCALE 中的一天时间信息以正确地确定本地时间。QUTCOFFSET 是一个系统值，它指定当前本地时间与全球标准时间（UTC）相差的小时数。对于中央标准时间（CST），这将是 -6:00。对于中央夏令时时间（CDT），正确的值是 -5:00。QUTCOFFSET 值使 JVM 能够确定 UTC 的正确值。

作业的 LOCALE 信息是这样设置的: 创建包含一天时间信息的 *LOCALE 对象，并使用 QLOCALE 系统值或作业的用户概要文件上的 LOCALE 关键字来对作业指定该 *LOCALE。可以在 OS/400全球化出版物中找到有关创建和使用 LOCALE 的详细信息。

正确地设置 *LOCALE 信息将允许 JVM 使 user.timezone 属性在缺省情况下具有正确的时区。可在命令行上手工设置 user.timezone 属性以覆盖 *LOCALE 对象提供的缺省设置。

以下是 LC_TOD 信息的一个示例，要为 Java 配置正确的时区，必须将该信息包括在 *LOCALE 对象中:

```
LC_TOD

% TZDIFF is number of minutes difference from GMT
tzdiff   -300
% Timezone name (this is the value that you would have passed to
% the JVM as the user.timezone property.) See abbreviations later
% in this document.
tname    "<C><S><T>"
% Name used for daylight savings time.
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
```

下表指示系统值和相关联的 Java 字符串对象。

注意: 系统值 “Hong Kong” 表示中国香港特别行政区。

系统值	Java 字符串对象
Africa/Abidjan	Africa/Abidjan
Africa/Accra	Africa/Accra
Africa/Addis_Ababa	Africa/Addis_Ababa
Africa/Algiers	Africa/Algiers
Africa/Asmera	Africa/Asmera

系统值	Java 字符串对象
Africa/Bamako	GMT
Africa/Bangui	Africa/Bangui
Africa/Banjul	Africa/Banjul
Africa/Bissau	Africa/Bissau
Africa/Blantyre	Africa/Blantyre
Africa/Brazzaville	Africa/Luanda
Africa/Bujumbura	Africa/Bujumbura
Africa/Cairo	Africa/Cairo
Africa/Casablanca	Africa/Casablanca
Africa/Ceuta	Europe/Paris
Africa/Conakry	Africa/Conakry
Africa/Dakar	Africa/Dakar
Africa/Dar_es_Salaam	Africa/Dar_es_Salaam
Africa/Djibouti	Africa/Djibouti
Africa/Douala	Africa/Douala
Africa/El_Aaiun	Africa/Casablanca
Africa/Freetown	Africa/Freetown
Africa/Gaborone	Africa/Gaborone
Africa/Harare	Africa/Harare
Africa/Johannesburg	Africa/Johannesburg
Africa/Kampala	Africa/Kampala
Africa/Khartoum	Africa/Khartoum
Africa/Kigali	Africa/Kigali
Africa/Kinshasa	Africa/Kinshasa
Africa/Lagos	Africa/Lagos
Africa/Libreville	Africa/Libreville
Africa/Lome	Africa/Lome
Africa/Luanda	Africa/Luanda
Africa/Lubumbashi	Africa/Lubumbashi
Africa/Lusaka	Africa/Lusaka
Africa/Malabo	Africa/Malabo
Africa/Maputo	Africa/Maputo
Africa/Maseru	Africa/Maseru
Africa/Mbabane	Africa/Mbabane
Africa/Mogadishu	Africa/Mogadishu
Africa/Monrovia	Africa/Monrovia
Africa/Nairobi	Africa/Nairobi
Africa/Ndjamena	Africa/Ndjamena
Africa/Niamey	Africa/Niamey
Africa/Nouakchott	Africa/Nouakchott
Africa/Ouagadougou	Africa/Ouagadougou

系统值	Java 字符串对象
Africa/Porto-Novo	Africa/Porto-Novo
Africa/Sao_Tome	Africa/Sao_Tome
Africa/Timbuktu	Africa/Timbuktu
Africa/Tripoli	Africa/Tripoli
Africa/Tunis	Africa/Tunis
Africa/Windhoek	Africa/Windhoek
America/Adak	America/Adak
America/Anchorage	America/Anchorage
America/Anguilla	America/Anguilla
America/Antigua	America/Antigua
America/Araguaina	America/Sao_Paulo
America/Aruba	America/Aruba
America/Asuncion	America/Asuncion
America/Atka	America/Adak
America/Barbados	America/Barbados
America/Belize	America/Belize
America/Bogota	America/Bogota
America/Boise	America/Denver
America/Buenos_Aires	America/Buenos_Aires
America/Cancun	America/Chicago
America/Caracas	America/Caracas
America/Cayenne	America/Cayenne
America/Cayman	America/Cayman
America/Chicago	America/Chicago
America/Chihuahua	America/Denver
America/Costa_Rica	America/Costa_Rica
America/Cuiaba	America/Cuiaba
America/Curacao	America/Curacao
America/Dawson	America/Los_Angeles
America/Dawson_Creek	America/Dawson_Creek
America/Denver	America/Denver
America/Detroit	America/New_York
America/Dominica	America/Dominica
America/Edmonton	America/Edmonton
America/El_Salvador	America/El_Salvador
America/Ensenada	America/Los_Angeles
America/Fort_Wayne	America/Indianapolis
America/Fortaleza	America/Fortaleza
America/Glace_Bay	America/Halifax
America/Godthab	America/Godthab
America/Goose_Bay	America/Thule

系统值	Java 字符串对象
America/Grand_Turk	America/Grand_Turk
America/Grenada	America/Grenada
America/Guadeloupe	America/Guadeloupe
America/Guatemala	America/Guatemala
America/Guayaquil	America/Guayaquil
America/Guyana	America/Guyana
America/Halifax	America/Halifax
America/Havana	America/Havana
America/Indiana/Indianapolis	America/Indianapolis
America/Indianapolis	America/Indianapolis
America/Inuvik	America/Denver
America/Iqaluit	America/New_York
America/Jamaica	America/Jamaica
America/Juneau	America/Anchorage
America/La_Paz	America/La_Paz
America/Lima	America/Lima
America/Los_Angeles	America/Los_Angeles
America/Louisville	America/New_York
America/Managua	America/Managua
America/Manaus	America/Manaus
America/Martinique	America/Martinique
America/Mazatlan	America/Mazatlan
America/Menominee	America/Winnipeg
America/Mexico_City	America/Mexico_City
America/Miquelon	America/Miquelon
America/Montevideo	America/Montevideo
America/Montreal	America/Montreal
America/Montserrat	America/Montserrat
America/Nassau	America/Nassau
America/New_York	America/New_York
America/Nipigon	America/New_York
America/Nome	America/Anchorage
America/Noronha	America/Noronha
America/Panama	America/Panama
America/Pangnirtung	America/Thule
America/Paramaribo	America/Paramaribo
America/Phoenix	America/Phoenix
America/Port-au-Prince	America/Port-au-Prince
America/Port_of_Spain	America/Port_of_Spain
America/Porto_Acre	America/Porto_Acre
America/Puerto_Rico	America/Puerto_Rico

系统值	Java 字符串对象
America/Rainy_River	America/Chicago
America/Rankin_Inlet	America/Chicago
America/Regina	America/Regina
America/Santiago	America/Santiago
America/Santo_Domingo	America/Santo_Domingo
America/Sao_Paulo	America/Sao_Paulo
America/Scoresbysund	America/Scoresbysund
America/Shiprock	America/Denver
America/St_Johns	America/St_Johns
America/St_Kitts	America/St_Kitts
America/St_Lucia	America/St_Lucia
America/St_Thomas	America/St_Thomas
America/St_Vincent	America/St_Vincent
America/Tegucigalpa	America/Tegucigalpa
America/Thule	America/Thule
America/Thunder_Bay	America/New_York
America/Tijuana	America/Tijuana
America/Tortola	America/Tortola
America/Vancouver	America/Vancouver
America/Virgin	America/St_Thomas
America/Whitehorse	America/Los_Angeles
America/Winnipeg	America/Winnipeg
America/Yakutat	America/Anchorage
America/Yellowknife	America/Denver
Antarctica/Casey	Antarctica/Casey
Antarctica/DumontDUrville	Antarctica/DumontDUrville
Antarctica/Mawson	Antarctica/Mawson
Antarctica/McMurdo	Antarctica/McMurdo
Antarctica/Palmer	Antarctica/Palmer
Antarctica/South_Pole	Antarctica/McMurdo
Arctic/Longyearbyen	Europe/Oslo
Asia/Aden	Asia/Aden
Asia/Almaty	Asia/Almaty
Asia/Amman	Asia/Amman
Asia/Anadyr	Asia/Anadyr
Asia/Aqtau	Asia/Aqtau
Asia/Aqtobe	Asia/Aqtobe
Asia/Ashkhabad	Asia/Ashkhabad
Asia/Baghdad	Asia/Baghdad
Asia/Bahrain	Asia/Bahrain
Asia/Baku	Asia/Baku

系统值	Java 字符串对象
Asia/Bangkok	Asia/Bangkok
Asia/Beirut	Asia/Beirut
Asia/Bishkek	Asia/Bishkek
Asia/Brunei	Asia/Brunei
Asia/Calcutta	Asia/Calcutta
Asia/Chungking	Asia/Shanghai
Asia/Colombo	Asia/Colombo
Asia/Dacca	Asia/Dacca
Asia/Damascus	Asia/Damascus
Asia/Dubai	Asia/Dubai
Asia/Dushanbe	Asia/Dushanbe
Asia/Gaza	Asia/Amman
Asia/Harbin	Asia/Shanghai
Asia/Hong_Kong	Asia/Hong_Kong
Asia/Irkutsk	Asia/Irkutsk
Asia/Istanbul	Europe/Istanbul
Asia/Jakarta	Asia/Jakarta
Asia/Jayapura	Asia/Jayapura
Asia/Jerusalem	Asia/Jerusalem
Asia/Kabul	Asia/Kabul
Asia/Kamchatka	Asia/Kamchatka
Asia/Karachi	Asia/Karachi
Asia/Kashgar	Asia/Shanghai
Asia/Katmandu	Asia/Katmandu
Asia/Krasnoyarsk	Asia/Krasnoyarsk
Asia/Kuala_Lumpur	Asia/Kuala_Lumpur
Asia/Kuwait	Asia/Kuwait
Asia/Macao	Asia/Macao
Asia/Magadan	Asia/Magadan
Asia/Manila	Asia/Manila
Asia/Muscat	Asia/Muscat
Asia/Nicosia	Asia/Nicosia
Asia/Novosibirsk	Asia/Novosibirsk
Asia/Omsk	Asia/Novosibirsk
Asia/Phnom_Penh	Asia/Phnom_Penh
Asia/Pyongyang	Asia/Pyongyang
Asia/Qatar	Asia/Qatar
Asia/Rangoon	Asia/Rangoon
Asia/Riyadh	Asia/Riyadh
Asia/Saigon	Asia/Saigon
Asia/Seoul	Asia/Seoul

系统值	Java 字符串对象
Asia/Shanghai	Asia/Shanghai
Asia/Singapore	Asia/Singapore
Asia/Taipei	Asia/Taipei
Asia/Tashkent	Asia/Tashkent
Asia/Tbilisi	Asia/Tbilisi
Asia/Tehran	Asia/Tehran
Asia/Tel_Aviv	Asia/Jerusalem
Asia/Thimbu	Asia/Thimbu
Asia/Tokyo	Asia/Tokyo
Asia/Ujung_Pandang	Asia/Ujung_Pandang
Asia/Ulan_Bator	Asia/Ulan_Bator
Asia/Urumqi	Asia/Shanghai
Asia/Vientiane	Asia/Vientiane
Asia/Vladivostok	Asia/Vladivostok
Asia/Yakutsk	Asia/Yakutsk
Asia/Yekaterinburg	Asia/Yekaterinburg
Asia/Yerevan	Asia/Yerevan
Atlantic/Azores	Atlantic/Azores
Atlantic/Bermuda	Atlantic/Bermuda
Atlantic/Canary	Atlantic/Canary
Atlantic/Cape_Verde	Atlantic/Cape_Verde
Atlantic/Faeroe	Atlantic/Faeroe
Atlantic/Jan_Mayen	Atlantic/Jan_Mayen
Atlantic/Madeira	Europe/London
Atlantic/Reykjavik	Atlantic/Reykjavik
Atlantic/South_Georgia	Atlantic/South_Georgia
Atlantic/St_Helena	Atlantic/St_Helena
Atlantic/Stanley	Atlantic/Stanley
Australia/ACT	Australia/Sydney
Australia/Adelaide	Australia/Adelaide
Australia/Brisbane	Australia/Brisbane
Australia/Broken_Hill	Australia/Broken_Hill
Australia/Canberra	Australia/Sydney
Australia/Darwin	Australia/Darwin
Australia/Hobart	Australia/Hobart
Australia/LHI	Australia/Lord_Howe
Australia/Lord_Howe	Australia/Lord_Howe
Australia/Melbourne	Australia/Sydney
Australia/NSW	Australia/Sydney
Australia/North	Australia/Darwin
Australia/Perth	Australia/Perth

系统值	Java 字符串对象
Australia/Queensland	Australia/Brisbane
Australia/South	Australia/Adelaide
Australia/Sydney	Australia/Sydney
Australia/Tasmania	Australia/Hobart
Australia/Victoria	Australia/Sydney
Australia/West	Australia/Perth
Australia/Yancowinna	Australia/Broken_Hill
Brazil/Acre	America/Porto_Acre
Brazil/DeNoronha	America/Noronha
Brazil/East	America/Sao_Paulo
Brazil/West	America/Manaus
CET	Europe/Paris
CST	America/Chicago
CST6CDT	America/Chicago
Canada/Atlantic	America/Halifax
Canada/Central	America/Winnipeg
Canada/East-Saskatchewan	America/Regina
Canada/Eastern	America/Montreal
Canada/Mountain	America/Edmonton
Canada/Newfoundland	America/St_Johns
Canada/Pacific	America/Vancouver
Canada/Saskatchewan	America/Regina
Canada/Yukon	America/Los_Angeles
Chile/Continental	America/Santiago
Chile/EasterIsland	Pacific/Easter
Cuba	America/Havana
EET	America/Indianapolis
EST5EDT	America/New_York
Egypt	Africa/Cairo
Eire	Europe/Dublin
Etc/GMT	GMT
Etc/GMT0	GMT
Etc/Greenwich	GMT
Etc/UCT	UTC
Etc/UTC	UTC
Etc/Universal	UTC
Etc/Zulu	UTC
Europe/Amsterdam	Europe/Amsterdam
Europe/Andorra	Europe/Andorra
Europe/Athens	Europe/Athens
Europe/Belfast	Europe/London

系统值	Java 字符串对象
Europe/Belgrade	Europe/Belgrade
Europe/Berlin	Europe/Berlin
Europe/Bratislava	Europe/Prague
Europe/Brussels	Europe/Brussels
Europe/Bucharest	Europe/Bucharest
Europe/Budapest	Europe/Budapest
Europe/Chisinau	Europe/Chisinau
Europe/Copenhagen	Europe/Copenhagen
Europe/Dublin	Europe/Dublin
Europe/Gibraltar	Europe/Gibraltar
Europe/Helsinki	Europe/Helsinki
Europe/Istanbul	Europe/Istanbul
Europe/Kaliningrad	Europe/Kaliningrad
Europe/Kiev	Europe/Kiev
Europe/Lisbon	Europe/Lisbon
Europe/Ljubljana	Europe/Belgrade
Europe/London	Europe/London
Europe/Luxembourg	Europe/Luxembourg
Europe/Madrid	Europe/Madrid
Europe/Malta	Europe/Malta
Europe/Minsk	Europe/Minsk
Europe/Monaco	Europe/Monaco
Europe/Moscow	Europe/Moscow
Europe/Oslo	Europe/Oslo
Europe/Paris	Europe/Paris
Europe/Prague	Europe/Prague
Europe/Riga	Europe/Riga
Europe/Rome	Europe/Rome
Europe/Samara	Europe/Samara
Europe/San_Marino	Europe/Rome
Europe/Sarajevo	Europe/Belgrade
Europe/Simferopol	Europe/Simferopol
Europe/Skopje	Europe/Belgrade
Europe/Sofia	Europe/Sofia
Europe/Stockholm	Europe/Stockholm
Europe/Tallinn	Europe/Tallinn
Europe/Tirane	Europe/Tirane
Europe/Vaduz	Europe/Vaduz
Europe/Vatican	Europe/Rome
Europe/Vienna	Europe/Vienna
Europe/Vilnius	Europe/Vilnius

系统值	Java 字符串对象
Europe/Warsaw	Europe/Warsaw
Europe/Zagreb	Europe/Belgrade
Europe/Zurich	Europe/Zurich
Factory	GMT
GB	Europe/London
GB-Eire	Europe/London
GMT	GMT
GMT0	GMT
Greenwich	GMT
HST	Pacific/Honolulu
Hongkong	Asia/Hong_Kong
Iceland	Atlantic/Reykjavik
Indian/Antananarivo	Indian/Antananarivo
Indian/Chagos	Indian/Chagos
Indian/Christmas	Indian/Christmas
Indian/Cocos	Indian/Cocos
Indian/Comoro	Indian/Comoro
Indian/Kerguelen	Indian/Kerguelen
Indian/Mahe	Indian/Mahe
Indian/Maldives	Indian/Maldives
Indian/Mauritius	Indian/Mauritius
Indian/Mayotte	Indian/Mayotte
Indian/Reunion	Indian/Reunion
Iran	Asia/Tehran
Israel	Asia/Jerusalem
Jamaica	America/Jamaica
Japan	Asia/Tokyo
Libya	Africa/Tripoli
MET	Europe/Paris
MST	America/Phoenix
MST7MDT	America/Denver
Mexico/BajaNorte	America/Tijuana
Mexico/BajaSur	America/Mazatlan
Mexico/General	America/Mexico_City
NZ	Pacific/Auckland
NZ-CHAT	Pacific/Chatham
Navajo	America/Denver
PRC	Asia/Shanghai
PST	America/Los_Angeles
PST8PDT	America/Los_Angeles
Pacific/Apia	Pacific/Apia

系统值	Java 字符串对象
Pacific/Auckland	Pacific/Auckland
Pacific/Chatham	Pacific/Chatham
Pacific/Easter	Pacific/Easter
Pacific/Efate	Pacific/Efate
Pacific/Enderbury	Pacific/Enderbury
Pacific/Fakaofo	Pacific/Fakaofo
Pacific/Fiji	Pacific/Fiji
Pacific/Funafuti	Pacific/Funafuti
Pacific/Galapagos	Pacific/Galapagos
Pacific/Gambier	Pacific/Gambier
Pacific/Guadalcanal	Pacific/Guadalcanal
Pacific/Guam	Pacific/Guam
Pacific/Honolulu	Pacific/Honolulu
Pacific/Kiritimati	Pacific/Kiritimati
Pacific/Kosrae	Pacific/Kosrae
Pacific/Majuro	Pacific/Majuro
Pacific/Marquesas	Pacific/Marquesas
Pacific/Nauru	Pacific/Nauru
Pacific/Niue	Pacific/Niue
Pacific/Norfolk	Pacific/Norfolk
Pacific/Noumea	Pacific/Noumea
Pacific/Pago_Pago	Pacific/Pago_Pago
Pacific/Palau	Pacific/Palau
Pacific/Pitcairn	Pacific/Pitcairn
Pacific/Ponape	Pacific/Ponape
Pacific/Port_Moresby	Pacific/Port_Moresby
Pacific/Rarotonga	Pacific/Rarotonga
Pacific/Saipan	Pacific/Saipan
Pacific/Samoa	Pacific/Pago_Pago
Pacific/Tahiti	Pacific/Tahiti
Pacific/Tarawa	Pacific/Tarawa
Pacific/Tongatapu	Pacific/Tongatapu
Pacific/Truk	Pacific/Truk
Pacific/Wake	Pacific/Wake
Pacific/Wallis	Pacific/Wallis
Poland	Europe/Warsaw
Portugal	Europe/Lisbon
Taiwan	Asia/Taipei
ROK	Asia/Seoul
Singapore	Asia/Singapore
SystemV/AST4ADT	America/Thule

系统值	Java 字符串对象
SystemV/CST6CDT	America/Chicago
SystemV/EST5EDT	America/New_York
SystemV/MST7MDT	America/Denver
SystemV/PST8PDT	America/Los_Angeles
SystemV/YST9YDT	America/Anchorage
Turkey	Europe/Istanbul
UCT	UTC
US/Alaska	America/Anchorage
US/Aleutian	America/Adak
US/Arizona	America/Phoenix
US/Central	America/Chicago
US/East-Indiana	America/Indianapolis
US/Eastern	America/New_York
US/Hawaii	Pacific/Honolulu
US/Michigan	America/New_York
US/Mountain	America/Denver
US/Pacific	America/Los_Angeles
US/Pacific-New	America/Los_Angeles
US/Samoa	Pacific/Pago_Pago
UTC	UTC
Universal	UTC
W-SU	Europe/Moscow
WET	Europe/London
Zulu	UTC

Java 语言环境

语言环境指的是世界上语言和风俗相同的地理或政治区域。在 Java(TM) 中，Locale 类表示语言环境。

支持的 Java 语言环境

IBM Developer Kit for Java 支持下面这些语言环境。iSeries 作业和 CNTRYID 确定了缺省语言环境。有关更多详细信息，参见 Java 系统属性。

JDK 1.1.6 中的语言环境名	ISO 语言环境名	语言 / 国家或地区
ar	ar_EG	阿拉伯语 / 埃及
be	be_BY	白俄罗斯语 / 白俄罗斯
bg	bg_BG	保加利亚语 / 保加利亚
ca	ca_ES	加泰隆语 / 西班牙
cs	cs_CZ	捷克语 / 捷克共和国
da	da_DK	丹麦语 / 丹麦
de	de_DE	德语 / 德国
de_AT	de_AT	德语 / 奥地利

JDK 1.1.6 中的语言环境名	ISO 语言环境名	语言 / 国家或地区
de_CH	de_CH	德语 / 瑞士
el	el_GR	希腊语 / 希腊
en	en_US	英语 / 美国
en_AU	en_AU	英语 / 澳大利亚
en_CA	en_CA	英语 / 加拿大
en_GB	en_GB	英语 / 英国
en_IE	en_IE	英语 / 爱尔兰
en_NZ	en_NZ	英语 / 新西兰
en_ZA	en_ZA	英语 / 南非
es	es_ES	西班牙语 / 西班牙
es_AR	es_AR	西班牙语 / 阿根廷
es_BO	es_BO	西班牙语 / 玻利维亚
es_CL	es_CL	西班牙语 / 智利
es_CR	es_CR	西班牙语 / 哥斯达黎加
es_DO	es_DO	西班牙语 / 多米尼加共和国
es_EC	es_EC	西班牙语 / 厄瓜多尔
es_GT	es_GT	西班牙语 / 危地马拉
es_HN	es_HN	西班牙语 / 洪都拉斯
es_MX	es_MX	西班牙语 / 墨西哥
es_NI	es_NI	西班牙语 / 尼加拉瓜
es_PA	es_PA	西班牙语 / 巴拿马
es_PE	es_PE	西班牙语 / 秘鲁
es_PR	es_PR	西班牙语 / 波多黎各
es_PY	es_PY	西班牙语 / 巴拉圭
es_SV	es_SV	西班牙语 / 萨尔瓦多
es_UY	es_UY	西班牙语 / 乌拉圭
es_VE	es_VE	西班牙语 / 委内瑞拉
et	et_EE	爱沙尼亚语 / 爱沙尼亚
fi	fi_FI	芬兰语 / 芬兰
fr	fr_FR	法语 / 法国
fr_BE	fr_BE	法语 / 比利时
fr_CA	fr_CA	法语 / 加拿大
fr_CH	fr_CH	法语 / 瑞士
hr	hr_HR	克罗地亚语 / 克罗地亚
hu	hu_HU	匈牙利语 / 匈牙利
is	is_IS	冰岛语 / 冰岛
it	it_IT	意大利语 / 意大利
it_CH	it_CH	意大利语 / 瑞士
iw	iw_IL	希伯莱语 / 以色列
ja	ja_JP	日语 / 日本
ko	ko_KR	韩国语 / 韩国

JDK 1.1.6 中的语言环境名	ISO 语言环境名	语言 / 国家或地区
lt	lt_LT	立陶宛语 / 立陶宛
lv	lv_LV	拉脱维亚语 / 拉脱维亚
mk	mk_MK	马其顿语 / 马其顿
nl	nl_NL	荷兰语 / 荷兰
nl_BE	nl_BE	荷兰语 / 比利时
no	no_NO_B	挪威语 / 挪威
no_NO_NY	no_NO_NY	挪威语 / 挪威
pl	pl_PL	波兰语 / 波兰
pt	pt_PT	葡萄牙语 / 葡萄牙
ro	ro_RO	罗马尼亚语 / 罗马尼亚
ru	ru_RU	俄语 / 俄罗斯
sh	sh_SP	塞尔维亚 — 克罗地亚语 / 塞尔维亚
sk	sk_SK	斯洛伐克语 / 斯洛伐克
sl	sl_SI	斯洛文尼亚语 / 斯洛文尼亚
sq	sq_AL	阿尔巴尼亚语 / 阿尔巴尼亚
sr	sr_SP	塞尔维亚语 / 塞尔维亚
sv	sv_SE	瑞典语 / 瑞典
tr	tr_TR	土耳其语 / 土耳其
uk	uk_UA	乌克兰语 / 乌克兰
zh	zh_CN	简体中文
zh_TW	zh_TW	繁体中文

示例: 使用 java.util.DateFormat 类使日期国际化: 此示例显示如何使用语言环境来格式化日期。

示例 1: 演示使用 java.util.DateFormat 类来使日期国际化

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//************************
// 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));
  }
}
```

有关更多信息，参见创建国际化的 Java(TM) 程序。

示例: 使用 java.util.NumberFormat 类来使数字显示国际化: 此示例显示如何使用语言环境来格式化数字。

示例 1: 演示使用 java.util.NumberFormat 类来使数字输出国际化

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//*************************
// 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));
     }
 }
```

有关更多信息，参见创建国际化的 Java(TM) 程序。

示例: 使用 java.util.ResourceBundle 类来使特定于语言环境的数据国际化: 此示例显示如何将语言环境与资源束配合使用来使程序字符串国际化。

要使 ResourceBundleExample 程序按预期的那样运行，下面这些属性文件是必需的:

RBExample.properties 的内容

```
Hello.text=Hello
```

RBExample_de.properties 的内容

```
Hello.text=Guten  Tag
```

RBExample_fr_FR.properties 的内容

```
Hello.text=Bonjour
```

示例 1: 演示使用 java.util.ResourceBundle 类来使特定于语言环境的数据国际化

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//*********************************
// 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"));
   }
}
```

有关更多信息，参见创建国际化的 Java(TM) 程序。

Java 字符编码

在内部，Java(TM) 虚拟机（JVM）总是操作 Unicode 数据。然而，所有传送到 JVM 中以及从 JVM 中传送出去的数据都具有与 file.encoding 属性相匹配的格式。读入 JVM 的数据将从 file.encoding 转换为 Unicode，而从 JVM 中发送出去的数据将从 Unicode 转换为 file.encoding。

Java 程序的数据文件存储在集成文件系统中。集成文件系统中的文件是使用编码字符集标识符（CCSID）作了标记的，CCSID 标识了文件中包含的数据的字符编码。有关在 iSeries 服务器上 file.encoding 与 CCSID 如何相关的描述，参见 File.encoding 值和 iSeries CCSID 表。

当 Java 程序读取数据时，期望该数据具有与 file.encoding 相匹配的字符编码。当 Java 程序将数据写至文件时，采用与 file.encoding 相匹配的字符编码来写该数据。这也适用于 javac 命令处理的 Java 源代码文件（.java 文件），并适用于使用 .net 包通过“传输控制协议／网际协议”（TCP/IP）套接字发送和接收的数据。

与对 System.in、System.out 和 System.err 读写的数据的处理方式相比，在将其它源指定给标准输入、标准输出和标准错误时对那些源读写的数据的处理方式并不相同。由于标准输入、标准输出和标准错误通常与 iSeries 服务器上的 EBCDIC 设备相连接，因此 JVM 对数据执行转换来从 file.encoding 的正常字符编码转换为与 iSeries 作业 CCSID 相匹配的 CCSID。当 System.in、System.out 或 System.err 重定向至文件或套接字而非定向至标准输入、标准输出和标准错误时，不执行此项附加数据转换，数据保持具有与 file.encoding 相匹配的字符编码。

当必须采用与 file.encoding 不同的字符编码来将数据读入 Java 程序或从 Java 程序写出时，程序可使用 Java IO 类 java.io.InputStreamReader、java.io.FileReader、java.io.OutputStreamReader 和 java.io.FileWriter。这些 Java 类允许指定 file.encoding 值，该值优先于 JVM 当前所使用的缺省 file.encoding 属性。

通过 JDBC API 流入或流出 DB2/400 数据库的数据在 iSeries 数据库的 CCSID 之间进行双向转换。

通过“Java 本机接口”传送至其它程序或从其它程序传送来的数据不进行转换。

有关国际化的更多信息，参见 OS/400 全球化。

您还可以查看 Sun Microsystems 的国际化以了解更多信息。

File.encoding 值和 iSeries CCSID: 此表显示了可能的 file.encoding 值与最匹配的 iSeries 编码字符集标识符（CCSID）之间的关系。

file.encoding	CCSID	描述
Big5	950	8 位 ASCII 繁体中文 BIG-5
CNS11643	964	汉语字符集（繁体中文）
Cp037	037	IBM EBCDIC 美国、加拿大、荷兰...
Cp273	273	IBM EBCDIC 德国、奥地利
Cp277	277	IBM EBCDIC 丹麦、挪威
Cp278	278	IBM EBCDIC 芬兰、瑞典
Cp280	280	IBM EBCDIC 意大利
Cp284	284	IBM EBCDIC 西班牙、拉丁美洲
Cp285	285	IBM EBCDIC 英国
Cp297	297	IBM EBCDIC 法国
Cp420	420	IBM EBCDIC 阿拉伯语
Cp424	424	IBM EBCDIC 希伯莱
Cp437	437	8 位 ASCII 美国 PC
Cp500	500	IBM EBCDIC 国际
Cp737	737	8 位 ASCII 希腊 MS-DOS
Cp775	775	8 位 ASCII 波罗的海 MS-DOS
Cp838	838	IBM EBCDIC 泰国
Cp850	850	8 位 ASCII 多国拉丁语 1
Cp852	852	8 位 ASCII 拉丁语 2
Cp855	855	8 位 ASCII 西里尔
Cp856	856	8 位 ASCII 希伯莱
Cp857	857	8 位 ASCII 拉丁语 5
Cp860	860	8 位 ASCII 葡萄牙
Cp861	861	8 位 ASCII 冰岛
Cp862	862	8 位 ASCII 希伯莱
Cp863	863	8 位 ASCII 加拿大
Cp864	864	8 位 ASCII 阿拉伯语
Cp865	865	8 位 ASCII 丹麦、挪威
Cp866	866	8 位 ASCII 西里尔

file.encoding	CCSID	描述
Cp868	868	8 位 ASCII 乌尔都语
Cp869	869	8 位 ASCII 希腊
Cp870	870	IBM EBCDIC 拉丁语 2
Cp871	871	IBM EBCDIC 冰岛
Cp874	874	8 位 ASCII 泰国
Cp875	875	IBM EBCDIC 希腊
Cp918	918	IBM EBCDIC 乌尔都语
Cp921	921	8 位 ASCII 波罗的海
Cp922	922	8 位 ASCII 爱沙尼亚
Cp930	930	IBM EBCDIC 日语扩展片假名
Cp933	933	IBM EBCDIC 韩国语
Cp935	935	IBM EBCDIC 简体中文
Cp937	937	IBM EBCDIC 繁体中文
Cp939	939	IBM EBCDIC 日语扩展拉丁语
Cp942	942	8 位 ASCII 日语
Cp943	943	开放环境的混合日语 PC 数据
Cp943C	943	开放环境的混合日语 PC 数据
Cp948	948	8 位 ASCII IBM 繁体中文
Cp949	949	8 位 ASCII 韩国语 KSC5601
Cp950	950	8 位 ASCII 繁体中文 BIG-5
Cp964	964	EUC 繁体中文
Cp970	970	EUC 韩国语
Cp1006	1006	ISO 8 位乌尔都语
Cp1025	1025	IBM EBCDIC 西里尔
Cp1026	1026	IBM EBCDIC 土耳其
Cp1046	1046	8 位 ASCII 阿拉伯语
Cp1097	1097	IBM EBCDIC 法尔西语
Cp1098	1098	8 位 ASCII 法尔西语
Cp1112	1112	IBM EBCDIC 波罗的海
Cp1122	1122	IBM EBCDIC 爱沙尼亚
Cp1123	1123	IBM EBCDIC 乌克兰
Cp1124	1124	ISO 8 位乌克兰
Cp1250	1250	MS-Win 拉丁语 2
Cp1251	1251	MS-Win 西里尔
Cp1252	1252	MS-Win 拉丁语 1
Cp1253	1253	MS-Win 希腊语
Cp1254	1254	MS-Win 土耳其语
Cp1255	1255	MS-Win 希伯莱语
Cp1256	1256	MS-Win 阿拉伯语
Cp1257	1257	MS-Win 波罗的海
Cp1258	1251	MS-Win 俄罗斯语

file.encoding	CCSID	描述
Cp1381	1381	8 位 ASCII 简体中文 GB
Cp1383	1383	EUC 简体中文
Cp33722	33722	EUC 日语
EUC_CN	1383	EUC 简体中文
EUC_JP	33722	EUC 日语
EUC_KR	970	EUC 韩国语
EUC_TW	964	EUC 繁体中文
GB2312	1381	8 位 ASCII 简体中文 GB
GBK	1386	新简体中文 8 位 ASCII 9
ISO2022CN_CNS	无	7 位 ASCII 繁体中文
ISO2022CN_GB	无	7 位 ASCII 简体中文
ISO2022JP	5054	7 位 ASCII 日语
ISO2022KR	25546	7 位 ASCII 韩国语
ISO8859_1	819	ISO 8859-1 ISO 拉丁语 1
ISO8859_2	912	ISO 8859-2 ISO 拉丁语 2
ISO8859_3	913	ISO 8859-3 ISO 拉丁语 3
ISO8859_4	914	ISO 8859-4 ISO 拉丁语 4
ISO8859_5	915	ISO 8859-5 ISO 拉丁语 5
ISO8859_6	1089	ISO 8859-6 ISO 拉丁语 6（阿拉伯语）
ISO8859_7	813	ISO 8859-7 ISO 拉丁语 7（希腊语／拉丁语）
ISO8859_8	916	ISO 8859-8 ISO 拉丁语 8（希伯莱）
ISO8859_9	920	ISO 8859-9 ISO 拉丁语 9（ECMA-128，土耳其）
JIS0201	897	日本工业标准 X0201
JIS0208	952	日本工业标准 X0208
JIS0212	953	日本工业标准 X0212
Johab	无	韩国组合韩国语本名编码（全）
K018_R		西里尔
KSC5601	949	8 位 ASCII 韩国语
MS874	874	MS-Win 泰国
SJIS	932	8 位 ASCII 日语
TIS620	874	泰国工业标准 620
UTF8	1208	UTF-8（IBM CCSID 1208，在 iSeries 服务器上尚不可用）
Unicode	13488	UNICODE，UCS-2
UnicodeBig	13488	与 Unicode 相同
UnicodeBigUnmarked		不带字节顺序标记的 Unicode
UnicodeLittle		最低有效位优先字节顺序的 Unicode
UnicodeLittleUnmarked		无字节顺序标记的 UnicodeLittle

有关缺省值，请参见缺省 file.encoding 值。

缺省 file.encoding 值: 此表显示当 Java(TM) 虚拟机启动时，如何根据 iSeries 编码字符集标识符（CCSID）设置 file.encoding 值。

iSeries CCSID	缺省 file.encoding	描述
37	ISO8859_1	美国、加拿大、新西兰和澳大利亚的英语；葡萄牙和巴西的葡萄牙语；以及荷兰的荷兰语
256	ISO8859_1	世界语 #1
273	ISO8859_1	德语 / 德国，德语 / 奥地利
277	ISO8859_1	丹麦语 / 丹麦，挪威语 / 挪威，挪威语 / 挪威 NY
278	ISO8859_1	芬兰语 / 芬兰
280	ISO8859_1	意大利语 / 意大利
284	ISO8859_1	加泰隆语 / 西班牙，西班牙语 / 西班牙
285	ISO8859_1	英语 / 英国，英语 / 爱尔兰
290	Cp943C	混合日语 EBCDIC 的 SBCS 部分（CCSID 5026）
297	ISO8859_1	法语 / 法国
420	Cp1046	阿拉伯语 / 埃及
423	ISO8859_7	希腊
424	ISO8859_8	希伯莱语 / 以色列
500	ISO8859_1	德语 / 瑞士，法语 / 比利时，法语 / 加拿大，法语 / 瑞士
833	Cp970	混合韩国语 EBCDIC 的 SBCS 部分（CCSID 933）
836	Cp1383	混合简体中文 EBCDIC 的 SBCS 部分（CCSID 935）
838	TIS620	泰语
870	ISO8859_2	捷克语 / 捷克共和国， 克罗地亚语 / 克罗地亚，匈牙利语 / 匈牙利，波兰语 / 波兰
871	ISO8859_1	冰岛语 / 冰岛
875	ISO8859_7	希腊语 / 希腊
880	ISO8859_5	保加利亚（ISO 8859_5）
905	ISO8859_9	扩展土耳其语
918	Cp868	乌尔都语
930	Cp943C	混合日语 EBCDIC（类似于 CCSID 5026）
933	Cp970	韩国语 / 韩国
935	Cp1383	简体中文
937	Cp950	繁体中文
939	Cp943C	混合日语 EBCDIC（类似于 CCSID 5035）

iSeries CCSID	缺省 file.encoding	描述
1025	ISO8859_5	白俄罗斯语 / 白俄罗斯，保加利亚语 / 保加利亚，马其顿语 / 马其顿，俄语 / 俄罗斯
1026	ISO8859_9	土耳其语 / 土耳其
1027	Cp943C	混合日语 EBCDIC 的 SBCS 部分（CCSID 5035）
1097	Cp1098	法尔西语
1112	Cp921	立陶宛语 / 立陶宛，拉脱维亚语 / 拉脱维亚，波罗的海语言
1388	GBK	混合简体中文 EBCDIC（包括 GBK）
5026	Cp943C	混合日语 EBCDIC CCSID（扩展片假名）
5035	Cp943C	混合日语 EBCDIC CCSID（扩展拉丁语）
8612	Cp1046	阿拉伯语（仅基本形式）（或 ASCII 420 和 8859_6）
9030	Cp874	泰语（主机扩展 SBCS）
13124	GBK	混合简体中文 EBCDIC 的 SBCS 部分（包括 GBK）
28709	Cp948	混合繁体中文 EBCDIC 的 SBCS 部分（CCSID 937）

发行版之间的兼容性

只要 Java(TM) 类文件不使用 Sun 已删除的或已更改了对其的支持的少数几个功能（参见 Sun 文档），它们就向上兼容（JDK 1.1.x -> 1.2.x -> 1.3.x » -> 1.4.x «）。只要程序只使用较早的 JDK 级别中的可用 Java 功能，类文件也向下兼容（» 1.4.x -> 1.3.x -> 1.2.x -> 1.1.x «）。由于 iSeries 服务器是一个相符实现，所以这在 iSeries 服务器上也成立。有关发行版之间的可用性的信息，参见 The Source for Java Technology java.sun.com 。

当使用 “创建 Java 程序”（CRTJVAPGM）命令来对 iSeries 服务器上的 Java 程序进行优化时，将把 JVAPGM 连接至类文件。在 V4R4 上，这些 JVAPGM 的内部结构已更改。这表示在 V4R4 之前创建的 JVAPGM 在 V4R4 和更新版本的发行版上无效。必须重建那些 JVAPGM。如果不执行任何操作，则系统自动创建具有以前的优化级别的 JVAPGM。然而，建议您手工执行 CRTJVAPGM，特别是对 JAR 或 ZIP 文件更是如此。这将产生最佳的优化，并且程序大小最小。

为了在优化级别 40 得到最佳的性能，建议每当 OS/400 发行版或 JDK 版本更改时都执行 CRTJVAPGM。由于在 CRTJVAPGM 上使用 JDKVER 设施将导致把 Sun JDK 中的方法直接插入到 JVAPGM 中，所以在这种情况下就更应该那样做。这将大大改进性能。然而，如果在后续的使那些直接插入无效的发行版上对 JDK 进行更改，则那些程序的实际运行速度可能比采用更低的优化级别还要慢。这是因为必须运行特殊的场合代码才能进行正确的操作。

有关更详细的性能信息，参见 Java 运行时性能。

使用 IBM Developer Kit for Java 进行数据库访问

借助 IBM Developer Kit for Java(TM), Java 程序可通过三种方法来访问数据库文件:

- JDBC 驱动程序说明了 IBM Developer Kit for Java JDBC 驱动程序是如何使 Java 程序能够访问数据库文件的。
- SQLJ 支持说明了 IBM Developer Kit for Java 是如何使您能够使用嵌入在 Java 应用程序中的 SQL 语句的。
- » Java SQL 例程说明了如何使用 Java 存储过程和 Java 用户定义函数来访问 Java 程序。«

使用 IBM Developer Kit for Java JDBC 驱动程序来访问 iSeries 数据库

IBM Developer Kit for Java(TM) JDBC 驱动程序（也称为“本机”驱动程序）提供了对 iSeries 数据库文件的程序式访问。通过使用“Java 数据库连接”（JDBC）API, 使用 Java 语言编写的应用程序可借助嵌入式“结构化查询语言”（SQL）来访问 JDBC 数据库功能、运行 SQL 语句、检索结果以及将更改传播回到数据库中。

» JDBC API 还可用来在分布式异构环境中与多个数据源交互作用。

JDBC API 所基于的 SQL99“命令语言接口”（CLI）是 ODBC 的基础。JDBC 提供了自然且易于使用的从 Java 编程语言到 SQL 标准中定义的抽象概念的映射。

要使用 JDBC 驱动程序, 请查看下列各项:

JDBC 入门
您可以遵循有关编写 JDBC 程序以及在 iSeries 服务器上运行该程序的教程。

连接
一个应用程序可以同时拥有多个连接。在 JDBC 中, 可使用 Connection 对象来表示与数据源的连接。用于面向数据库处理 SQL 语句的 Statement 对象是通过 Connection 对象创建的。

DatabaseMetaData
应用程序服务器和工具使用 DatabaseMetaData 接口来确定如何与给定数据源交互作用。应用程序还可使用 DatabaseMetaData 方法来获取关于特定数据源的信息。

异常
Java 语言使用异常来为它的程序提供错误处理能力。异常是在运行程序时发生的事件, 此事件会打断指令的正常流向。

事务
事务是逻辑工作单元。事务用来在并行访问期间提供数据完整性、正确的应用程序语义以及数据的一致性视图。所有与 JDBC 相符的驱动程序都必须支持事务。

语句类型
Statement 接口及其 PreparedStatement 和 CallableStatement 子类用来面向数据库处理 SQL 命令。SQL 语句导致生成 ResultSet 对象。

ResultSet
ResultSet 接口提供对通过运行查询生成的结果的访问。可以将 ResultSet 的数据想象成一个带有特定数目个列以及特定数目个行的表。缺省情况下, 表的行是按顺序检索的。在一行中, 可以按任何次序访问列值。

JDBC 对象合用

由于 JDBC 中使用的许多对象的创建成本都很高昂，如 Connection、Statement 和 ResultSet 对象，因此使用 JDBC 对象合用可以显著提高性能。借助对象合用，您可以重新使用这些对象，而不是在每次需要它们时都进行创建。

批处理更新

批处理更新支持允许对数据库所作的许多更新作为用户程序与数据库之间的单一事务传送。当必须同时执行许多更新时，批处理更新可以显著改进性能。

高级数据类型

在 iSeries 数据库中，提供了若干种称为 SQL3 数据类型的新数据类型。SQL3 数据类型提供了极大的灵活性。它们适合于存储序列化的 Java 对象、“可扩展标记语言”（XML）文档以及多媒体数据，如歌曲、产品图片、雇员照片和电影剪辑。SQL3 数据类型包括下列各项:

- 单值类型
- 大型对象，如“二进制大对象”、“字符大对象”和“双字节字符大对象”
- Datalink

RowSet

RowSet 规范设计成更象是一个框架而不是实际的实现。RowSet 接口定义了一组所有 RowSet 都拥有的核心功能。

分布式事务

“Java 事务 API”（JTA）支持复杂的事务。它还支持将事务与 Connection 对象分开。JTA 与 JDBC 配合工作以将事务与 Connection 对象分开，并允许单一连接并行地处理多个事务。反过来，也允许多个连接处理单一事务。

性能提示

借助这些性能提示，可以让 JDBC 应用程序获得最佳性能。

有关 JDBC 的更多信息，参阅 Sun Microsystems 的 JDBC 文档。

有关“iSeries 本机 JDBC”驱动程序的更多信息，参见 IBM Developer Kit for Java JDBC Web 页面。

JDBC 入门

Developer Kit for Java 附带交付的“Java(TM) 数据库连接”（JDBC）驱动程序称为 Developer Kit for Java JDBC 驱动程序。此驱动程序一般也称为本机 JDBC 驱动程序。

要选择适合您的需要的 JDBC 驱动程序，请考虑下列建议:

- 直接在数据库驻留所在的服务器上运行的程序应使用本机 JDBC 驱动程序以提高性能。这包括大多数 servlet 和 JavaServer Page（JSP）解决方案，以及编写为以本地方式在 iSeries 服务器上运行的应用程序。
- 必须连接至远程 iSeries 服务器的程序应使用 Toolbox JDBC 驱动程序。Toolbox 是健壮的 JDBC 实现，它是作为 Toolbox for Java 的一部分提供的。作为纯粹的 Java，为客户机设置 Toolbox JDBC 驱动程序是一件很容易的事情，并且，它只要求很少量的服务器设置。
- 在 iSeries 服务器上运行并需要连接至远程非 iSeries 数据库的程序应使用本机 JDBC 驱动程序并设置与该远程服务器的“分布式关系数据库体系结构”（DRDA）连接。

要开始使用 JDBC，参见下列各项:

JDBC 驱动程序的类型
此主题定义 JDBC 驱动程序类型。定义驱动程序类型的目的是对用来连接数据库的技术进行分类。

需求
此主题指示为了访问下列各项而需要满足的需求:

- 核心 JDBC
- JDBC 2.0 可选包
- Java 事务 API（JTA）

JDBC 教程
这是您朝着编写 JDBC 程序并在带有本机 JDBC 驱动程序的 iSeries 服务器上运行它所需迈出的重要第一步。

JDBC 驱动程序的类型:  本主题定义 “Java(TM) 数据库连接”（JDBC）驱动程序类型。驱动程序类型用来对用来连接数据库的技术进行分类。JDBC 驱动程序供应商使用这些类型来描述它们的产品的操作方式。对于一些应用程序而言，一些 JDBC 驱动程序类型比另外一些更为适合。

类型 1: “类型 1”驱动程序是“桥接”驱动程序。它们使用另一种技术如“开放数据库连接”（ODBC）来与数据库通信。由于许多“关系数据库管理系统”（RDBMS）平台都有 ODBC 驱动程序可用，所以这是一个优点。使用“Java 本机接口”（JNI）来从 JDBC 驱动程序调用 ODBC 函数。

在可以将 JDBC 与“类型 1”驱动程序配合使用之前，需要安装和配置桥接驱动程序。对于产品应用程序而言，这可能是一个严重的缺点。由于 applet 不能装入本机代码，所以“类型 1”驱动程序不能在 applet 中使用。

类型 2: “类型 2”驱动程序使用本机 API 来与数据库系统通信。使用 Java 本机方法来调用执行数据库操作的 API 函数。“类型 2”驱动程序通常比“类型 1”驱动程序快。

“类型 2”驱动程序要求安装和配置本机二进制代码才能工作。“类型 2”驱动程序也使用 JNI。由于 applet 不能装入本机代码，所以不能在 applet 中使用“类型 2”驱动程序。“类型 2”JDBC 驱动程序可能要求安装一些“数据库管理系统”（DBMS）联网软件。

Developer Kit for Java JDBC 驱动程序是“类型 2”JDBC 驱动程序。

类型 3: 这些驱动程序使用联网协议和中间件来与服务器通信。然后，服务器将协议转换成特定于 DBMS 的 DBMS 函数调用。

由于“类型 3”JDBC 驱动程序在客户机不需要任何本机二进制代码，所以它们是最灵活的 JDBC 解决方案。“类型 3”驱动程序不需要任何客户机安装。

类型 4: “类型 4”驱动程序使用 Java 来实现 DBMS 供应商联网协议。由于协议通常具有专利，所以 DBMS 供应商通常是唯一提供“类型 4”JDBC 驱动程序的公司。

“类型 4”驱动程序全都是 Java 驱动程序。这表示没有客户机安装或配置。然而，如果下层协议没有很好地处理诸如安全性和网络连接性之类的问题，则“类型 4”驱动程序可能不适合于一些应用程序。

Toolbox JDBC 驱动程序是“类型 4”JDBC 驱动程序，这表示 API 是纯 Java 联网协议驱动程序。

JDBC 需求: 在编写和部署 JDBC 应用程序之前，需要安装下列各项:

- 核心 JDBC
- JDBC 2.0 可选包
- Java 事务 API

核心 JDBC: 对于对本地数据库的核心“Java(TM) 数据库连接”（JDBC）访问，没有任何需求。所有支持都是内置的、预先已安装好并且已配置。

注意:

- 过去，需要添加 JDK 1.2 和更高版本的符号链接。对于 V4R5 PTF SF65439 和 V5R1 PTF SI00959，已除去此项需求。
- 过去，您必须确保“关系数据库目录”中至少有一个项才能连接至系统（通常是 *LOCAL，表示连接至本地数据库）。已解除此项需求。如果没有本地数据库的项，则访问本地系统时将使用系统的名称来创建该项。对于要使用本机 JDBC 驱动程序来连接的任何远程系统，仍必须配置“关系数据库目录”项。

JDBC 2.0 可选包: 如果需要使用 JDBC 2.0 可选包的类，则必须在类路径中包括 jdbc2_0-stdext.jar 文件。这个“Java 压缩文档”（JAR）文件包含将应用程序编写为使用 JDBC 2.0 可选包所需的所有标准接口。要将这个 JAR 文件添加至扩展类路径，请创建从 UserData 扩展目录到这个 JAR 文件的位置的符号链接。您只需要将此操作执行一次；在运行时，JDBC 2.0 可选包 JAR 文件始终可供应用程序使用。使用以下命令来将可选包添加至扩展类路径:

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

注意: 此项需求仅用于 JDK 1.2 和 1.3。由于 JDK 1.4 是第一个带有 JDBC 3.0 支持的发行版，所以所有 JDBC（即核心 JDBC 和可选包）都已移到基本 JDK 运行时 JAR 文件中，程序总是能够找到这个 JAR 文件。

Java 事务 API: 如果需要在应用程序中使用“Java 事务 API”（JTA），则必须在类路径中包括 jta-spec1_0_1.jar 文件。这个 JAR 文件包含将应用程序编写为使用 JTA 所需的所有标准接口。要将这个 JAR 文件添加至扩展类路径，请创建从 UserData 扩展目录到这个 JAR 文件的位置的符号链接。这是一次性的操作，一旦完成，JTA JAR 文件在运行时就始终可供应用程序使用。使用以下命令来将 JTA 添加至扩展类路径:

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

JDBC 相符性: 本机 JDBC 驱动程序与所有相关 JDBC 规范相符。JDBC 驱动程序的相符级别并不依赖于 OS/400 发行版，而是依赖于所使用的 JDK 发行版。各种 JDK 的本机 JDBC 驱动程序相符级别列示如下:

JDK 发行版	JDBC 驱动程序的相符级别
JDK 1.1	此 JDK 与 JDBC 1.0 相符。
JDK 1.2	此 JDK 与 JDBC 2.0 相符并支持 JDBC 2.1 可选包。
JDK 1.3	此 JDK 与 JDBC 2.0 相符并支持 JDBC 2.1 可选包（对于 JDK 1.3，没有与 JDBC 相关的更改）。
JDK 1.4	此 JDK 与 JDBC 3.0 相符，但不再存在 JDBC 可选包（对它的支持现在是核心 JDK 的一部分）。

JDBC 教程: 下面是有关如何编写“Java(TM) 数据库连接”（JDBC）程序并在带有本机 JDBC 驱动程序的 iSeries 服务器上运行该程序的教程。本教程设计为向您展示要让程序运行 JDBC 所必需的基本步骤。

示例程序创建一个表并将一些数据植入该表。程序处理查询以从数据库中获取该数据并将其显示在屏幕上。

运行示例程序: 要运行示例程序，请执行下列步骤:

1. 将程序复制到工作站。
   a. 复制示例程序并将其粘贴到工作站上的某个文件中。
   b. 将该文件保存为与所提供的公用类同名并具有 .java 扩展名。在本案例中，必须在本地工作站上将文件命名为 BasicJDBC.java。
2. 将文件从工作站传送至 iSeries 服务器。从命令提示，输入下列命令:

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

   要使这些命令能够起作用，必须有用于存放文件的目录。在示例中，/home/cujo 就是该位置，但可使用您所要的任何位置。

   注意: 根据您的服务器的设置方式的不同，前面提到的 FTP 命令也有可能不同，但它们应该是类似的。只要将文件传送到集成文件系统中，如何将文件传送至 iSeries 服务器并不要紧。诸如 VisualAge for Java 之类的工具可以将此过程完全自动化。
3. 确保将类路径设置为存放文件的目录，以便运行 Java 命令时它们能够找到该文件。从 CL 命令行，可使用 WRKENVVAR 来查看为您的用户概要文件设置了哪些环境变量。
   - 如果您看到名为 CLASSPATH 的环境变量，则必须确保用于存放 .java 文件的位置位于该环境变量中列示的目录字符串中，如果尚未指定该位置，则必须添加它。
   - 如果没有 CLASSPATH 环境变量，则必须添加一个。必须使用以下命令来完成此操作:

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

   注意: 要从 CL 命令编译 Java 代码，必须包括 tools.jar 文件。这个 JAR 文件包含 javac 命令。
4. 将 Java 文件编译成类文件。
   从 CL 命令行输入以下命令:

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

   还可以从 QSH 中编译 Java 文件:

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

   QSH 自动确保可找到 tools.jar 文件。因此，不必将其添加至类路径。当前目录也位于类路径中。通过发出更改目录（cd）命令，还可找到 BasicJDBC.java 文件。

   注意: 还可在工作站上编译文件并使用 FTP 来将类文件以二进制方式发送至 iSeries 服务器。这就是 Java 能够在任何平台上运行的一个示例。通过从 CL 命令行或从 QSH 中使用以下命令来运行程序:

```
java BasicJDBC
```

   输出如下所示:

```
----------------------
| 1 | Frank Johnson  |
| 2 | Neil Schwartz  |
| 3 | Ben Rodman     |
```

```
| 4 | Dan Gloore     |
----------------------
There were 4 rows returned.
Output is complete.
Java program completed.
```

参考: 有关 Java 和 JDBC 的更多信息，请参考下列资源:

- 本机 JDBC 驱动程序外部 Web 站点 
- Toolbox JDBC 驱动程序外部 Web 站点 
- Sun 的 JDBC 页面 
- iSeries 和 iSeries 用户的 Java/JDBC 论坛
- IBM JDBC 电子邮件地址

示例: JDBC:  这是有关如何使用 BasicJDBC 程序的示例。

示例: BasicJDBC

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
////////////////////////////////////////////////////////////////////////////
//
// 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 the
        // way it should. 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 qgpl.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();
        }
    }
}
```

将 JNDI 用于示例:  DataSource 与 “Java(TM) 命名和目录接口” （JNDI）携手工作。JNDI 是目录服务的 Java 抽象层，就象 “Java 数据库连接” （JDBC）是数据库的抽象层一样。JNDI 最经常与 “轻量级目录访问协议” （LDAP）配合使用，但它也可以与 “CORBA 对象服务” （COS）、 “Java 远程方法调用” （RMI）注册表或下层文件系统配合使用。这些各式各样的使用是通过各种目录服务提供程序完成的（目录服务提供程序将公共 JNDI 请求转换成特定目录服务请求）。

DataSource 样本是使用 JNDI 文件系统服务提供程序设计的。如果您想要运行所提供的示例，则必须有就绪的 JNDI 服务提供程序。

请遵循下面这些指导来为文件系统服务提供程序设置环境:

1. 从 Sun Microsystems 的 JNDI站点  下载文件系统 JNDI 支持。
2. 单击**继续**以下载 JNDI 1.2.1。将显示许可证协议。
3. 单击**接受**，然后单击 **FS 上下文**以显示 JNDI 上下文支持的下载选项。
4. 下载 fscontext.zip 并从工作站将该文件解压缩（unzip）。
5. 使用 FTP 来将 fscontext.jar 和 providerutil.jar 传送至系统并将它们放在 /QIBM/UserData/Java400/ext 中。这是扩展目录，在运行应用程序时，能够自动找到您放在这里的 JAR 文件（即无需将它们包括在类路径中）。

在有了对 JNDI 的服务提供程序的支持之后，必须为应用程序设置上下文信息。这可以通过将必需的信息置于 SystemDefault.properties 文件中来完成。在系统上，可以在若干个位置中指定缺省属性，但最好的方法是在主目录（即 /home/）中创建名为 SystemDefault.properties 的文本文件。

要创建文件，请使用下列各行或将它们添加至现有文件。

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

这些行指定文件系统服务提供程序处理 JNDI 请求，并且 /DataSources/jdbc 是使用 JNDI 的任务的根。可以更改此位置，但所指定的目录必须存在。所指定的位置就是绑定和部署示例 DataSource 的位置。 

连接

 Connection 对象表示与 “Java(TM) 数据库连接” （JDBC）中的数据源的连接。用于面向数据库处理 SQL 语句的 Statement 对象是通过 Connection 对象创建的。一个应用程序可以同时拥有多个连接。这些 Connection 对象可以全都连接至同一个数据库，也可以连接至不同的数据库。

在 JDBC 中获取连接可以通过两种方法完成:

- 通过 DriverManager 类。
- 通过使用 DataSource。

由于使用 DataSource 来获取连接能够增强应用程序的可移植性和可维护性，所以这是首选方法。它还允许应用程序透明地使用连接和语句合用以及分布式事务。

有关获取连接的详细信息，参见下列各节:

DriverManager
DriverManager 是一个静态类，它管理可供应用程序使用的 JDBC 驱动程序集合。

Connection 属性
这个表列示了有效的 JDBC 驱动程序连接属性、它们的值以及它们的描述。

将 DataSource 与 UDBDataSource 配合使用
通过将 DataSource 设置为具有特定的属性，然后使用“Java 命名和目录接口”（JNDI）将其绑定到一些目录服务中，可以将其与 UDBDataSource 类一起部署。

DataSource 属性
这个表列示了有效的 DataSource 属性、它们的值以及它们的描述。

其它 DataSource 实现
本机 JDBC 驱动程序附带提供了 DataSource 接口的其它实现。在采用 UDBDataSource 及其相关功能之前，这些实现只是用作桥梁。

在获取连接之后，可使用它来完成下列 JDBC 任务:

- 创建各种类型的 Statement 对象，以便与数据库交互作用。
- 控制面向数据库的事务。
- 检索关于数据库的元数据。

DriverManager: DriverManager 是 Java(TM) Developer Kit（JDK）提供的静态类。这个类负责管理可供应用程序使用的“Java 数据库连接”（JDBC）驱动程序集合。如果有必要的话，应用程序可并行地使用多个 JDBC 驱动程序。应用程序通过使用“统一资源定位器”（URL）来指定 JDBC 驱动程序。通过将特定 JDBC 驱动程序的 URL 传送给 DriverManager，应用程序通知 DriverManager 应该将什么类型的 JDBC 连接返回给应用程序。

在可以执行此操作之前，必须让 DriverManager 知道有哪些 JDBC 驱动程序可用以使其可以交出连接。通过调用 Class.forName 方法，DriverManager 根据传送给该方法的类字符串名称来将一个类装入到正在运行的 Java 虚拟机（JVM）中。下面是使用 class.forName 方法来装入本机 JDBC 驱动程序的示例:

示例: 装入本机 JDBC 驱动程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

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

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

JDBC 驱动程序设计成在它们的驱动程序实现类装入时自动告知 DriverManager 关于它们自己的信息。在处理先前提到的代码行之后，本机 JDBC 驱动程序便可供与之配合工作的 DriverManager 使用。以下代码行使用本机 JDBC URL 来请求 Connection 对象:

示例: 请求 Connection 对象

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

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

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

JDBC URL 的最简单形式是由冒号分隔的三个值的列表。列表中的第一个值表示协议，对于 JDBC URL，它总是 jdbc。第二个值是子协议，使用 db2 来指定本机 JDBC 驱动程序。第三个值是系统名，用于建立与特定系统的连接。可使用两个特殊值来连接至本地数据库。它们是 *LOCAL 和 localhost（这两个值都不区分大小写）。还可提供特定的系统名，如下所示:

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

这将创建与 rchasmop 系统的连接。如果您正在尝试连接的系统是远程系统（例如，通过 “分布式关系数据库体系结构” ），则必须使用关系数据库目录中的系统名。

注意: 在没有指定时，还将使用当前用来注册的用户标识和密码来建立与数据库的连接。

属性: DriverManager.getConnection 方法接收先前指示的单一字符串 URL，它只是 DriverManager 上的其中一个用于获取 Connection 对象的方法。DriverManager.getConnection 方法还有一个版本，该版本接收用户标识和密码。以下是此版本的一个示例:

示例: 接收用户标识和密码的 DriverManager.getConnection 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

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

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

此代码行尝试作为用户 cujo 并使用密码 newtiger 来连接至本地数据库，而不考虑谁正在运行应用程序。DriverManager.getConnection 方法还有一个版本，该版本接收 java.util.Properties 对象以允许进行进一步的定制。下面是一个示例:

示例: 接收 java.util.Properties 对象的 DriverManager.getConnection 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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);
```

此代码在功能上等价于前面提到的将用户标识和密码作为参数传送的版本。

请参考 Connection 属性以获取本机 JDBC 驱动程序的连接属性的完整列表。

URL 属性: 另一种指定属性的方法是将属性放在 URL 对象本身上的列表中。列表中的每个属性都由分号分隔，该列表必须具有属性名 = 属性值格式。如以下示例所示，这仅仅是一种快捷方式，并不会显著更改执行处理的方式:

示例: 指定 URL 属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

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

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

此代码在功能上再次与上面提到的示例等价。

如果同时在属性对象中和 URL 对象上指定了某个属性值，则 URL 版本优先于属性对象中的版本。下面是一个示例:

示例: URL 属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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);
```

此示例将使用 URL 字符串中的用户标识和密码，而不是使用 Properties 对象中的版本。这导致在功能上与前面提到的代码等价。

有关更多信息，参见下列示例:

- 并行地使用本机 JDBC 和 Toolbox JDBC
- Access 属性
- 无效的用户标识和密码

示例: 并行地使用本机 JDBC 和 Toolbox JDBC:  这是有关如何在程序中使用本机 JDBC 连接和 Toolbox JDBC 连接的示例。

示例: 并行地使用本机 JDBC 和 Toolbox JDBC

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
////////////////////////////////////////////////////////////////////////////
//
// 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 Toolbox
// JDBC connection.
//
// This technique is convenient because it allows you to use different
// JDBC drivers for different tasks concurrently. For example, the
// Toolbox 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.AS400JDBCConnection)
            System.out.println("conn2 is running under the Toolbox 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());
      }
   }
}
```

«

示例: Access 属性: » 这是有关如何使用 Access 属性的示例。

示例: Access 属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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.
        }
    }
}
```

«

示例: 无效的用户标识和密码: 这是有关如何以 SQL 命名方式使用 Connection 属性的示例。

示例: 无效的用户标识和密码

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
///////////////////////////////////////////////////////////////////////////////
//
// 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());
        }

    }
}
```

Connection 属性: 这个表包含有效的 JDBC 驱动程序连接属性、它们的值以及它们的描述:

属性	值	含义
″access″	″all″, ″read call″, ″read only″	这个值可用来限制可以对特定连接执行的操作的类型。缺省值为 ″all″，并且基本上表示该连接对 JDBC API 具有完全的访问权。″read call″ 值只允许连接执行查询和调用存储过程。通过 SQL 语句来更新数据库的尝试将停止。″read only″ 值可用来将连接限制为只能执行查询。存储过程调用和更新语句将停止。
 ″batch style″	″2.0″, ″2.1″	JDBC 2.1 规范定义了另一个方法来控制对批处理更新中的异常的处理。驱动程序可遵从这两个方法中的任何之一。缺省情况是按照 JDBC 2.0 规范中的定义来进行工作。 
″block size″	″0″, ″8″, ″16″, ″32″, ″64″, ″128″, ″256″, ″512″	这是对结果集每次提取的行数。对于结果集的典型仅向前处理，将获取具有此大小的块。于是，当应用程序处理每一行时，并不会访问数据库。仅当到达块的末尾时，数据库才会请求另一个数据块。 仅当将 ″blocking enabled″ 属性设置为 true 时才使用此值。 将块大小属性设置为 ″0″ 的效果与将 ″blocking enabled″ 属性设置为 ″false″ 的效果相同。 缺省情况是使用块大小为 ″32″ 的分块。目前，这是一个相当任意的决定，将来可能会更改缺省值。 目前，对可滚动结果集不使用分块。
″blocking enabled″	″true″, ″false″	这个值用来确定连接是否应该对结果集行检索使用分块。分块可以显著改进处理结果集的性能。 缺省情况下，此属性设置为 true。

属性	值	含义
» "cursor hold"	"true", "false"	这个值指定在提交事务时是否应保持结果集处于打开状态。true 值表示应用程序在调用 commit 后能够访问其打开的结果集。false 值表示 commit 将关闭该连接之下打开的任何游标。 缺省情况下，此属性设置为 true。 这个值属性用作为连接创建的所有结果集的缺省值。借助 JDBC 3.0 中添加的游标可保持性支持，如果应用程序以后指定另一可保持性，则将替换这个缺省值。 «
» "data truncation"	"true", "false"	这个值指定字符数据的截断是应该导致生成警告和异常（true）还是应该仅仅是安静地截断数据（false）。如果缺省值是 true，则应顾及字符字段的数据截断。 «
» "date format"	"julian", "mdy", "dmy", "ymd", "usa", "iso", "eur", "jis"	此属性允许您更改日期的格式化方式。 «
» "date separator"	"/", "-", ".", ",", "b"	此属性允许您更改日期分隔符。这只有在与一些 dateFormat 值组合使用时才有效（视系统规则而定）。 «
» "decimal separator"	".", ","	此属性允许您更改小数分隔符。 «

属性	值	含义
″do escape processing″	″true″, ″false″	此属性设置一个标志，以指示在连接之下的语句是否必须执行转义处理。使用转义处理是一种编码 SQL 语句的方法，借助这种方法，SQL 语句对所有平台都具有一般性和类似性，但数据库接着读取转义子句并将其替代为用户的正确系统特定版本。 除了必须要在系统上执行一些额外的工作之外，各方面都不错。如果您知道您只使用已包含有效 iSeries SQL 语法的 SQL 语句，则建议将此值设置为 ″false″ 以提高性能。 由于必须与 JDBC 规范相符（即，转义处理在缺省情况下处于活动状态），所以此属性的缺省值是 true。 添加这个值的原因是 JDBC 规范存在一个缺点。只能在 Statement 类中关闭转义处理。如果处理的是简单的语句，则这样做的效果不错。您创建语句、关闭转义处理并开始处理语句。然而，对于准备语句和可调用语句，此方案不起作用。您在构造准备语句或可调用语句时提供 SQL 语句，之后不进行更改。因此语句是预先准备的，之后更改转义处理是没有意义的。有了此连接属性就有了避开额外开销的方法。
″errors″	″basic″, ″full″	此属性允许在 SQLException 对象消息中返回完全的系统辅助级错误文本。缺省值是 basic，即只返回标准消息文本。

属性	值	含义
″libraries″	空格分隔的库列表。（库列表也可以由冒号或逗号分隔。）	此属性允许将库列表放到服务器作业的库列表中，或设置特定的缺省库。 ″naming″ 属性影响此属性的工作方式。在缺省情况下（即 ″naming″ 设置为 ″sql″），JDBC 的工作方式与 ODBC 相似。库列表不影响连接的处理方式。所有未限定的表都有缺省库。缺省情况下，那个库与连接的用户概要文件同名。如果指定了 libraries 属性，则列表中的第一个库成为缺省库。如果在连接 URL 上指定了缺省库（如在 “jdbc:db2:*local/mylibrary” 中），则该缺省库将覆盖此属性中的任何值。 如果将 ″naming″ 设置为 ″system″，则将把对此属性指定的每个库添加至用户的库列表，并搜索库列表来对未限定的表引用进行解析。
″lob threshold″	任何小于 500000 的值	此属性告诉驱动程序: 如果 LOB 列小于阈值大小，则将实际值而不是 LOB 列的定位器放到结果集存储器中。此属性对列大小操作而不是对 LOB 数据大小本身操作。例如，即使 LOB 定义为对每个 LOB 最多存放 1 MB，但所有列值都在 500 KB 之下，也仍使用定位器。 注意，大小限制设置为允许提取数据块，但始终不会导致数据块增大为超过 16 MB 最大分配大小。对于大型结果集，仍很容易超出此限制，这将导致提取失败。您必须小心地处理块大小属性和此属性与数据块大小的交互作用。 缺省值是 0。即始终将定位器用于 LOB 数据。
″naming″	″sql″，″system″	此属性允许您使用传统 iSeries 命名语法或标准 SQL 命名语法。″system″ 命名表示应使用 “/” 字符来分隔集合和表值，″sql″ 命名表示应使用 “.” 字符来分隔值。 这个值的设置还会对缺省库产生影响。有关这方面的进一步信息，参见 库属性。 缺省情况是使用 ″sql″ 命名。

属性	值	含义
″password″	任何内容	此属性允许对连接指定密码。如果不同时指定 ″user″ 属性，则此属性不能正确工作。这些属性允许您作为不是运行 iSeries 作业的用户来连接至数据库。 指定 ″user″ 和 ″password″ 属性的效果与将连接方法与签名 getConnection(String url, String userId, String password) 配合使用相同。
» ″prefetch″	″true″, ″false″	此属性指定驱动程序是应该在处理之后立即提取结果集的最初数据还是要等待到请求数据时才提取数据。如果缺省值为 true，则应预提取数据。 对于使用 “本机 JDBC” 驱动程序的应用程序，这很少会成为问题。此属性主要与某些 Java 存储过程和用户定义函数进行内部配合使用，对于这些 Java 存储过程和用户定义函数，在您请求数据之前，数据库引擎不代替您从结果集中提取任何数据至关重要。 «
» ″reuse objects″	″true″, ″false″	此属性指定在您关闭一些类型的对象之后驱动程序是否应尝试重新使用它们。这将提高性能。缺省值是 true。 «
» ″server trace″	整数的字符串表示法	此属性启用对 JDBC 服务器作业的跟踪。如果启用服务器跟踪，则将在客户机连接至服务器时启动跟踪并在断开连接时结束跟踪。 将把跟踪数据收集到服务器上的假脱机文件中。通过添加常量并在 set 方法上传送该和数，可以打开多个级别的服务器跟踪。 **注意**：此属性通常由支持人员使用，这里不进一步讨论它的值。 «
» ″time format″	″hms″, ″usa″, ″iso″, ″eur″, ″jis″	此属性允许您更改时间的格式化方式。 «
» ″time separator″	″:″, ″.″, ″,″, ″b″	此属性允许您更改时间分隔符。这只有在与一些 timeFormat 值组合使用时才有效（视系统规则而定）。 «
″trace″	″true″, ″false″	此属性允许打开连接跟踪。它可用作简单的调试辅助。我们有将来增强此功能的想法。有关调试的更多信息，参见 D2. JDBC 驱动程序抛出异常。我怎么做？。 缺省值是 ″false″，即不使用跟踪。

属性	值	含义
″transaction isolation″	″none″, ″read committed″, ″read uncommitted″, ″repeatable read″, ″serializable″	此属性允许您设置连接的事务隔离级别。在将此属性设置为特定级别与在 Connection 接口中的 setTransactionIsolation 方法上指定某个级别之间并无差别。 由于 JDBC 在缺省情况下具有自动提交方式，所以此属性的缺省值是 ″none″。
″translate binary″	″true″, ″false″	此属性可用来强制 JDBC 驱动程序将二进制和 varbinary 数据值视为 char 和 varchar 数据值。 此属性的缺省值是 ″false″，即不将二进制数据视为与字符数据相同。
» ″use block insert″	″true″, ″false″	此属性允许本机 JDBC 驱动程序进入块插入方式来将数据块插入数据库。这是批处理更新的优化版本。这种优化方式只能在确保不会违反特定系统约束或发生数据插入故障和潜在地损坏数据的应用程序中使用。 打开此属性的应用程序只应该在尝试执行批处理更新时才连接至本地系统。由于不能通过 DRDA 来管理分块插入，所以这些应用程序不应使用 DRDA 来建立远程连接。 应用程序还必须确保带有 SQL insert 语句和 values 子句的 PreparedStatement 包含所有 insert 和 values 参数。values 列表中不允许常量。这是系统的分块插入引擎的一项要求。 缺省值是 false。 «
″user″	任何内容	此属性允许对连接指定用户标识。如果不同时指定 ″password″ 属性，则此属性不能正确工作。这些属性允许您作为不是运行 iSeries 作业的用户来连接至数据库。 指定 ″user″ 和 ″password″ 属性的效果与将连接方法与签名 getConnection(String url, String userId, String password) 配合使用相同。

示例: 创建 UDBDataSource 并将其与 JNDI 绑定: » 这是有关如何创建 UDBDataSource 并将其与 JNDI 绑定的示例。

示例: 创建 UDBDataSource 并将其与 JNDI 绑定

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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);
    }
}
```

示例: 创建 UDBDataSourceBind 并设置 DataSource 属性: 这是有关如何创建 UDBDataSource 并设置用户标识和密码作为 DataSource 属性的示例。

示例: 创建 UDBDataSourceBind 并设置 DataSource 属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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);
    }
}
```

示例: 在绑定 UDBDataSource 之前获取初始上下文: 以下示例在绑定 UDBDataSource 之前获取初始上下文。然后，对该上下文使用 lookup 方法来返回 DataSource 类型的对象来供应用程序使用。

示例: 在绑定 UDBDataSource 之前获取初始上下文

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
    }
}
```

示例: 创建 UDBDataSource 并获取用户标识和密码: 这是有关如何创建 UDBDataSource 并使用 getConnection 方法来在运行时获取用户标识和密码的示例。

示例: 创建 UDBDataSource 并获取用户标识和密码

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/// 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();
    }
}
```

将 DataSource 与 UDBDataSource 配合使用: DataSource 接口设计成使您能够相当灵活地使用“Java(TM) 数据库连接”（JDBC）驱动程序。DataSource 的使用可以分成两个阶段:

- **部署**
  部署是在实际运行 JDBC 应用程序之前发生的设置阶段。部署通常涉及将 DataSource 设置为具有特定的属性并接着使用“Java 命名和目录接口”（JNDI）来将其绑定到某个目录服务中。目录服务通常是“轻量级目录访问协议”（LDAP），但也可以是许多其它的目录服务，如“公共对象请求代理体系结构（CORBA）对象服务”、“Java 远程方法调用”（RMI）或下层文件系统。
- **使用**
  通过将 DataSource 的部署与运行时使用分开，许多应用程序可以对 DataSource 设置进行重复使用。通过更改部署的某些方面，所有使用该 DataSource 的应用程序都能够自动获得那些更改。

DataSource 的一个优点是它们允许 JDBC 驱动程序代替应用程序工作，并且不会直接影响应用程序开发过程。有关更多信息，参见连接池、语句合用和分布式事务。

UDBDataSourceBind: UDBDataSourceBind 程序是一个有关创建 UDBDataSource 并将其与 JNDI 绑定的示例。此程序完成所请求的所有基本任务。即，此程序将 UDBDataSource 对象实例化、对此对象设置属性、检索 JNDI 上下文并将该对象绑定至 JNDI 上下文内的一个名称。

部署时代码随供应商的不同而不同。应用程序必须导入它要使用的特定 DataSource 实现。在导入列表中，将导入由包限定的 UDBDataSource 类。在此应用程序中，您最不熟悉的部分应该是使用 JNDI 完成的工作（例如，检索 Context 对象和对 bind 的调用）。有关附加信息，参见 Sun Microsystems 的 JNDI  。

在此程序运行并成功完成之后，JNDI 目录服务中便有一个名为 SimpleDS 的新项。这个项位于 JNDI 上下文指定的位置中。现在，部署 DataSource 实现。应用程序可使用 DataSource 来检索数据库连接以及与 JDBC 相关的工作。

UDBDataSourceUse: UDBDataSourceUse 程序是使用先前部署的应用程序的 JDBC 应用程序示例。

此 JDBC 应用程序获取初始上下文，就象前一示例中它在绑定 UDBDataSource 之前所做的那样。然后，对该上下文使用 lookup 方法来返回 DataSource 类型的对象来供应用程序使用。

注意: 运行时应用程序只对 DataSource 接口的方法感兴趣，因此它无需了解实现类。这使应用程序具有可移植性。

假定 UDBDataSourceUse 是一个复杂的应用程序，它需要运行您的组织中的大型操作。您的组织中有成打或更多的类似大型应用程序。您必须更改网络中其中一个系统的名称。通过运行部署工具并更改单一 UDBDataSource 属性，就能够在所有应用程序中获得这种新行为，而不必更改应用程序代码。DataSource 的其中一个好处是允许您合并系统设置信息。另一项主要好处是允许应用程序实现对应用程序不可视的功能，如连接池、语句合用以及支持分布式事务。

在密切分析 UDBDataSourceBind 和 UDBDataSourceUse 之后，您可能会觉得奇怪: DataSource 对象是怎么知道要执行什么操作的？在这些程序的任何之一中都没有用于指定系统、用户标识或密码的代码。 UDBDataSource 类的所有属性都具有缺省值；缺省情况下，它使用正在运行的应用程序的用户概要文件和密码来连接本地 iSeries 服务器。如果要确保使用用户概要文件 cujo 来进行连接，则可通过两种方法完成此操作:

- 将用户标识和密码设置为 DataSource 属性。参见示例: 创建 UDBDataSourceBind 并设置 DataSource 属性以了解如何使用此项技术。
- 使用在运行时接收用户标识和密码的 DataSource getConnection 方法。参见示例: 创建 UDBDataSource 并获取用户标识和密码以了解如何使用此项技术。

就象可以对使用 DriverManager 创建的连接指定属性一样，可以对 UDBDataSource 指定许多属性。请参考 DataSource 属性以获取对本机 JDBC 驱动程序支持的属性的列表。

尽管这些列表很相似，但在将来的发行版中并不一定会相似。我们鼓励您开始对 DataSource 接口进行编码。

注意: 本机 JDBC 驱动程序还有另外两个 DataSource 实现，但不建议直接使用它们。

- DB2DataSource
- DB2StdDataSource

DataSource 属性: 这个表包含有效的数据源属性、它们的值以及它们的描述:

Set 方法（数据类型）	值	描述
setAccess （String）	″all″，″read call″，″read only″	这个属性可用来限制可以对特定连接执行的操作的类型。缺省值为 ″all″，并且基本上表示该连接对 “Java(TM) 数据库连接”（JDBC）API 具有完全的访问权。 ″read call″ 值只允许连接执行查询和调用存储过程。通过 SQL 语句来更新数据库的尝试将导致 SQLException。 ″read only″ 值将连接限制为只能执行查询。尝试处理存储过程调用或更新语句将导致 SQLException。
setBatchStyle （String）	″2.0″，″2.1″	JDBC 2.1 规范定义了另一个方法来控制对批处理更新中的异常的处理。驱动程序可遵从这两个方法中的任何之一。缺省情况是按照 JDBC 2.0 规范中的定义来进行工作。
setUseBlocking （boolean）	″true″，″false″	这个属性用来确定连接是否应该对结果集行检索使用分块。分块可以显著改进处理结果集的性能。 缺省情况下，此属性设置为 true。
setBlockSize （int）	″0″，″8″，″16″，″32″，″64″，″128″，″256″，″512″	这个属性指示对结果集每次提取的行数。对于结果集的典型仅向前处理，如果数据库包含的满足查询的行有那么多的话，则将获取具有此大小的块。仅当在 JDBC 驱动程序的内部存储器中到达块的末尾时，才将另一个数据块请求发送至数据库。 仅当将 useBlocking 属性设置为 true 时才使用此值。有关更多信息，请参考 setUseBlocking。 将块大小属性设置为 ″0″ 的效果与调用 setUseBlocking(false) 相同。 缺省情况是使用块大小为 ″32″ 的分块。这是一个相当任意的决定，在将来的发行版中可能会更改缺省值。 目前，对可滚动结果集不使用分块。 使用分块将影响用户应用程序所具有的游标灵敏度的程度。灵敏游标可以看到其它 SQL 语句所作的更改。然而，由于进行了数据高速缓存，所以仅当需要从数据库提取数据时才检测更改。

Set 方法（数据类型）	值	描述
setCursorHold（boolean）	″true″, ″false″	这个属性指定在提交事务时是否应保持结果集处于打开状态。true 值表示应用程序在调用 commit 后能够访问其打开的结果集。false 值表示 commit 将关闭该连接之下打开的任何游标。 缺省情况下，此属性设置为 true。 这个属性用作为连接创建的所有结果集的缺省值。借助 JDBC 3.0 中添加的游标支持（有关详细信息，参见 ResultSet 特征一节），如果应用程序以后指定另一游标支持，则将替换这个缺省值。
setDataTruncation（boolean）	″true″, ″false″	这个属性指定下列各项: • 字符数据的截断是否应该导致生成警告和异常（true） • 是否应该仅仅是安静地截断数据（false）。 有关附加的详细信息，参见 DataTruncation。
setDatabaseName（String）	任何名称	此属性指定 DataSource 尝试连接的数据库。缺省值是 *LOCAL。数据库名必须存在于运行应用程序的系统上的关系数据库目录中，或者是特殊值 *LOCAL 或 localhost 以指定本地系统。
setDataSourceName（String）	任何名称	此属性允许传送 ConnectionPoolDataSource “Java 命名和目录接口”（JNDI）名称以支持连接池。
setDateFormat（String）	″julian″, ″mdy″, ″dmy″, ″ymd″, ″usa″, ″iso″, ″eur″, ″jis″	此属性允许您更改日期的格式化方式。
setDateSeparator（String）	″/″, ″-″, ″.″, ″,″, ″b″	此属性允许您更改日期分隔符。这只有在与一些 dateFormat 值组合使用时才有效（视系统规则而定）。
setDecimalSeparator（String）	″.″, ″,″	此属性允许您更改小数分隔符。
setDescription（String）	任何名称	此属性允许设置此 DataSource 对象的文本描述。
setDoEscapeProcessing（boolean）	″true″, ″false″	此属性指定是否对 SQL 语句执行转义处理。 此属性的缺省值是 true。
setFullErrors（boolean）	″true″, ″false″	此属性允许在 SQLException 对象消息中返回整个系统的辅助级错误文本。缺省值是 false。
setLibraries（String）	空格分隔的库列表	此属性允许将库列表放到服务器作业的库列表中。仅当使用了 setSystemNaming(true) 时才使用此属性。

Set 方法（数据类型）	值	描述
setLobThreshold（int）	任何小于 500000 的值	此属性告诉驱动程序: 如果 LOB 列小于阈值大小，则存放实际值而不是“定位器对象”（LOB）定位器。
setLoginTimeout（int）	任何值	目前忽略此属性，计划将来使用它。
setNetworkProtocol（int）	任何值	目前忽略此属性，计划将来使用它。
setPassword（String）	任何字符串	此属性允许对连接指定密码。如果未设置用户标识，则忽略此属性。
setPortNumber（int）	任何值	目前忽略此属性，计划将来使用它。
setPrefetch（boolean）	″true″，″false″	此属性指定驱动程序是应该在处理之后立即提取结果集的最初数据还是要等待到请求数据时才提取数据。缺省值是 true。
setReuseObjects（boolean）	″true″，″false″	此属性指定在您关闭一些类型的对象之后驱动程序是否应尝试重新使用它们。这将提高性能。缺省值是 true。
setServerName（String）	任何名称	目前忽略此属性，计划将来使用它。
setServerTraceCategories（int）	整数的字符串表示法	此属性启用对 JDBC 服务器作业的跟踪。如果启用服务器跟踪，则将在客户机连接至服务器时启动跟踪并在断开连接时结束跟踪。 将把跟踪数据收集到服务器上的假脱机文件中。通过添加常量并在 set 方法上传送该和数，可以打开多个级别的服务器跟踪。 **注意**: 此属性通常由支持人员使用，这里不进一步讨论它的值。
setSystemNaming（boolean）	″true″，″false″	此属性允许指定是应该使用句点（SQL 命名）还是应该使用斜杠（系统命名）来分隔集合和表。此属性还确定是使用缺省库（SQL 命名）还是使用库列表（系统命名）。缺省值是 SQL 命名。
setTimeFormat（String）	″hms″，″usa″，″iso″，″eur″，″jis″	此属性允许您更改时间的格式化方式。
setTimeSeparator（String）	″:″，″.″，″,″，″b″	此属性允许您更改时间分隔符。这只有在与一些 timeFormat 值组合使用时才有效（视系统规则而定）。
setTrace（boolean）	″true″，″false″	此属性可启用简单跟踪。缺省值是 false。
setTransactionIsolationLevel（String）	″none″，″read committed″，″read uncommitted″，″repeatable read″，″serializable″	此属性允许指定事务隔离级别。由于 JDBC 在缺省情况下具有自动提交方式，所以此属性的缺省值是 ″none″。

Set 方法（数据类型）	值	描述
setTranslateBinary （boolean）	″true″, ″false″	此属性可用来强制 JDBC 驱动程序将二进制和 varbinary 数据值视为 char 和 varchar 数据值。 此属性的缺省值是 false。
setUseBlockInsert （boolean）	″true″, ″false″	此属性允许本机 JDBC 驱动程序进入块插入方式来将数据块插入数据库。这是批处理更新的优化版本。这种优化方式只能在确保不会违反特定系统约束或发生数据插入故障和潜在地损坏数据的应用程序中使用。 打开此属性的应用程序只应该在尝试执行批处理更新时才连接至本地系统。由于不能通过 DRDA 来管理分块插入，所以这些应用程序不应使用 DRDA 来建立远程连接。 应用程序还必须确保带有 SQL insert 语句和 values 子句的 PreparedStatement 包含所有 insert 和 values 参数。values 列表中不允许常量。这是系统的分块插入引擎的一项要求。 缺省值是 false。
setUser （String）	任何内容	此属性允许设置用于获取连接的用户标识。此属性要求您同时设置 password 属性。

其它 DataSource 实现:  本机 JDBC 驱动程序附带包括的 DataSource 接口有两个实现。已不赞成使用这些 DataSource 实现。尽管您仍可以使用这些实现，但将来不会对它们进行改进；例如，未将健壮连接和语句合用添加至这些实现。这些实现将存在到您采用 UDBDataSource 接口及其相关函数为止。

DB2DataSource: DB2DataSource 是 DataSource 接口的早期实现，它不符合完整的规范（即，它比规范更早出现）。目前，DB2DataSource 的存在目的只是为了允许 WebSphere(R) 用户迁移至当前发行版，否则不应使用。

DB2StdDataSource: DB2StdDataSource 是 DB2DataSource 实现的修订版本，在 JDBC 可选包规范成为最终规范之后，DB2StdDataSource 就变为与规范相符。提供新版本的目的是不破坏已在 DB2DataSource 版本上编写的代码。

由于支持所有的旧属性，所以即使您编写了使用这些 DataSource 实现的应用程序，迁移至 UDBDataSource 也只是一项很普通的任务。建议您迁移至 UDBDataSource 以获取新的 UDBDataSource 类的功能。

IBM Developer Kit for Java 的 DatabaseMetaData 接口

IBM Developer Kit for Java(TM) JDBC 驱动程序实现 DatabaseMetaData 接口的目的是为了提供关于它的下层数据源的信息。此接口主要由应用程序服务器和工具用来确定如何与给定的数据源交互。应用程序还可使用 DatabaseMetaData 方法来获取关于某个数据源的信息，但这种用法没有那么强的代表性。

DatabaseMetaData 接口包含超过 150 个方法，可根据这些方法所提供的下列类型的信息来对它们进行分类:

- 关于数据源的一般信息
- 对给定功能的数据源支持
- 数据源限制
- SQL 对象及它们的属性
- 数据源提供的事务支持

DatabaseMetaData 接口还包含超过 40 个字段，它们是用作各个 DatabaseMetaData 方法的返回值的常量。

有关对 DatabaseMetaData 接口中的方法所作的更改的信息，参见 JDBC 3.0 中的更改。

创建 DatabaseMetaData 对象: DatabaseMetaData 对象是使用 Connection 方法 getMetaData 创建的。在创建该对象之后，就可使用它来动态地查找关于下层数据源的信息。以下示例创建 DatabaseMetaData 对象并使用它来确定表名所允许的最大字符数:

示例: 创建 DatabaseMetaData 对象

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

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

检索一般信息: 一些 DatabaseMetaData 方法用来动态地查找关于数据源的一般信息以及获取关于它的实现的详细信息。这些方法中的其中一些包括:

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

确定功能支持: 很大一组 DatabaseMetaData 方法可用来确定驱动程序或下层数据源是否支持给定的功能或功能集。除此之外，还有些方法描述了所提供的支持的级别。一些用于描述对个别功能的支持的方法包括:

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

用于描述功能支持级别的方法包括:

- supportsANSI92EntryLevelSQL
- supportsCoreSQLGrammar

数据源限制: 另一组方法用于提供给定数据源施加的限制。此类别中的一些方法包括:

- getMaxRowSize

- getMaxStatementLength
- getMaxTablesInSelect
- getMaxConnections
- getMaxCharLiteralLength
- getMaxColumnsInTable

这个组中的方法将限制值作为整数返回。如果返回值为零，则表示没有限制，或者限制是未知的。

SQL 对象及它们的属性: 许多 DatabaseMetaData 方法提供了关于植入给定数据源的 SQL 对象的信息。这些方法可确定 SQL 对象的属性。这些方法还返回 ResultSet 对象，在这些对象中，每一行都描述特定的对象。例如，getUDTs 方法返回一个 ResultSet 对象，在此对象中，对于数据源中已定义的每个用户定义表（UDT）都有一行。此类别的示例包括:

- getSchemas 和 getCatalogs
- getTables
- getPrimaryKeys
- getProcedures 和 getProcedureColumns
- getUDTs

事务支持: 一小组方法用于提供关于数据源支持的事务语义的信息。此类别的示例包括:

- supportsMultipleTransactions
- getDefaultTransactionIsolation

有关如何使用 DatabaseMetaData 接口的示例，参见示例: IBM Developer Kit for Java的 DatabaseMetaData 接口。

JDBC 3.0 中的更改: » 在 JDBC 3.0 中，对一些方法的返回值作了更改。在 JDBC 3.0 中，更新了下列方法，对它们返回的 ResultSet 添加了字段。

- getTables
- getColumns
- getUDTs
- getSchemas

注意: 如果正在使用 Java Development Kit（JDK）1.4 来开发应用程序，您可能会认识到在测试时会返回特定数目个列。您编写应用程序并期望访问所有这些列。然而，如果还将应用程序设计为在前发行版的 JDK 上运行，则当该应用程序尝试访问在较早的 JDK 发行版中不存在的这些字段时将接收到 SQLException。SafeGetUDTs 就是一个可以将应用程序编写为能够与 JDK 1.4、JDK 1.3 以及先前 JDK 发行版配合工作的示例。

«

示例: IBM Developer Kit for Java 的 DatabaseMetaData 接口: 此示例显示如何返回表列表。

示例 1: 返回表列表

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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():
```

有关更多信息，参见 IBM Developer Kit for Java(TM) 的 DatabaseMetaData 接口。

示例: 使用带有多个列的元数据 ResultSet: 这是有关如何使用带有多个列的元数据 ResultSet 的示例。

示例: 使用带有多个列的元数据 ResultSet

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////////////////////////////////////////////
//
// 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.
             }
```

```
            }
        }
    }
}
```

异常

Java(TM) 语言使用异常来为它的程序提供错误处理能力。异常是在运行程序时发生的事件，此事件会打断指令的正常流向。

Java 运行时系统和 Java 包中的许多类在某些情况下使用 throw 语句来抛出异常。可以在 Java 程序中使用同一机制来抛出异常。

要了解更多关于异常的信息，参见下列各节:

SQLException

SQLException 类及其子类型提供关于访问数据源时发生的错误和警告的信息。

SQLWarning

如果方法导致了数据库访问警告，则那些方法将生成 SQLWarning 对象。下列接口中的方法可生成 SQLWarning:

- Connection
- Statement 及其子类型、PreparedStatement 和 CallableStatement
- ResultSet

DataTruncation

DataTruncation 是 SQLWarning 的子类。在不抛出 SQLWarning 时，有时会抛出和连接 DataTruncation 对象，就象抛出和连接其它 SQLWarning 对象一样。

安静截断

setMaxFieldSize 语句方法允许对任何列指定最大字段大小。即使数据因为大小超过最大字段大小值而导致截断，也不报告警告或异常。

SQLException: SQLException 类及其子类型提供关于访问数据源时发生的错误和警告的信息。

与大部分 JDBC（由接口定义）不同，异常支持是在类中提供的。运行 JDBC 应用程序时发生的异常的基本类是 SQLException。JDBC API 的每个方法都声明为能够抛出 SQLException。SQLException 是 java.lang.Exception 的扩展，它提供了与数据库上下文中发生的故障相关的附加信息。明确地说，SQLException 提供了下列信息:

- 文本描述
- SQLState
- 错误代码
- 对同时发生的任何其它异常的引用

ExceptionExample 是一个能够正确处理 SQLException 的捕获（在此情况下期望发生这种情况）并转储它所提供的所有信息的程序。

注意: JDBC 提供了可以将异常链接到一起的机制。这允许驱动程序或数据库在单一请求上报告多个错误。当前，本机 JDBC 驱动程序在任何情况下都不会这样做。然而，提供此信息的目的仅是作为参考，而并非清晰地指示驱动程序将来永远都不会这样做。

如上所述，出错时将抛出 SQLException 对象。这是正确的，但并不是完整的描述。在实践中，本机 JDBC 驱动程序很少抛出实际的 SQLException。它抛出它自己的 SQLException 子类的实例。如下所示，这使您能够确定关于实际失败的内容的更多信息。

DB2Exception.java: 任何一个 DB2Exception 对象都不是直接抛出的。这个基本类用于存放所有 JDBC 异常的公共功能。这个类有两个子类，它们是 JDBC 抛出的标准异常。这两个子类是 DB2DBException.java 和 DB2JDBCException.java。DB2DBException 是报告给您的直接来自数据库的异常。当 JDBC 驱动程序发现关于它自己的问题时，便抛出 DB2JDBCException。以此方式分割异常类层次结构允许您以不同的方式处理两种类型的异常。

DB2DBException.java: 如前所述，DB2DBException 是直接来自数据库的异常。当 JDBC 驱动程序调用 CLI 并取回 SQLERROR 返回码时，将遇到这些异常。在这些情况下，调用 CLI 函数 SQLError 来获取消息文本、SQLState 和供应商代码。还检索 SQLMessage 的替换文本并将其返回给您。DatabaseException 类导致一个错误，数据库识别该错误并将其报告给 JDBC 驱动程序，以便为其构建异常对象。

DB2JDBCException.java: 为来自 JDBC 驱动程序本身的错误条件生成 DB2JDBCException。此异常类的功能具有根本上的不同；JDBC 驱动程序本身负责处理异常以及操作系统和数据库对源于数据库的异常处理的其它问题的消息语言翻译。每当有可能时，JDBC 驱动程序遵守数据库的 SQLState。JDBC 驱动程序抛出的异常的供应商代码总是 -99999。CLI 层识别并返回的 DB2DBException 通常也具有 -99999 错误代码。JDBCException 类导致一个错误，JDBC 驱动程序识别该错误并为它自己构建异常。在本发行版的开发期间运行时，创建了以下输出。注意，堆栈的顶部包含 DB2JDBCException。这指示在对数据库进行任何请求之前 JDBC 驱动程序报告了错误。 «

示例: SQLException: » 这是对 SQLException 进行高速缓存并对它提供的信息进行转储的示例。

示例: SQLException

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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: » 如果下列接口中的方法导致了数据库访问警告，则那些方法将生成 SQLWarning 对象:

- Connection
- Statement 及其子类型、PreparedStatement 和 CallableStatement
- ResultSet

当方法生成 SQLWarning 对象时，并不会通知调用者已发生数据访问警告。必须对适当的对象调用 getWarnings 方法才能检索到 SQLWarning 对象。然而，在某些情况下可能会抛出 SQLWarning 的 DataTruncation 子类。应该注意的是，本机 JDBC 驱动程序可选择忽略一些由数据库生成的警告以提高效率。例如，当您尝试通过 ResultSet.next 方法检索位于 ResultSet 末尾之后的数据时，系统将生成警告。在此情况下，next 方法定义为返回 false 而不是 true，以通知您发生了错误。由于没有必要创建对象来重申这一点，因此仅仅是忽略该警告。

如果发生多个数据访问警告，则它们链接至第一个警告，并可通过调用 SQLWarning.getNextWarning 方法来检索这些警告。如果链中没有更多的警告，则 getNextWarning 返回 null。

将把后续的 SQLWarning 对象继续添加到链中，直到处理下一个语句为止，或者，对于 ResultSet 对象这种情况，直到重新定位游标时为止。结果是，将除去链中的所有 SQLWarning 对象。

使用 Connection、Statement 和 ResultSet 对象可能会导致生成 SQLWarning。SQLWarning 是信息性消息，指示尽管特定操作已成功完成，但可能有您应当了解的其它信息。SQLWarning 是 SQLException 类的扩展，但并不抛出它们。而是，它们连接至导致生成它们的对象。在生成 SQLWarning 时，并不会发生任何情况来通知应用程序已生成警告。应用程序必须主动请求警告信息。

与 SQLException 相似，SQLWarning 可以相互链接。可以对 Connection、Statement 或 ResultSet 对象调用 clearWarnings 方法来清除该对象的警告。

注意: 调用 clearWarnings 方法并不会清除所有警告。它只清除与特定对象相关联的警告。

如果您不以手工方式清除 SQLWarning 对象，则 JDBC 驱动程序将在特定的时间清除它们。执行下列操作时将清除 SQLWarning 对象:

- 对于 Connection 接口，在创建新的 Statement、PreparedStatement 或 CallableStatement 对象时清除警告。
- 对于 Statement 接口，在处理下一个语句时（或者，对于 PreparedStatement 和 CallableStatement，在再次处理语句时）清除警告。
- 对于 ResultSet 接口，在重新定位游标时清除警告。 «

DataTruncation: DataTruncation 是 SQLWarning 的子类。在不抛出 SQLWarning 时，有时会抛出和连接 DataTruncation 对象，就象抛出和连接其它 SQLWarning 对象一样。DataTruncation 对象提供了 SQLWarning 所没有返回的附加信息。可用的信息包括:

- 应该已传送的数据的字节数。
- 截断的列或参数索引。
- 该索引是用于参数还是用于 ResultSet 列。
- 截断是发生在读取数据库时还是发生在写数据库时。
- 实际传送的数据量。

在一些情况下，此信息简明易懂，但引起该信息的原因并非一目了然。例如，如果使用 PreparedStatement 的 setFloat 方法来将值插入到存放整数值的列中，则可能会因为浮点大于该列可以存放的最大值而导致 DataTruncation。在这些情况下，截断的字节计数没有意义，但驱动程序提供截断信息十分重要。

报告 set() 和 update() 方法: JDBC 驱动程序之间存在着微妙的差异。一些驱动程序（如本机和 Toolbox JDBC 驱动程序）在进行参数设置时捕获和报告数据截断问题。这是在 PreparedStatement set 方法或 ResultSet update 方法上完成的。其它驱动程序在处理语句时报告问题，并且通过 execute、executeQuery 或 updateRow 方法完成。

在您提供不正确的数据时（而不在处理不能进一步继续进行时）报告问题失败有两个优点:

- 当有问题时，可以在应用程序中解决故障，而不是在处理时解决问题。
- 通过在设置参数时进行检查，JDBC 驱动程序可确保在处理语句时交给数据库的值有效。这允许数据库优化它的工作，并且处理可以更快地完成。

ResultSet.update() 方法抛出 DataTruncation 异常: 在过去的一些发行版中，当存在截断条件时，ResultSet.update() 方法发布警告。当要将数据值插入数据库时，便会发生这种情况。规范指示 JDBC 驱动程序在这些情况下应抛出异常。因此，JDBC 驱动程序以此方式工作。

在处理接收数据截断错误的 ResultSet update 函数与处理为用于接收错误的更新或插入语句设置的准备语句参数之间不应存在显著的差异。在这两种情况下，问题是完全相同的；您提供了与使用位置不相符的数据。

小数点右边的 NUMERIC 和 DECIMAL 截断是安静地进行的。这就是 UDB NT 的 JDBC 的工作方式和 iSeries 服务器上的交互式 SQL 的工作方式。

注意: 发生数据截断时，不将任何值舍入。在 NUMERIC 或 DECIMAL 列中放不下的任何参数的分数部分都将丢失，而没有警告。

下面是一些示例，这些示例假定 values 子句中的值实际上是准备语句上设置的参数:

```
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.
```

数据截断警告与数据截断异常之间的差异

规范指出对将要写至数据库的值发生的数据截断应抛出异常。如果没有对正在写至数据库的值执行数据截断，则应生成警告。这表示在标识数据截断情况的那一点，您还必须了解数据截断正在处理的语句类型。将这一点作为要求，下面列示了数种 SQL 语句类型的行为:

- 在 SELECT 语句中，查询参数从不会损坏数据库内容。因此，始终通过发布警告来处理数据截断情况。
- 在 VALUES INTO 和 SET 语句中，输入值仅用来生成输出值。因此，发出警告。

- 在 CALL 语句中，JDBC 驱动程序不能确定存储过程如何使用参数。当存储过程参数发生截断时，始终抛出异常。
- 所有其它语句类型都抛出异常，而不是发布警告。

Connection 和 DataSource 的数据截断属性: 已经有了一个可供许多发行版使用的数据截断属性。该属性的缺省值为 true，这表示检查数据截断问题，并发布警告或抛出异常。提供此属性的目的是当您不关心值在数据库列中是否放得下时为您提供方便并提高性能。您想让驱动程序将尽可能多的值放到列中。

数据截断属性只影响基于字符和二进制的数据类型: 在几个发行版以前，数据截断属性确定是否可抛出数据截断异常。设立数据截断属性的目的是当截断对于 JDBC 应用程序而言不重要时使它们不必担心值被截断。当应用程序尝试将 100 插入 DECIMAL(2,0) 时您想要在数据库中存储 00 或 10 值的情况是非常少的。因此，JDBC 驱动程序的数据截断属性已更改为只考虑参数用于基于字符的类型（如 CHAR、VARCHAR、CHAR FOR BIT DATA 和 VARCHAR FOR BIT DATA）的情况。

数据截断属性只适用于参数: 数据截断属性是 JDBC 驱动程序的设置而不是数据库的设置。因此，它不影响语句文字。例如，即使将数据截断标志设置为 false，所处理的下列用于将值插入数据库中的 CHAR(8) 列的语句也仍将失败（假定连接是在别处分配的 java.sql.Connection 对象）。

```
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.
```

本机 JDBC 驱动程序对数据不足够截断抛出异常: 本机 JDBC 驱动程序并不查看您为参数提供的数据。这样做只会减慢处理速度。然而，在某些情况下是否将值截断并不要紧，但您尚未将数据截断连接属性设置为 false。

例如，即使所有关于值的重要内容都能够装下，但 “dettinger”（这是所传送的 char(10)）也会抛出异常。这刚好是 UDB NT 的 JDBC 的工作方式；然而，这并不是当您在 SQL 语句中将值作为文字传送时所得到的行为。在这种情况下，数据库引擎将安静地扔掉附加的空间。

与 JDBC 驱动程序不抛出异常相关的问题包括:

- 无论需要与否，在每个 set 方法上都有大量的性能开销。对于大多数将不会有什么益处的情况，在象 setString() 一样普通的函数上存在可观的性能开销。
- 变通方法很普通，例如，对传入的字符串值调用 trim 函数。
- 存在与数据库列相关的问题需要考虑。CCSID 37 中的空格根本就不是 CCSID 65535 或 13488 中的空格。

安静截断: setMaxFieldSize 语句方法允许对任何列指定最大字段大小。即使数据因为大小超过最大字段大小值而导致截断，也不报告警告或异常。此方法与前面提到的数据截断属性一样，只影响基于字符的类型，如 CHAR、VARCHAR、CHAR FOR BIT DATA 和 VARCHAR FOR BIT DATA。 «

事务

» **事务**是逻辑工作单元。要完成逻辑工作单元，必须对数据库执行若干项操作。事务支持允许应用程序确保下列各项:

- 执行所有用于完成逻辑工作单元的步骤。
- 当工作单元的其中一个步骤失败时，可以撤销作为该逻辑工作单元的一部分完成的所有工作，数据库可以返回到事务开始之前它所处的状态。

事务用来在并行访问期间提供数据完整性、正确的应用程序语义以及数据的一致性视图。所有与 “Java(TM) 数据库连接”（JDBC）相符的驱动程序都必须支持事务。

注意: 本节只讨论本地事务和事务的标准 JDBC 概念。Java 和本机 JDBC 驱动程序支持“Java 事务 API”（JTA）、分布式事务和两阶段提交协议（2PC）。

所有事务工作都是在 Connection 对象级别处理的。当事务的工作完成时，可通过调用 commit 方法来将其最终化。如果应用程序使事务异常终止，则调用 rollback 方法。

位于某个连接之下的所有 Statement 对象都是事务的一部分。这表示如果应用程序创建了三个 Statement 对象并使用每一个对象来更改数据库，则当发生 commit 或 rollback 调用时，全部三个语句的工作都将成为永久的或都将被废弃。

当单纯使用 SQL 时，使用 commit 和 rollback SQL 语句来使事务最终化。不能动态地准备这些 SQL 语句，不应尝试在 JDBC 应用程序中使用它们来完成事务。

要在应用程序中使用事务，请查看下列各项:

自动提交方式
JDBC 使用自动提交方式，在此方式下，对数据库所作的每一更新都将立即具有永久性。

事务隔离级别
事务隔离级别指定哪些数据对事务中的语句可视并且直接影响并行访问级别。

保存点
保存点是检查点，应用程序可以回滚至保存点而不会丢弃整个事务。请查找下列关于“保存点”的信息:

- 设置和回滚至保存点
- 释放保存点

自动提交方式:  缺省情况下，JDBC 使用名为“自动提交”的操作方式。这表示对数据库所作的每一更新都将立即具有永久性。在自动提交方式下，任何其逻辑工作单元要求对数据库进行多次更新的情况都不能安全地完成。当以自动提交方式运行时，在进行一项更新之后并且在进行任何其它更新之前，如果应用程序或系统发生了任何情况，则不能撤销第一项更改。

由于在自动提交方式下将立即使更改具有永久性，因此应用程序不需要调用 commit 方法或 rollback 方法。这简化了应用程序的编写工作。

在连接的存在期间，可动态地启用和禁用自动提交方式。假定数据源已存在，以下列方式启用自动提交:

```
Connection connection = dataSource.getConnection();

Connection.setAutoCommit(false);        // Disables auto-commit.
```

如果在某个事务的中间更改自动提交设置，则将自动提交任何暂挂的工作。如果对作为分布式事务一部分的连接启用自动提交，则将生成 SQLException。 

事务隔离级别:  事务隔离级别指定哪些数据对事务中的语句可视。通过定义对同一目标数据源执行的事务之间的可能交互作用，这些级别直接影响并行访问级别。

数据库反常: 数据库反常是生成的结果，这些结果从单一事务的作用域中看上去是不正确的，但从所有事务的作用域中看上去却是正确的。下面描述了不同类型的数据库反常:

- 在以下情况下，发生**脏**读取:

1. 事务 A 将行插入到表中。
2. 事务 B 读取这个新行。
3. 事务 A 回滚。

事务 B 可能已根据事务 A 插入的行对系统执行了工作，但该行永远不会成为数据库的永久部分。

- 在以下情况下，发生**不可重复**读取:
  1. 事务 A 读取行。
  2. 事务 B 更改该行。
  3. 事务 A 再次读取同一行并获取新结果。
- 在以下情况下，发生**幻象**读取:
  1. 事务 A 读取所有满足 SQL 查询上的 WHERE 子句的行。
  2. 事务 B 插入附加的满足 WHERE 子句的行。
  3. 事务 A 对 WHERE 条件进行重新求值并获得附加的行。

注意: 由于 DB2/400 具有锁定策略，所以它不会总是使应用程序暴露在规定级别的可允许数据库反常之下。

JDBC 事务隔离级别: 在 IBM Developer Kit for Java JDBC API 中共有五个级别的事务隔离。它们按照限制性从低到高的次序列示如下:

JDBC_TRANSACTION_NONE
这是一个特殊的常量，指示 JDBC 驱动程序不支持事务。

JDBC_TRANSACTION_READ_UNCOMMITTED
此级别允许事务查看对数据所作的未提交更改。在此级别，所有数据库反常都是有可能的。

JDBC_TRANSACTION_READ_COMMITTED
此级别表示在提交事务之前，在该事务中所作的任何更改在该事务之外都不可视。这杜绝了脏读取的可能性。

JDBC_TRANSACTION_REPEATABLE_READ
此级别表示保持将读取的行锁定，从而使另一事务在此事务完成之前不能更改这些行。这将禁止脏读取和不可重复读取。幻象读取仍是有可能的。

JDBC_TRANSACTION_SERIALIZABLE
在事务期间将表锁定，从而使其它对表添加值或除去值的事务不能更改 WHERE 条件。这将杜绝所有类型的数据库反常。

可使用 setTransactionIsolation 方法来更改连接的事务隔离级别。

注意事项: 有一种常见的误解，即认为 JDBC 规范定义了前面提到的五个事务级别。通常认为 TRANSACTION_NONE 值表示在不存在提交控制的情况下运行这一概念。然而，JDBC 规范并没有以同一方式定义 TRANSACTION_NONE。在 JDBC 规范中，将 TRANSACTION_NONE 定义成这样的一个级别: 在这个级别，驱动程序不支持事务并且不是与 JDBC 相符的驱动程序。在调用 getTransactionIsolation 方法时，从来不会报告 NONE 级别。

由于 JDBC 驱动程序的缺省事务隔离级别是由实现定义的，所以问题稍微有点复杂。本机 JDBC 驱动程序缺省事务隔离级别的缺省事务隔离级别是 NONE。这允许驱动程序使用没有日志的文件，并且您不需要作任何指定（如指定 QGPL 库中的文件）。

本机 JDBC 驱动程序允许将 JDBC_TRANSACTION_NONE 传送给 setTransactionIsolation 方法或指定 none 作为连接属性。然而，当值为 none 时，getTransactionIsolation 方法总是报告 JDBC_TRANSACTION_READ_UNCOMMITTED。跟踪正在运行的级别是应用程序的职责（如果应用程序有此需要的话）。

在过去的发行版中，由于系统没有真正自动提交方式这一概念，所以，如果对自动提交指定 true，则 JDBC 驱动程序将通过把事务隔离级别更改为 none 来进行处理。这在功能上极为相似，但并非在所有方案下都能提供正确的结果。现在情况已有所不同；数据库已将自动提交的概念与事务隔离级别的概念分开。因此，在启用自动提交的情况下在 JDBC_TRANSACTION_SERIALIZABLE 级别运行是完全有效的。唯一无效的方案是在 JDBC_TRANSACTION_NONE 级别运行并且不处于自动提交方式。当系统不是在具有事务隔离级别的情况下运行时，应用程序不能控制提交边界。

JDBC 规范与 iSeries 平台之间的事务隔离级别: iSeries 平台的事务隔离级别具有公共的名称，这些名称与 JDBC 规范提供的那些名称不匹配。下表对 iSeries 平台使用的那些与 JDBC 规范所使用的名称不匹配的名称作了匹配:

JDBC 级别 *	**iSeries 级别**
JDBC_TRANSACTION_NONE	*NONE 或 *NC
JDBC_TRANSACTION_READ_UNCOMMITTED	*CHG 或 *UR
JDBC_TRANSACTION_READ_COMMITTED	*CS
JDBC_TRANSACTION_REPEATABLE_READ	*ALL 或 *RS
JDBC_TRANSACTION_SERIALIZABLE	*RR

* 在这个表中，JDBC_TRANSACTION_NONE 值与 iSeries 级别 *NONE 和 *NC 排在一起是为了使您看得更清楚。这并不是直接的从规范到 iSeries 级别的匹配。 «

保存点: » 保存点允许在事务中设置 “分段点”。保存点是检查点，应用程序可以回滚至保存点而不会丢弃整个事务。保存点是 JDBC 3.0 中新增的，这表示应用程序必须在 Java(TM) Development Kit（JDK）1.4 上运行才能使用它们。此外，Developer Kit for Java 不识别保存点，这表示如果不将 JDK 1.4 与前发行版的 Developer Kit for Java 配合使用，则不支持保存点。

注意: 系统提供了用于使用保存点的 SQL 语句。建议 JDBC 应用程序不要在应用程序中直接使用这些语句。这样做可能可以起作用，但这样做时 JDBC 驱动程序将丧失它的跟踪保存点的能力。至少应避免将两种模式（即，使用您自己的保存点 SQL 语句和使用 JDBC API）混合使用。

设置和回滚至保存点: 在事务的整个工作过程中都可以设置保存点。以后，如果发生任何错误，则应用程序可回滚至这些保存点中的任何一个并从该点继续处理。在以下示例中，应用程序将值 FIRST 插入数据库表。之后，设置保存点并将另一个值 SECOND 插入到数据库中。接着发出返回该保存点的回滚操作并撤销插入 SECOND 的工作，但保留 FIRST 作为暂挂事务的一部分。最后，插入值 THIRD 并提交事务。数据库表包含值 FIRST 和 THIRD。

示例: 设置和回滚至保存点

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
```

尽管在自动提交方式下设置保存点不大可能会导致问题，但由于事务结束时保存点的生命也将结束，所以不能回滚至保存点。

释放保存点: 应用程序可通过 Connection 对象上的 releaseSavepoint 方法来释放保存点。在释放保存点之后，尝试回滚至该保存点将导致异常。在提交或回滚事务时，将自动释放所有保存点。在回滚某个保存点时，还将释放位于它之后的其它保存点。

分布式事务

“Java(TM) 数据库连接”（JDBC）中的事务通常是本地的。这表示单一连接将执行事务的所有工作并且连接每次只能处理一个事务。当该事务的所有工作都已完成或已失败之后，调用提交或回滚来使工作永久化，然后就可以开始新事务。

Java 中有一项高级事务支持可用，此项支持提供了超出本地事务的功能性。此项支持完全是由 “Java 事务 API”（JTA）1.0.1 规范  指定的。

“Java 事务 API”（JTA）支持复杂的事务。它还支持将事务与 Connection 对象分开。由于 JDBC 是按照 “对象数据库连接”（ODBC）和 X/Open 的 “调用层接口”（CLI）规范建模的，所以 JTA 是按照 X/Open 的 “扩展体系结构”（XA）规范建模的。JTA 与 JDBC 配合工作以将事务与 Connection 对象分开。通过将事务与 Connection 对象分开，这允许单一连接并行地处理多个事务。反过来，也允许多个 Connection 处理单一事务。

注意: 如果您计划使用 JTA，则请参考 JDBC 入门以了解关于扩展类路径中的必需 “Java 压缩文档”（JAR）文件的更多信息。您同时需要 JDBC 2.0 可选包和 JTA JAR 文件（如果运行的是 JDK 1.4，则 JDK 能够自动找到这些文件）。缺省情况下不查找这些文件。

与 JTA 的事务: 当 JTA 与 JDBC 配合使用时，它们之间存在一系列用于完成事务工作的步骤。通过 XADataSource 类提供了对 XA 的支持。这个类支持以与它的 ConnectionPoolDataSource 超类完全相同的方式设置连接池。

借助 XADataSource 实例，可以检索 XAConnection 对象。XAConnection 对象同时用作 JDBC Connection 对象和 XAResource 对象的容器。XAResource 对象设计成可处理 XA 事务支持。XAResource 通过名为事务标识（XID）的对象来处理事务。

XID 是一个必须实现的接口。它表示 X/Open 事务标识符的 XID 结构的 Java 映射。此对象包含三个部件:

- 全局事务的格式标识
- 全局事务标识
- 分支限定符

请查看 JTA 规范以了解有关此接口的完整详细信息。

示例: 使用 JTA 来处理事务显示了如何使用 JTA 来在应用程序中处理事务。

将 UDBXADataSource 支持用于合用和分布式事务: “Java 事务 API”支持提供了对连接池的直接支持。UDBXADataSource 是 ConnectionPoolDataSource 的扩展，它允许应用程序访问合用的 XAConnection 对象。由于 UDBXADataSource 是 ConnectionPoolDataSource，所以 UDBXADataSource 的配置和用法与将 DataSource 支持用于对象合用中描述的相同。

XADataSource 属性: 除了 ConnectionPoolDataSource 提供的属性之外，XADataSource 接口提供了下列属性:

设置方法 （数据类型）	值	描述
setLockTimeout （int）	0 或任何正数值	任何正数值都是事务级别的有效锁超时（以秒计）。 锁超时值为 0 表示尽管可能在其它级别（作业或表）实施了锁超时，但没有在事务级别实施锁超时。 缺省值为 0。
setTransactionTimeout （int）	0 或任何正数值	任何正数值都是有效的事务超时（以秒计）。 事务超时为 0 表示没有实施事务超时值。如果事务的活动时间长于超时值，则将其标记为仅回滚，后续的在其之下执行工作的尝试将导致发生异常。 缺省值为 0。

ResultSet 和事务: 除了象前一示例所示那样区分事务的开始和结束之外，还可以将事务暂挂一段时间并在以后继续。这为在事务期间创建的 ResultSet 资源提供了许多方案。

简单事务结束: 结束事务时，将自动关闭所有在该事务之下创建的处于打开状态的 ResultSet。建议在完成使用 ResultSet 时显式地将它们关闭以确保最大的并行处理度。然而，如果在进行 XAResource.end 调用之后访问任何在事务期间打开的 ResultSet，则会导致异常。

请查看显示了此行为的示例: 结束事务。

暂挂和继续: 当事务处于暂挂状态时，不允许访问当该事务处于活动状态时创建的 ResultSet，并且此访问将导致异常。然而，在继续事务之后，ResultSet 将再次可用，并保持处于暂挂事务之前它所处的状态。

请查看显示了此行为的示例: 暂挂和继续事务。

所生成的暂挂的 ResultSet: 当事务处于暂挂状态时，不能访问 ResultSet。然而，可以在另一个事务之下重新处理 Statement 对象以执行工作。由于 JDBC Statement 对象每次只能有一个 ResultSet（不包括 JDBC 3.0 对同一存储过程调用中可以有多个并行 ResultSet 的支持），所以必须关闭已暂挂的事务的 ResultSet 才能满足新事务的请求。这就是所发生的情况。

请查看显示了此行为的示例: 暂挂的 ResultSet。

注意: 尽管 JDBC 3.0 允许 Statement 对一个存储过程调用同时打开多个 ResultSet，但将它们视为一个单元，如果在新事务之下重新处理该 Statement，则将把那些 ResultSet 全部关闭。目前，不可能让来自两个事务的 ResultSet 对单一语句同时处于活动状态。

多路复用: JTA API 设计成将事务与 JDBC 连接分离。此 API 允许您让多个连接并行地处理单一事务或单一连接并行地处理多个事务。这称为**多路复用**，这样就可以执行许多在单独使用 JDBC 的情况下不能完成的复杂任务。

此示例显示处理单一事务的多个连接。

此示例显示同时发生多个事务的单一连接。

有关使用 JTA 的进一步信息，参见 JTA 规范。JDBC 3.0 规范还包含有关这两种技术如何一起工作以支持分布式事务的信息。

两阶段提交和事务记录: JTA API 彻底将分布式两阶段提交协议的职责外部化到应用程序这一级。如示例所示，当使用 JTA 和 JDBC 来在 JTA 事务之下访问数据库时，应用程序使用 XAResource.prepare() 和 XAResource.commit() 方法或仅仅使用 XAResource.commit() 方法来提交更改。

另外，当使用单一事务来访问多个相异数据库时，应用程序负责确保执行跨那些数据库的事务原子所必需的两阶段提交协议以及任何相关联的记录。通常，在应用程序服务器或事务监控器的控制之下执行跨多个数据库（即 XAResource）的两阶段提交处理及其记录，从而使应用程序本身实际上并不关心这些问题。

例如，应用程序可调用一些 commit() 方法或者在不出错的情况下从它的处理返回。于是，下层应用程序服务器或事务监控器将开始进行每个参与单一分布式事务的数据库（XAResource）的处理。

应用程序服务器在两阶段提交处理期间将使用扩展记录。它将依次对每个参与数据库（XAResource）调用 XAResource.prepare() 方法，然后对每个参与数据库（XAResource）调用 XAResource.commit() 方法。

如果在此处理期间发生故障，则应用程序服务器的事务监控器记录允许应用程序服务器本身接着使用 JTA API 来恢复分布式事务。由应用程序服务器或事务监控器控制的此项恢复允许应用程序服务器使事务具有在每个参与数据库（XAResource）上已知的状态。这确保整个分布式事务跨所有参与数据库具有大家熟知的状态。

示例: 使用 JTA 来处理事务:  这是有关如何在应用程序中使用 “Java(TM) 事务 API” （JTA）来处理事务的示例。

示例: 使用 JTA 来处理事务

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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:  Cleaup exception.");
            e.printStackTrace();
            }
        }
    }
}
```

«

示例: 在一个事务中工作的多个连接: » 这是有关如何使用多个在单一事务中工作的连接的示例。

示例: 在一个事务中工作的多个连接

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 UDBXADataSource.
         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: Cleaup exception " +
                              e.getMessage());
        }
          try {
           if (c2 != null) {
              c2.close();
           }
        }
        catch (SQLException e) {
           System.out.println("Note: Cleaup exception " +
                              e.getMessage());
        }
          try {
           if (c3 != null) {
              c3.close();
           }
        }
        catch (SQLException e) {
           System.out.println("Note: Cleaup exception " +
                              e.getMessage());
        }
      }
   }
}
```

«

示例: 将一个连接与多个事务配合使用: » 这是有关如何将单一连接与多个事务配合使用的示例。

示例: 将一个连接与多个事务配合使用

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 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();
            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 = JDXATest.xidFactory();
            Xid xid2 = JDXATest.xidFactory();
            Xid xid3 = JDXATest.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:  Cleaup exception.");
            e.printStackTrace();
            }
       }
   }
}
```

«

示例: 暂挂的 ResultSet: » 这是有关如何在另一个事务下重新处理 Statement 对象以执行工作的示例。

示例: 暂挂的 ResultSet

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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:  Cleaup exception.");
            e.printStackTrace();
            }
        }
    }
}
```

«

示例: 结束事务: » 这是有关如何在应用程序中结束事务的示例。

示例: 结束事务

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 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, 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:  Cleaup exception.");
            e.printStackTrace();
            }
        }
    }
}
```

«

示例: 暂挂和继续事务: » 这是先暂挂事务然后继续该事务的示例。

示例: 暂挂和继续事务

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 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 nonXAStmt = conn.createStatement();
            ResultSet nonXARS = nonXAStmt.executeQuery("SELECT * FROM CUJOSQL.JTATABLE");
            while (nonXARS.next()) {
                // Process here...
            }
            nonXARS.close();
            nonXAStmt.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:  Cleaup exception.");
            e.printStackTrace();
            }
        }
    }
}
```

语句类型

 Statement 接口及其 PreparedStatement 和 CallableStatement 子类用来面向数据库处理结构化查询语言（SQL）命令。SQL 语句导致生成 ResultSet 对象。

可使用 Connection 接口上的许多方法来创建 Statement 接口的子类。可以同时在单一 Connection 对象之下创建许多个 Statement 对象。在过去的发行版中，有可能给出可以创建的 Statement 对象的精确数目。在本发行版中，由于不同类型的 Statement 对象在数据库引擎中具有不同数目个“句柄”，因此已不可能这样做。因此，所使用的 Statement 对象的类型会对某一时刻在某个连接之下可以活动的语句数产生影响。

应用程序调用 Statement.close 方法来指示应用程序已完成对语句的处理。当关闭创建 Statement 对象的连接时，所有那些 Statement 对象也将关闭。然而，您不应完全依靠此行为来关闭 Statement 对象。例如，如果将应用程序更改为使用连接池而不是显式地关闭连接，则会因为从不关闭连接而导致应用程序“泄漏”语句句柄。通过在不再需要 Statement 对象时立即将它们关闭，可以立即释放语句所使用的外部数据库资源。

本机 JDBC 驱动程序会尝试检测语句泄漏并代替您处理它们。然而，依赖于该项支持将导致性能下降。

要使用语句及其子类，请查看下列各项:

Statement
Statement 对象用于处理静态 SQL 语句和获取它所生成的结果。

PreparedStatement
PreparedStatement 是 Statement 接口的子类，它提供了对向 SQL 语句添加参数的支持。

CallableStatement
CallableStatement 扩展 PreparedStatement 接口，除了 PreparedStatement 提供的输入参数支持之外，它还提供了对输出和输入／输出参数的支持。

由于 CallableStatement、PreparedStatement 和 Statement 之间存在着继承层次结构关系（CallableStatement 扩展 PreparedStatement，而 PreparedStatement 扩展 Statement），所以每个接口的功能在扩展该接口的类中也可用。

例如，在 PreparedStatement 和 CallableStatement 类中也支持 Statement 类的功能。主要例外是 Statement 类上的 executeQuery、executeUpdate 和 execute 方法。这些方法接收 SQL 语句来进行动态处理，如果尝试将它们与 PreparedStatement 或 CallableStatement 对象配合使用，则会导致异常。

Statement:  Statement 对象用于处理静态 SQL 语句和获取它所生成的结果。每次只能为 Statement 对象打开一个 ResultSet。如果存在打开的 ResultSet，则所有处理 SQL 语句的语句方法都隐式地关闭语句的当前 ResultSet。

创建语句: Statement 对象是使用 createStatement 方法从 Connection 对象创建的。例如，假定已存在名为 conn 的 Connection 对象，以下代码行将创建一个 Statement 对象以用于将 SQL 语句传送至数据库:

```
Statement stmt = conn.createStatement();
```

指定 ResultSet 特征: ResultSet 的特征与最终创建它们的语句相关联。Connection.createStatement 方法允许您指定这些 ResultSet 特征。以下是有效 createStatement 方法调用的一些示例:

示例: createStatement 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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);
```

有关这些特征的更多信息，参见 ResultSet。

处理语句: 使用 Statement 对象处理 SQL 语句是通过 executeQuery()、executeUpdate() 和 execute() 方法完成的。

从 SQL 查询返回结果: 如果要处理返回 ResultSet 对象的 SQL 查询语句，则应使用 executeQuery() 方法。您可以参考一个示例程序，此程序使用 Statement 对象 executeQuery 方法来获取 ResultSet。

注意: 如果使用 executeQuery 处理的 SQL 语句不返回 ResultSet，则抛出 SQLException。

返回 SQL 语句的更新计数: 如果知道 SQL 是返回更新计数的“数据定义语言”（DDL）语句或“数据操纵语言”（DML）语句，则应使用 executeUpdate() 方法。StatementExample 程序使用 Statement 对象的 executeUpdate 方法。

处理预期返回为未知的 SQL 语句: 如果 SQL 语句类型是未知的，则应使用 execute 方法。在处理此方法之后，JDBC 驱动程序就可以通过 API 调用告诉应用程序 SQL 语句生成了什么类型的结果。如果结果是至少一个 ResultSet，则 execute 方法返回 true，如果返回值是更新计数，则返回 false。得到此信息之后，应用程序可使用语句方法的 getUpdateCount 或 getResultSet 来检索处理 SQL 语句所得到的返回值。StatementExecute 程序对 Statement 对象使用 execute 方法。此程序期望传送 SQL 语句作为参数。程序处理语句并确定关于所要处理的内容的信息，但不查看您提供的 SQL 的文本。

注意: 当结果是 ResultSet 时调用 getUpdateCount 方法将返回 -1。当结果是更新计数时调用 getResultSet 方法将返回空。

cancel 方法: 本机 JDBC 驱动程序的方法是同步的，这可以防止对同一对象运行的两个线程损坏该对象。cancel 方法是一个例外。一个线程可使用 cancel 方法来结束在同一对象的另一线程中长时间运行的 SQL 语句。本机 JDBC 驱动程序不能强制线程停止工作；它只能请求线程停止正在执行的任务。因此，要让被取消语句停止仍需要时间。cancel 方法可用来停止系统上的失控 SQL 查询。 «

示例: 使用 Statement 对象的 executeUpdate 方法: » 这是有关如何使用 Statement 对象的 executeUpdate 方法的示例。

示例: 使用 Statement 对象的 executeUpdate 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 扩展 Statement 接口，它提供了对向 SQL 语句添加参数的支持。

传送给数据库的 SQL 语句通过一个包含两个步骤的过程来返回结果。首先准备它们，然后处理它们。借助 Statement 对象，这两个阶段对应用程序而言变成一个阶段。PreparedStatement 允许将这两个步骤分开。准备步骤在创建对象时发生，而处理步骤在对 PreparedStatement 对象调用 executeQuery、executeUpdate 或 execute 方法时发生。

如果不添加参数标记，能够将 SQL 处理分割成单独的阶段并没有意义。参数标记放在应用程序中，从而使它能够告诉数据库它在准备时并不具有特定的值，但它在处理之前提供一个值。在 SQL 语句中，参数标记是使用问号表示的。

通过使用参数标记，有可能创建用于特定请求的一般 SQL 语句。例如，给定以下 SQL 查询语句:

```
SELECT * FROM EMPLOYEE_TABLE WHERE LASTNAME = 'DETTINGER'
```

这是一个特定的 SQL 语句，它只返回一个值；即关于名为 Dettinger 的雇员的信息。通过添加参数标记，可以使语句更为灵活:

```
SELECT * FROM EMPLOYEE_TABLE WHERE LASTNAME = ?
```

通过简单地将参数标记设置为某个值，可以获取关于表中的任何雇员的信息。

由于前一个 Statement 示例可以只经过一次准备阶段并接着使用不同的参数值来重复地进行处理，所以 PreparedStatement 能够提供比 Statement 更高的性能。

注意: 要支持本机 JDBC 驱动程序的语句合用，必须使用 PreparedStatement。

创建 PreparedStatement: prepareStatement 方法用来创建新的 PreparedStatement 对象。与 createStatement 方法不同，创建 PreparedStatement 对象时必须提供 SQL 语句。在那个时候，对 SQL 语句进行预编译以供使用。例如，假定已存在名为 conn 的 Connection 对象，以下示例将创建 PreparedStatement 对象并准备要在数据库中处理的 SQL 语句。

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM EMPLOYEE_TABLE
                                              WHERE LASTNAME = ?");
```

指定 ResultSet 特征和自动生成的键支持: 与 createStatement 方法相同，重载 prepareStatement 方法的目的是提供对指定 ResultSet 特征的支持。prepareStatement 方法还具有变体，可使用自动生成的键。以下是有效 prepareStatement 方法调用的一些示例:

示例: prepareStatement 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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);
```

处理参数: 在可以处理 PreparedStatement 对象之前，必须将每个参数标记设置为一些值。PreparedStatement 对象提供了许多个用于设置参数的方法。所有这些方法的格式均为 set<Type>，其中 <Type> 是 Java 数据类型。这些方法的一些示例包括 setInt、setLong、setString、setTimestamp、setNull 和 setBlob。几乎所有这些方法都有两个参数:

- 第一个参数是该参数在语句中的索引。参数标记具有从 1 开始的编号。
- 第二个参数是要对第一个参数设置的值。有几个 set<Type> 方法具有附加的参数，如 setBinaryStream 上的长度参数。

有关更多信息，请查阅 java.sql 包的 Javadoc。通过对 ps 给出在先前示例中准备的 SQL 语句，以下代码说明了如何在处理之前指定参数值:

```
ps.setString(1,'Dettinger');
```

如果尝试处理带有尚未设置的参数标记的 PreparedStatement，则将抛出 SQLException。

注意: 在设置参数标记之后，除非发生下列情况，否则参数标记将保持具有同一个值。

- 另一个 set 方法调用更改了该值。
- 调用 clearParameters 方法时除去了该值。

clearParameters 方法将所有参数都标记为尚未设置。在进行 clearParameters 调用之后，在执行下一个过程之前，必须再次对所有参数调用 set 方法。

ParameterMetaData 支持: 新的 ParameterMetaData 接口允许检索关于参数的信息。此支持与 ResultSetMetaData 相符并且类似。提供了全面的诸如精度、标度、数据类型、数据类型名以及该参数是否允许空值之类的信息。

参见示例: ParameterMetaData 以了解如何在应用程序中使用这项新支持。 «

处理 PreparedStatement: » 与处理 Statement 对象相似，使用 PreparedStatement 对象处理 SQL 语句是通过 executeQuery、executeUpdate 和 execute 方法完成的。与 Statement 版本不同，由于在创建对象时已提供了 SQL 语句，所以不在这些方法上传送参数。由于 PreparedStatement 扩展 Statement，所以应用程序可尝试调用 executeQuery、executeUpdate 和 execute 方法的接收 SQL 语句的版本。这样做将导致抛出 SQLException。

从 SQL 查询返回结果: 如果要处理返回 ResultSet 对象的 SQL 查询语句，则应使用 executeQuery 方法。PreparedStatementExample 程序使用 PreparedStatement 对象的 executeQuery 方法来获取 ResultSet。

注意: 如果使用 executeQuery 方法处理的 SQL 语句不返回 ResultSet，则抛出 SQLException。

返回 SQL 语句的更新计数: 如果知道 SQL 是返回更新计数的“数据定义语言”（DDL）语句或“数据操纵语言”（DML）语句，则应使用 executeUpdate 方法。PreparedStatementExample 样本程序使用 PreparedStatement 对象的 executeUpdate 方法。

处理预期返回为未知的 SQL 语句: 如果 SQL 语句类型是未知的，则应使用 execute 方法。在处理此方法之后，JDBC 驱动程序就可以通过 API 调用告诉应用程序 SQL 语句所生成的结果的类型。如果结果是至少一个 ResultSet，则 execute 方法返回 true，如果返回值是更新计数，则返回 false。得到此信息之后，应用程序可使用 getUpdateCount 或 getResultSet 语句方法来检索处理 SQL 语句所得到的返回值。

注意: 当结果是 ResultSet 时调用 getUpdateCount 方法将返回 -1。当结果是更新计数时调用 getResultSet 方法将返回空。 «

示例: 使用 PreparedStatement 来获取 ResultSet: » 这是有关使用 PreparedStatement 对象的 executeQuery 方法来获取 ResultSet 的示例。

示例: 使用 PreparedStatement 来获取 ResultSet

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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.");
            }

        }
    }
}
```

«

示例: ParameterMetaData: » 这是有关使用 ParameterMetaData 接口来检索关于参数的信息的示例。

示例: ParameterMetaData

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////////////////////////////////////////////
//
// 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));
        }
    }
}
```

CallableStatement:  CallableStatement 接口扩展 PreparedStatement，它提供了对输出和输入 / 输出参数的支持。CallableStatement 接口还具有对 PreparedStatement 接口提供的输入参数的支持。

CallableStatement 接口允许使用 SQL 语句来调用存储过程。存储过程是具有数据库接口的程序。这些程序拥有下列各项:

- 它们可以拥有输入和输出参数，或者既输入又输出的参数。
- 它们可以有返回值。

- 它们有能力返回多个 ResultSet。

在 JDBC 中，存储过程调用在概念上是对数据库的单一调用，但与存储过程相关联的程序可以拥有数以百计的数据库请求。存储过程程序还可以执行许多其它通常不是使用 SQL 语句执行的程序任务。

由于 CallableStatement 遵循将准备与处理阶段分开这种 PreparedStatement 模型，所以它们有可能进行最优重新使用（有关详细信息，参见 PreparedStatement）。由于存储过程的 SQL 语句绑定到程序中，所以可以将它们作为静态 SQL 处理，并且能够以该方式获得进一步的性能增益。一个以最优方式使用存储过程的示例是将许多数据库工作封装在单一可重新使用数据库调用中。只有这个调用将通过网络前往其它系统，但该请求可以在远程系统上完成许多工作。

创建 CallableStatement: 使用 prepareCall 方法来创建新的 CallableStatement 对象。对于 prepareStatement 方法，必须在创建 CallableStatement 对象时提供 SQL 语句。在那个时候，对 SQL 语句进行预编译。例如，假定已存在名为 conn 的 Connection 对象，以下程序语句将创建 CallableStatement 对象并完成用于获取已可以在数据库中进行处理的 SQL 语句的准备阶段:

```
PreparedStatement ps = conn.prepareStatement("? = CALL ADDEMPLOYEE(?, ?, ?");
```

ADDEMPLOYEE 存储过程对新雇员姓名、他的社会保障号以及他的经理的用户标识接收输入参数。根据此信息，可使用关于雇员的信息（如他的开始日期、分公司以及部门，等等）来更新多个公司数据库表。此外，存储过程是一个可以生成该雇员的标准用户标识和电子邮件地址的程序。存储过程还可以将带有初始用户名和密码的电子邮件发送给雇用经理；然后，雇用经理可以将该信息提供给雇员。

ADDEMPLOYEE 存储过程设置为带有返回值。返回码可以是成功代码，也可以是在发生故障时可由调用程序使用的故障代码。还可以将返回值定义成新雇员的公司标识号。最后，存储过程程序可以以内部方式处理查询并保持来自那些查询的 ResultSet 处于打开状态且可供调用程序使用。通过所返回的 ResultSet 来查询有关新雇员的所有信息并使其可供调用者使用是合理的。

下列各节对如何完成每一种这些类型的任务作了阐述。

指定 ResultSet 特征和自动生成的键支持: 对于 createStatement 和 prepareStatement，存在多个版本的 prepareCall，它们提供了对指定 ResultSet 特征的支持。与 prepareStatement 不同，prepareCall 方法未提供变体来使用来自 CallableStatement 的自动生成的键（JDBC 3.0 不支持此概念）。以下是一些有效 prepareCall 方法调用的示例:

示例: prepareCall 方法

```
// The following is new in JDBC 2.0

CallableStatement cs2 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE);

// New in JDBC 3.0

CallableStatement cs3 = conn.prepareCall("? = CALL ADDEMPLOYEE(?, ?, ?)",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATEABLE,
    ResultSet.HOLD_CURSOR_OVER_COMMIT);
```

处理参数: 如前所述，CallableStatement 对象可以接收三种类型的参数:

- **IN**
  处理 IN 参数的方式与 PreparedStatement 相同。使用所继承的 PreparedStatement 类的各种设置方法来设置参数。
- **OUT**
  OUT 参数是使用 registerOutParameter 方法处理的。registerOutParameter 的最常见形式是接收索引参数作为

第一个参数，并接收 SQL 类型作为第二个参数。这告诉 JDBC 驱动程序在处理语句时期望从参数得到什么样的数据。可以在 java.sql 包 Javadoc 中找到 registerOutParameter 方法的其它两个变体。

- **INOUT**
  INOUT 参数要求完成 IN 参数和 OUT 参数的工作。对于每个 INOUT 参数，在可以处理语句之前，必须调用设置方法和 registerOutParameter 方法。如果不设置或注册任何参数，则处理语句时将抛出 SQLException。

有关更多信息，请参考示例: 创建带有输入和输出参数的过程。

对于 PreparedStatement，除非再次调用设置方法，否则 CallableStatement 参数值在两次处理之间保持不变。clearParameters 方法不影响为输出注册的参数。在调用 clearParameters 之后，必须再次将所有 IN 参数设置为一个值，但所有 OUT 参数都不必再次注册。

注意: 一定不能将参数的概念与参数标记的索引混淆。存储过程调用期望将特定数目个参数传送给它。特定 SQL 语句使用 ? 字符（参数标记）来表示在运行时提供的值。请考虑以下示例来领会两个概念之间的差异:

```
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
```

通过名称访问存储过程参数: 存储过程的参数具有相关联的名称，如以下存储过程声明所示:

示例: 存储过程参数

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
CREATE
PROCEDURE MYLIBRARY.APROC
    (IN PARM1 INTEGER)
LANGUAGE SQL SPECIFIC MYLIBRARY.APROC
BODY: BEGIN
    <Perform a task here...>
END BODY
```

存在一个名为 PARM1 的整数参数。在 JDBC 3.0 中，支持通过名称以及通过索引来指定存储过程参数。为此过程设置 CallableStatement 的代码如下:

```
CallableStatement cs = con.prepareCall("CALL APROC(?)");

cs.setString("PARM1", 6);     //Sets input parameter at index 1 (PARM1) to 6.
```

有关更多信息，参见处理 CallableStatement。

处理 CallableStatement: 使用 CallableStatement 对象处理 SQL 存储过程调用是使用配合 PreparedStatement 对象使用的方法完成的。

返回存储过程的结果: 如果在存储过程内处理 SQL 查询语句，则可以使查询结果可供调用该存储过程的程序使用。还可以在存储过程中调用多个查询，调用程序可处理可用的所有 ResultSet。

有关更多信息，参见示例: 创建带有多个 ResultSet 的过程。

注意: 如果使用 executeQuery 来处理存储过程，并且没有返回 ResultSet，则抛出 SQLException。

并行地访问 ResultSet: 返回存储过程的结果处理 ResultSet 和存储过程，并提供了使用所有 Java(TM) Development Kit（JDK）发行版的示例。在示例中，按照从存储过程打开的第一个 ResultSet 到所打开的最后一个 ResultSet 的次序来处理 ResultSet。在使用下一个 ResultSet 之前将关闭前一个 ResultSet。

在 JDK 1.4 中，支持从存储过程中并行地使用 ResultSet。

注意: 在 V5R2 中，已通过“命令行接口”（CLI）将此功能添加至下层系统支持。因此，在早于 V5R2 的系统上运行的 JDK 1.4 不具有此项支持可用。

返回存储过程的更新计数: 返回存储过程的更新计数是 JDBC 规范中讨论的一项功能，但在 iSeries 平台上当前不支持此功能。没有办法从存储过程调用返回多个更新计数。如果需要来自存储过程中处理的 SQL 语句的更新计数，则有两种方法返回值:

- 将值作为输出参数返回。
- 将值作为返回值来从参数返回。这是输出参数的一种特殊情况。有关更多信息，参见处理具有返回值的存储过程。

处理具有未知预期返回的存储过程: 如果来自存储过程调用的结果是未知的，则应使用 execute 方法。在处理此方法之后，JDBC 驱动程序就可以通过 API 调用告诉应用程序存储过程生成了什么类型的结果。如果结果是一个或多个 ResultSet，则 execute 方法返回 true。存储过程调用不返回更新计数。

处理具有返回值的存储过程: iSeries 平台支持具有类似于函数返回值的返回值的存储过程。象其它参数标记那样对存储过程的返回值作标记，并且将其标记为由存储过程调用指定。下面就是这样的一个示例:

```
? = CALL MYPROC(?, ?, ?)
```

存储过程调用的返回值总是具有整数类型，并且必须象任何其它输出参数那样注册。

有关更多信息，参见示例: 创建具有返回值的过程。

示例: IBM Developer Kit for Java 的 CallableStatement 接口: 这是一个关于如何使用 CallableStatement 接口的示例。

示例: CallableStatement 接口

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

有关更多信息，参见 CallableStatement。

示例: 创建带有多个 ResultSet 的过程: **注意:** 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
        }
    }
}
```

示例: 创建带有输入和输出参数的过程: **注意**: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
        }
    }
}
```

示例: 创建带有返回值的过程: **注意**: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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. Note that return value support
            // is new in V4R5.
            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 assumptions 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 接口提供对通过运行查询生成的结果的访问。在概念上，可以将 ResultSet 的数据想象成一个带有特定数目个列以及特定数目个行的表。缺省情况下，表的行是按顺序检索的。在一行中，可以按任何次序访问列值。

要使用 ResultSet 对象，请查看下列各项:

ResultSet 特征
本节讨论 ResultSet 的特征，如:

- ResultSet 类型
- 并行性
- 通过提交 Connection 对象来关闭 ResultSet 的能力。
- ResultSet 特征的指定

游标移动
iSeries 的 “Java(TM) 数据库连接” （JDBC）驱动程序支持可滚动 ResultSet。借助可滚动 ResultSet，可使用许多游标定位方法来以任何次序处理数据行。

检索 ResultSet 数据
了解 ResultSet 对象如何提供用于获取行的列数据的方法。

更改 ResultSet
借助 iSeries JDBC 驱动程序，可以通过执行下列任务来更改 ResultSet:

- 更新行
- 删除行
- 插入行
- 更改定位更新

创建 ResultSet
可通过使用 Statement、PreparedStatement 或 CallableStatement 接口提供的 executeQuery 方法来创建 ResultSet 对象。本节还讨论了当应用程序不再需要 ResultSet 对象时将它们关闭。

ResultSet 特征:  缺省情况下，创建的所有 ResultSet 都具有仅向前类型、具有只读并行度并且跨提交边界保持游标。例外情况是 WebSphere 目前会更改游标可保持性缺省值，使得在提交时隐式地关闭游标。通过可在 Statement、PreparedStatement 和 CallableStatement 对象上访问的方法，可配置这些特征。

ResultSet 类型: ResultSet 类型指定关于 ResultSet 的下列信息:

- ResultSet 是否可滚动。
- “Java(TM) 数据库连接” （JDBC）ResultSet 的类型，这些类型由 ResultSet 接口中的常量来定义。

这些 ResultSet 类型的定义如下:

TYPE_FORWARD_ONLY
只能用来从 ResultSet 的开头处理到末尾的游标。这是缺省类型。

TYPE_SCROLL_INSENSITIVE
可用来通过 ResultSet 在各个方向上滚动的游标。此类型的游标对它处于打开状态时对数据库所作的更改不灵敏。它包含当处理查询或提取数据时满足查询的行。

TYPE_SCROLL_SENSITIVE
可用来通过 ResultSet 在各个方向上滚动的游标。此类型的游标对它处于打开状态时对数据库所作的更改灵敏。对数据库所作的更改直接影响 ResultSet 数据。

JDBC 1.0 ResultSet 总是仅向前的。在 JDBC 2.0 中添加了可滚动游标。

注意: 启用分块和块大小连接属性影响 TYPE_SCROLL_SENSITIVE 游标的灵敏度。分块通过在 JDBC 驱动程序层本身中对数据进行高速缓存来增强性能。

参见示例: 灵敏和不灵敏 ResultSet, 此示例显示了在将行插入表中时灵敏与不灵敏 ResultSet 之间的差异。

参见示例: ResultSet 灵敏度, 此示例显示了更改是如何根据 ResultSet 的灵敏度影响 SQL 语句的 WHERE 子句的。

并行性: 并行性确定了是否可更新 ResultSet。这些类型也是由 ResultSet 接口中的常量定义的。可用的并行性设置如下:

CONCUR_READ_ONLY
只能用于将数据读出数据库的 ResultSet。这是缺省设置。

CONCUR_UPDATEABLE
允许对其进行更改的 ResultSet。可以将这些更改放到下层数据库中。有关更多信息, 参见更改 ResultSet。

JDBC 1.0 ResultSet 总是仅向前的。在 JDBC 2.0 中添加了可更新 ResultSet。

注意: 根据 JDBC 规范, 如果值不能一起使用的话, 则允许 JDBC 驱动程序更改 ResultSet 并行性设置的 ResultSet 类型。在这样的情况下, JDBC 驱动程序在 Connection 对象上放置一个警告。

应用程序在一种情况下会指定 TYPE_SCROLL_INSENSITIVE, CONCUR_UPDATEABLE ResultSet。不灵敏性是通过复制数据来在数据库引擎中实现的。然后, 不允许通过该副本对下层数据库进行更新。如果指定此这种组合, 则驱动程序将把灵敏度更改为 TYPE_SCROLL_SENSITIVE 并创建一个警告, 指示已更改您的请求。

可保持性: 可保持性特征确定对 Connection 对象调用 commit 是否会关闭 ResultSet。用于使用可保持性特征的 JDBC API 是 V3.0 中新增的。然而, 本机 JDBC 驱动程序为若干个发行版提供了一个连接属性, 允许对在某连接之下创建的所有 ResultSet 指定缺省值 (参见 Connection 属性和 DataSource 属性)。API 支持将覆盖连接属性的任何设置。ResultSet 常量定义了可保持性特征的值, 如下所示:

HOLD_CURSOR_OVER_COMMIT
在调用 COMMIT 子句时所有打开的游标都保持打开。这是本机 JDBC 缺省值。

CLOSE_CURSORS_ON_COMMIT
在调用 COMMIT 子句时所有打开的游标都将关闭。

注意: 对连接调用 rollback 将始终关闭所有打开的游标。这是一个很少人知道的事实, 但却是数据库处理游标的常用方法。

根据 JDBC 规范, 游标可保持性的缺省值是由实现定义的。一些平台选择使用 CLOSE_CURSORS_ON_COMMIT 作为缺省值。对于大多数应用程序而言这通常不会成为一个问题, 但如果跨提交边界使用游标的话, 就必须要了解正在使用的驱动程序执行什么操作。Toolbox JDBC 驱动程序也使用 HOLD_CURSORS_ON_COMMIT 缺省值, 但 UDB for Windows(R) NT 的 JDBC 驱动程序的缺省值为 CLOSE_CURSORS_ON_COMMIT。

指定 ResultSet 特征: 在创建 ResultSet 对象之后, ResultSet 的特征就不会发生更改。因此, 必须在创建对象之前指定特征。可通过 createStatement、prepareStatement 和 prepareCall 方法的重载变体来指定这些特征。

请查看下列主题来指定 ResultSet 特征:

- 指定 Statement 的 ResultSet 特征
- 指定 PreparedStatement 的 ResultSet 特征和自动生成的键支持

- 指定 CallableStatement 的 ResultSet 特征和自动生成的键支持

注意: 有些 ResultSet 方法可用来获取 ResultSet 类型和 ResultSet 的并行性，但没有方法可用于获取 ResultSet 的可保持性。 «

示例: 灵敏和不灵敏 ResultSet: » 以下示例显示在将行插入到表中时灵敏与不灵敏 ResultSet 之间的差异。

示例: 灵敏和不灵敏 ResultSet

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
        }
    }
}
```

*示例: **ResultSet 灵敏度:*** 以下示例显示更改是如何根据 ResultSet 的灵敏度影响 SQL 语句的 WHERE 子句的。

示例: ResultSet 灵敏度

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
        }
    }
}
```

游标移动:  ResultSet.next 方法用来在 ResultSet 中移动，每次移动一行。借助 “Java(TM) 数据库连接”（JDBC）2.0，iSeries JDBC 驱动程序支持可滚动 ResultSet。可滚动 ResultSet 允许通过使用 previous、absolute、relative、first 和 last 方法来以任何次序处理数据行。

缺省情况下， JDBC ResultSet 始终只能前进，这表示唯一可调用的有效游标定位方法是 next()。必须显式地请求可滚动 ResultSet。有关更多信息，参见 ResultSet 类型。

借助可滚动 ResultSet，可使用下列游标定位方法:

方法	描述
Next	此方法将游标在 ResultSet 中向前移动一行。 如果游标定位在有效的行上，则此方法返回 true，否则返回 false。
Previous	此方法将游标在 ResultSet 中向后移动一行。 如果游标定位在有效的行上，则此方法返回 true，否则返回 false。
First	此方法将游标移至 ResultSet 中的第一行。 如果游标定位在第一行上，则此方法返回 true，如果 ResultSet 是空的，则返回 false。
Last	此方法将游标移至 ResultSet 中的最后一行。 如果游标定位在最后一行上，则此方法返回 true，如果 ResultSet 是空的，则返回 false。
BeforeFirst	此方法将游标移至刚好位于 ResultSet 中的第一行之前的位置。 对于空的 ResultSet，此方法没有任何作用。此方法不返回值。
AfterLast	此方法将游标移至刚好位于 ResultSet 中的最后一行之后的位置。 对于空的 ResultSet，此方法没有任何作用。此方法不返回值。
Relative (int rows)	此方法相对于游标的当前位置来移动游标。 • 如果 rows 为零，则此方法没有任何作用。 • 如果 rows 为正数，则游标向前移动该行数。如果当前位置与 ResultSet 末尾之间的行数少于输入参数所指定的行数，则此方法的操作与 afterLast 相似。 • 如果 rows 为负数，则游标向后移动该行数。如果当前位置与 ResultSet 末尾之间的行数少于输入参数所指定的行数，则此方法的操作与 beforeFirst 相似。 如果游标定位在有效的行上，则此方法返回 true，否则返回 false。

方法	描述
Absolute (int row)	此方法将游标移至 row 值指定的行。 如果 row 值是正数，则游标定位在距离 ResultSet 开头该行数的位置处。第一行的编号为 1，第二行为 2，依此类推。如果 ResultSet 中的行数少于 row 值所指定的行数，则此方法的操作方式与 afterLast 相同。 如果 row 值是负数，则游标定位在距离 ResultSet 末尾该行数的位置处。最后一行的编号为 -1，倒数第二行的编号为 -2，依此类推。如果 ResultSet 中的行数少于 row 值所指定的行数，则此方法的操作方式与 beforeFirst 相同。 如果 row 值为 0，则此方法的操作方式与 beforeFirst 相同。 如果游标定位在有效的行上，则此方法返回 true，否则返回 false。

检索 ResultSet 数据:  ResultSet 对象提供了若干个用于获取行的列数据的方法。所有这些方法的格式均为 get<*Type*>，其中 <*Type*> 是 Java(TM) 数据类型。这些方法的一些示例包括 getInt、getLong、getString、getTimestamp 和 getBlob。几乎所有这些方法都接收单一参数，该参数是 ResultSet 中的列索引或列名称。

ResultSet 列具有从 1 开始的编号。如果使用了列名称，并且 ResultSet 中存在多个同名的列，则返回第一个列。一些 get<Type> 方法具有附加的参数，如可以传送给 getTime、getDate 和 getTimestamp 的可选 Calendar 对象。请参考 java.sql 包的 Javadoc 以获取完整的详细信息。

对于返回对象的 get 方法，当 ResultSet 为空时返回值便为空。对于原语类型，不能返回空。在这些情况下，值为 0 或 false。如果应用程序必须对空、0 或 false 加以区分，则可以在调用之后立即使用 wasNull 方法。于是，此方法可确定该值实际上是 0 还是 false 值，或者该值是否是因为 ResultSet 值真的为空而返回的。

有关如何使用 ResultSet 接口的示例，参见示例: IBM Developer Kit for Java的 ResultSet 接口。

ResultSetMetaData 支持: 当对 ResultSet 对象调用 getMetaData 方法时，该方法将返回用于描述该 ResultSet 对象的列的 ResultSetMetaData 对象。尽管在运行时之前所处理的 SQL 语句是未知的，但可使用 ResultSetMetaData 方法来确定应使用什么 get 方法来检索数据。以下代码示例使用 ResultSetMetaData 来确定结果集中的每个列类型:

示例: 使用 ResultSetMetaData 来确定结果集中的每个列类型

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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);
```

有关如何使用 ResultSetMetaData 接口的示例，参见示例: IBM Developer Kit for Java的 ResultSetMetaData 接口。

更改 ResultSet: » ResultSet 的缺省设置是只读的。然而，借助“Java(TM) 数据库连接”（JDBC）2.0，iSeries JDBC 驱动程序提供了对可更新 ResultSet 的全面支持。您可以参考 ResultSet 并行性以了解如何更新 ResultSet。

更新行: 可通过 ResultSet 接口来更新数据库表中的行。此过程涉及的步骤如下:

1. 使用各种 update<*Type*> 方法来更改特定行的值，其中 <*Type*> 是 Java 数据类型。这些 update<*Type*> 方法与可用于检索值的 get<*Type*> 方法相对应。
2. 对下层数据库应用行。

在执行第二个步骤之前，不会更新数据库本身。在不调用 updateRow 方法的情况下更新 ResultSet 中的列并不会对数据库作任何更改。

可使用 cancelUpdates 方法来撤销对行的计划更新。在调用 updateRow 方法之后，对数据库所作的更改就会成为最终更改，也就不能撤销了。

注意: 由于数据库没办法指出已更新了哪些行，所以 rowUpdated 方法始终返回 false。相应地，updatesAreDetected 方法返回 false。

删除行: 可通过 ResultSet 接口来删除数据库表中的行。提供了 deleteRow 方法，此方法用于删除当前行。

插入行: 可通过 ResultSet 接口来将行插入数据库表。此过程需要使用“插入行”，应用程序明确地将游标移至该行并构建要插入到数据库中的值。此过程涉及的步骤如下:

1. 将游标定位在插入行上。
2. 在新行中设置每个列值。
3. 将该行插入到数据库中，并可选择将游标移回到 ResultSet 中的当前行。

注意: 并不会将新行插入到表中游标的所在位置。而是通常将新行添加至表数据空间的末尾。缺省情况下，关系数据库不依赖于位置。例如，您不应该期望将游标移至第三行并插入当后续用户提取数据时将在第四行之前显示的内容。

对定位更新的支持: 除了通过 ResultSet 更新数据库的方法以外，可使用 SQL 语句来发出定位更新。此项支持依赖于使用命名游标。JDBC 提供了 Statement 中的 setCursorName 方法和 ResultSet 中的 getCursorName 方法来提供对这些值的访问。

由于本机 JDBC 驱动程序支持定位更新功能，所以两个 DatabaseMetaData 方法 supportsPositionedUpdated 和 supportsPositionedDelete 都返回 true。

有关更多信息，参见示例: 通过另一个语句的游标来使用语句更改值。

有关更多信息，参见示例: 通过另一个语句的游标来从表中除去值。 «

示例: 通过另一个语句的游标来从表中除去值: » 这是有关如何通过另一个语句的游标来从表中除去值的示例。

示例: 通过另一个语句的游标来从表中除去值

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
        }
    }
}
```

示例: 通过另一个语句的游标来使用语句更改值: 这是有关如果通过另一个语句的游标来使用语句更改值的示例。

示例: 通过另一个语句的游标来使用语句更改值

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
        }
    }
}
```

创建 ResultSet: 要创建 ResultSet 对象，可使用 Statement、PreparedStatement 或 CallableStatement 接口中的 executeQuery 方法。然而，也可使用其它方法。例如，getColumns、getTables、getUDTs 和 getPrimaryKeys 等 DatabaseMetaData 方法返回 ResultSet。也有可能让单一 SQL 语句返回多个 ResultSet 来进行处理。在调用 Statement、PreparedStatement 或 CallableStatement 接口提供的 execute 方法之后，还可使用 getResultSet 方法来检索 ResultSet 对象。

有关更多信息，参见示例: 创建带有多个 ResultSet 的过程。

关闭 ResultSet: 尽管在关闭相关联的 Statement 对象时将自动关闭 ResultSet 对象，但建议您在使用 ResultSet 对象完毕后将它们关闭。这样做将立即释放内部数据库资源，从而可以提高应用程序吞吐量。

关闭由 DatabaseMetaData 调用生成的 ResultSet 也十分重要。由于不能直接访问用来创建这些 ResultSet 的 Statement 对象，所以不直接对该 Statement 对象调用 close。这些对象以特定的方式链接在一起，当您关闭外部 ResultSet 对象时，JDBC 驱动程序将关闭内部的 Statement 对象。尽管即使不以手工方式关闭这些对象系统也能够继续工作，然而这样将不必要地多使用一些资源。

注意: ResultSet 的可保持性特征也可代替您自动关闭 ResultSet。允许对 ResultSet 对象多次调用 close。

示例: IBM Developer Kit for Java 的 ResultSet 接口: 这是一个关于如何使用 ResultSet 接口的示例。

示例 1: ResultSet 接口

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
            int rowCount = 0;
            while (rs.next()) {
                rowCount++;
                System.out.println("Data for row " + rowCount);
                for (int i = 1; i <= colCount; i++)
                    System.out.println("   Row " + i + ": " + rs.getString(i));
            }

        } 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");
                }
            }
        }
    }
}
```

示例: IBM Developer Kit for Java 的 ResultSetMetaData 接口: **注意**: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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");
                }
            }
        }
    }
}
```

JDBC 对象合用

在讨论 “Java(TM) 数据库连接” （JDBC）和性能时，对象合用是最常提到的主题。由于 JDBC 中使用的许多对象的创建成本都很高昂，如 Connection、Statement 和 ResultSet 对象，因此通过重复使用这些对象（而不是在每次需要它们时进行创建）可以显著提高性能。

许多应用程序已代替您处理对象合用。例如，WebSphere 具有对合用 JDBC 对象的广泛支持并允许您控制如何管理合用。正因为如此，您不必关心自己的合用机制就能够获得所要的功能。然而，当未提供此支持时，您必须找到用于除不重要的应用程序之外的所有应用程序的解决方案。

要在 JDBC 程序中使用对象合用，请查看下列各项:

将 DataSource 支持用于对象合用

可使用 DataSource 来让多个应用程序共享一个公共的配置来访问数据库。这是通过让每个应用程序都引用同一个 DataSource 名实现的。

ConnectionPoolDataSource 属性

可使用 ConnectionPoolDataSource 接口提供的属性集来对其进行配置。

基于 DataSource 的语句合用

可在连接池内使用语句合用。UDBConnectionPoolDataSource 接口的 maxStatements 属性允许 DataSource 指定可以在一个连接之下合用多少个语句。

构建您自己的合用解决方案

您可以开发自己的连接和语句合用，而不需要 DataSource 的支持或依靠另一产品。

将 DataSource 支持用于对象合用:  通过使用 DataSource，可以让多个应用程序共享一个公共的配置来访问数据库。这是通过让每个应用程序都引用同一个 DataSource 名实现的。

通过使用 DataSource，可以从一个中央位置更改许多个应用程序。例如，如果更改所有应用程序所使用的缺省库的名称，并且已使用单一 DataSource 来获取所有这些应用程序的连接，则可在该 DataSource 中更新集合的名称。于是，所有应用程序都将开始使用新的缺省库。

当使用 DataSource 来获取应用程序的连接时，可使用本机 JDBC 驱动程序对连接池的内置支持。此项支持是作为 ConnectionPoolDataSource 接口的实现提供的。

合用是通过交出“逻辑”Connection 对象而不是物理 Connection 对象实现的。**逻辑 Connection 对象**是由合用的 Connection 对象返回的连接对象。每个逻辑连接对象都用作由合用的连接对象表示的物理连接的临时句柄。对于应用程序而言，当返回 Connection 对象时，两者之间不存在显著的区别。在对 Connection 对象调用 close 方法时，存在着微妙的区别。此调用将使逻辑连接无效并将物理连接返回到池中，于是另一个应用程序便能够在池中使用该物理连接。此项技术允许许多个逻辑连接对象重复使用单一物理连接。

设置连接池: 连接池是通过创建引用 ConnectionPoolDataSource 对象的 DataSource 对象实现的。ConnectionPoolDataSource 对象具有许多属性，可以设置那些属性来处理与池维护相关的各个方面。

有关更多的详细信息，请参考有关如何使用 UDBDataSource 和 UDBConnectionPoolDataSource 来设置连接池的示例。您还可以查看 Java 命名和目录接口（JNDI）以了解关于 JNDI 在此示例中所扮演的角色的详细信息。

在示例中，将两个 DataSource 对象绑定到一起的链接是 dataSourceName。此链接告知 DataSource 对象延迟建立与自动管理合用的 ConnectionPoolDataSource 对象的连接。

合用与非合用应用程序: 在使用 Connection 合用与不使用 Connection 合用的应用程序之间没有区别。因此可以在完成应用程序代码之后添加合用支持，而不必对应用程序代码作任何更改。

有关更多详细信息，请参考示例: 测试连接池的性能。

以下是在开发期间以本地方式运行前一程序所产生的输出。

```
Start timing the non-pooling DataSource version...
Time spent: 6410

Start timing the pooling version...
Time spent: 282

Java program completed.
```

缺省情况下，UDBConnectionPoolDataSource 对单一连接进行合用。如果应用程序需要连接若干次并且每次只需要一个连接，则使用 UDBConnectionPoolDataSource 是不错的解决方案。如果需要许多个并行连接，则必须配置 ConnectionPoolDataSource 以便与您的需要和资源相匹配。 

示例: 使用 UDBDataSource 和 UDBConnectionPoolDataSource 来设置连接池: 这是有关如何将连接池与 UDBDataSource 和 UDBConnectionPoolDataSource 配合使用的示例。

示例: 使用 UDBDataSource 和 UDBConnectionPoolDataSource 来设置连接池

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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);
  }
}
```

示例: 测试连接池的性能:  这是有关如何针对非合用示例的性能测试合用示例的性能的示例。

示例: 测试连接池的性能

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 属性:  ConnectionPoolDataSource 接口提供了一组属性，可通过这些属性来对其进行配置。下表提供了这些属性的描述。

属性	描述
initialPoolSize	第一次将池实例化时，此属性确定将多少个连接放到池中。如果将此值指定为超出 minPoolSize 和 maxPoolSize 的范围，则使用 minPoolSize 或 maxPoolSize 来作为要创建的初始连接的数目。
maxPoolSize	在使用池时，可以请求比池所拥有的连接数更多的连接。此属性指定允许在池中创建的连接的最大数目。 当池具有最大的大小并且所有连接都已在使用中时，应用程序并不会“阻塞”并等待连接返回到池中。而是，JDBC 驱动程序根据 DataSource 属性来构造新连接并返回该连接。 如果将 maxPoolSize 指定为 0，则只要系统有可以交出的资源，便允许池无限增大。
minPoolSize	长时间使用池会导致其中的连接数增加。如果活动级别降低到某一程度而使得永远不会将一些 Connection 从池中拉出，则会没有特定理由地占用资源。 在这样的情况下，JDBC 驱动程序有能力释放一些已累积起来的连接。此属性允许您告知 JDBC 释放连接，确保始终有特定数目的连接可供使用。 如果将 minPoolSize 指定为 0，则池就有可能释放它的所有连接，而应用程序也就有可能对每个连接请求都要花时间建立连接。
maxIdleTime	连接对它们处于空闲状态（未在使用中）的时间长度进行跟踪。此属性指定应用程序在释放连接之前允许连接处于未使用状态多长时间（即，存在的连接数多于所需的连接数）。 此属性是以秒计的时间，而没有指定实际发生关闭的时间。它指定应该在经过多长的一段足够时间后释放连接。
propertyCycle	此属性表示在强制实施这些规则之间允许经过的秒数。

注意: 将 maxIdleTime 或 propertyCycle 时间设置为 0 表示 JDBC 驱动程序自己不检查将要从池中除去的连接。仍强制实施对初始、最小和最大大小指定的规则。

当 maxIdleTime 和 propertyCycle 不是 0 时，使用管理线程来对池进行监控。此线程每隔 propertyCycle 秒苏醒一次并检查池中的所有连接，以查看哪些连接的未使用时间已超过 maxIdleTime 秒。在达到 minPoolSize 之前，将从池中除去符合此条件的连接。

基于 DataSource 的语句合用: » UDBConnectionPoolDataSource 接口上的另一个可用属性是 maxStatements。此属性允许在连接池内进行语句合用。语句合用仅对 PreparedStatement 和 CallableStatement 有效。不对 Statement 对象进行合用。

语句合用的实现与连接池的实现类似。当应用程序调用 Connection.prepareStatement(″select * from tablex″) 时，合用模块便检查是否已在该连接之下准备了 Statement 对象。如果是的话，便将逻辑 PreparedStatement 对象而不是物理对象交给您。当您调用 close 时，便将 Connection 对象返回到池中，丢弃逻辑 Connection 对象，于是便可重复使用该 Statement 对象。

maxStatements 属性允许 DataSource 指定可以在一个连接之下合用多少个语句。0 值指示不应使用语句合用。当语句池变满时，应用最近最少使用算法来确定要丢弃哪一个语句。

示例: 测试两个 DataSource的性能测试一个只使用连接池的 DataSource 与另一个使用语句和连接池的 DataSource。

以下示例是在开发期间以本地方式运行此程序的输出。

```
Deploying statement pooling data source
Start timing the connection pooling only version...
Time spent: 26312

Starting timing the statement pooling version...
Time spent: 2292
Java program completed
```
«

示例: 测试两个 DataSource 的性能: » 这是有关测试一个只使用连接池的 DataSource 与另一个使用语句和连接池的 DataSource 的示例。

示例: 测试两个 DataSource 的性能

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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);

    }
}
```

构建您自己的连接池: 您可以开发自己的连接和语句合用，而不需要 DataSource 的支持或依靠另一产品。

我们使用小型 Java 应用程序来演示合用技术，但此技术同样适用于 servlet 或大型 N 层应用程序。本示例用来演示性能问题。

演示应用程序具有两个功能:

- 将新的索引和名称插入数据库表。
- 从表中读取给定索引的名称。

可以从 IBM 的 Developer Kit for Java JDBC Web 页面 下载完整的应用程序代码。

示例应用程序的性能并不很好。在标准工作站上，通过此代码运行 100 个 getValue 方法调用和 100 个 putValue 方法调用平均需要 31.86 秒。

问题在于每个请求的数据库工作过多。即，您需要获取连接、获取语句、处理语句、关闭语句和关闭连接。不应该在每个请求之后废弃所有信息，而是必须有一种方法来重复使用此过程的某些部分。**连接池**指的是将用于创建连接的代码替换为用于从池中获取连接的代码，然后将用于关闭连接的代码替换为用于将连接返回到池中以供使用的代码。

连接池的构造函数创建连接并将它们放到池中。池类具有用于定位要使用的连接以及用于在完成使用连接后将连接返回到池中的 take 和 put 方法。由于池对象是共享资源，所以这些方法是同步的，但您不会想让多个线程同时尝试操纵合用的资源。

对 getValue 方法的调用代码作了更改。未显示 putValue 方法，但对其作了精确的更改，可以从 IBM 的 Developer Kit for Java JDBC Web Page  或者此更改。连接池对象的实例化也没有显示。您可以调用构造函数并传入要在池中创建的连接对象的数目。应该在启动应用程序的时候完成此步骤。

适当应用连接池代码后，运行具有这些更改的前一应用程序（即，进行 100 个 getValue 方法和 100 个 putValue 方法请求）的平均时间将缩短为 13.43 秒。工作量的处理时间与不进行连接池时的原始处理时间相比削减过半。

构建您自己的语句合用: 当使用连接池时，处理每个语句时的创建和关闭语句也浪费了时间。这是另一个浪费可重复使用的对象的一个示例。

要重复使用对象，可使用准备语句类。在大多数应用程序中，将重复使用同一 SQL 语句而只作小幅更改。例如，通过应用程序进行的一个迭代可生成以下查询:

```
SELECT * from employee where salary > 100000
```

下一个迭代可生成以下查询:

```
SELECT * from employee where salary > 50000
```

这是同一个查询，但使用不同的参数。使用以下查询可完成这两个查询:

```
SELECT * from employee where salary > ?
```

然后，在处理第一个查询时，可将参数标记（由问号指示）设置为 10000，在处理第二个查询时设置为 50000。由于以下三个原因，这将提供优于连接池可以提供的性能:

- 创建的对象更少。创建并重复使用 PreparedStatement 对象，而不是为每个请求创建 Statement 对象。因此，运行的构造函数更少。
- 可重复使用用于设置 SQL 语句的数据库工作（称为**准备**）。由于准备 SQL 语句的过程涉及确定 SQL 语句文本的内容以及系统应如何完成所请求的任务，所以此过程具有合理的成本开销。
- 在除去附加的对象创建时，有一个通常不会考虑到的好处。不需要破坏未创建的内容。此模型在 Java 垃圾收集器上更为容易，并且随着时间的推移将给许多用户带来性能方面的好处。

可将演示程序更改为合用 PreparedStatement 对象而不是 Connection。通过更改程序，您能够重复使用更多的对象并改进性能。您可以从编写包含将要合用的对象的类入手。这个类必须封装将要使用的各种资源。对于连接池示例，Connection 是合用的唯一资源，因此不需要封装类。每个合用对象都必须包含一个 Connection 和两个 PreparedStatement。然后，您可以创建包含数据库访问对象而不是连接的池类。

最后，应用程序必须更改为获取数据库访问对象并指定要使用该对象中的哪些资源。并不需要指定特定的资源，应用程序可以保持不变。

进行此项更改之后，同一测试的运行时间现在平均为 0.83 秒。此时间大约比程序的原始版本快 38 倍。

注意事项: 通过复制改进了性能。如果不重复使用某个项，则它浪费了所合用的资源。

大多数应用程序包含关键的代码部分。通常，应用程序在 10% 到 20% 的代码上花费了 80% 至 90% 的处理时间。如果应用程序中有可能使用 10,000 个 SQL 语句，则不会对所有 SQL 语句进行合用。我们的目标是对应用程序的关键代码部分中使用的 SQL 语句进行标识与合用。

在 Java 实现中创建对象的成本可能十分高昂。您可以从使用合用解决方案中受益。进程中使用的对象是在开始时（即在其它用户尝试使用系统之前）创建的。根据需要重复地使用这些对象。性能十分优异，并且有可能随着时间的过去不断地微调应用程序以使其能够更好地供更大数目的用户使用。结果是，对更多的对象进行合用。而且，这允许对应用程序的数据库访问进行更高效的多线程化，从而获得更大的吞吐量。

Java（使用 JDBC）基于动态 SQL，其速度往往比较缓慢。合用可以最大程度地缓解此问题。通过在启动时准备语句，可以静态地表述对数据库的访问。在准备语句之后，动态 SQL 与静态 SQL 之间只存在很小的性能差异。

Java 中的数据库访问性能可以很高效，并且可以在不牺牲面向对象设计或代码可维护性的前提下实现。编写代码来构建语句和连接池并不困难。此外，还可将代码代码更改并增强为支持多个应用程序和多种应用程序类型（基于 Web，客户机／服务器），等等。 

批处理更新

 JDBC 2.0 中的一项新功能是批处理更新支持。此功能允许对数据库所作的任何更新作为用户程序与数据库之间的单一事务传送。当必须同时执行许多更新时，此过程可以显著改进性能。例如，如果一间大公司要求它新雇用的雇员在星期一开始工作，则此项要求使得有必要同时对雇员数据库处理许多更新（在此情况下是插入）。创建一批更新并将它们作为一个单元提交至数据库可以节省处理时间。

批处理更新分为两类:

- 使用 Statement 对象的批处理更新。
- 使用 PreparedStatement 对象的批处理更新。

要使用批处理更新支持，请查看下列各项:

Statement 批处理更新

在执行 Statement 批处理更新之前，必须确保关闭自动提交。当自动提交设置处于关闭状态时，可以创建标准的 Statement 对象。然后，可使用 addBatch 方法来将语句添加至批处理。在添加所有要进行批处理的语句之后，可使用 executeBatch 方法来处理它们，也可随时使用 clearBatch 方法来将批处理清空。

PreparedStatement 批处理更新

preparedStatement 批处理与 Statement 批处理类似。然而，preparedStatement 批处理总是处理同一个已准备的语句，您只能更改该语句的参数。

BatchUpdateException

当对 executeBatch 方法的调用失败时，将抛出 BatchUpdateException。BatchUpdateException 允许您调用为了接收消息、SQLState 和供应商代码而始终需要调用的所有那些方法。BatchUpdateException 还提供了返回整数数组的 getUpdateCounts 方法。该整数数组包含来自批处理中到发生故障那一刻为止已处理的所有语句的更新计数。

分块插入支持

可使用分块插入（这是一个 iSeries 操作）来一次将若干行插入数据库表。

Statement 批处理更新: » 要执行 Statement 批处理更新，必须关闭自动提交。在 “Java(TM) 数据库连接”（JDBC）中，自动提交在缺省情况下处于打开状态。自动提交表示在处理每个 SQL 语句后提交对数据库所作的任何更新。如果要将正在交给数据库的一组语句视为一个功能组，则您不会想让数据库个别地提交每个语句。如果不关闭自动提交，并且位于批处理中间的语句失败，则由于已将半数语句最终化，所以不能回滚整个批处理并进行再试。此外，提交批处理中的每个语句这一附加工作将引起大量的开销。有关更多的详细信息，参见事务。

在关闭自动提交之后，便可以创建标准的 Statement 对象了。并不是使用 executeUpdate 之类的方法来处理语句，而是，使用 addBatch 方法来将它们添加至批处理。在添加所有要进行批处理的语句之后，可使用 executeBatch 方法来处理它们。随时可使用 clearBatch 方法来将批处理清空。

以下示例显示了如何使用这些方法:

示例: Statement 批处理更新

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
```

在此示例中，从 executeBatch 方法返回一个整数数组。对于在批处理中处理的每个语句，此数组都包含一个整数值。如果正在将值插入到数据库中，则每个语句的值都是 1（即，假定处理成功）。然而，一些语句可能是影响多行的更新语句。如果将任何除 INSERT、UPDATE 或 DELETE 之外的语句放到批处理中，则将发生异常。

«

PreparedStatement 批处理更新: » preparedStatement 批处理与 Statement 批处理类似；然而，preparedStatement 批处理总是处理同一个 “已准备的” 语句，您只能更改该语句的参数。以下是使用 preparedStatement 批处理的示例。

示例: PreparedStatement 批处理更新

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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: » 关于批处理更新的一个重要注意事项是当 executeBatch 方法调用失败时要执行什么操作。在这种情况下，将抛出一种新的异常，此异常名为 BatchUpdateException。BatchUpdateException

是 SQLException 的子类，它允许您调用为了接收消息、SQLState 和供应商代码而始终需要调用的所有那些方法。BatchUpdateException 还提供了返回整数数组的 getUpdateCounts 方法。该整数数组包含来自批处理中到发生故障那一刻为止已处理的所有语句的更新计数。数组长度告诉您批处理中的哪一个语句失败了。例如，如果异常中返回的数组的长度为 3，则表示批处理中的第四个语句失败了。因此，根据所返回的单一 BatchUpdateException 对象，您可以确定所有已成功的语句的更新计数、哪一个语句失败了以及关于故障的所有信息。

目前，处理批处理更新的标准性能与独立地处理每个语句的性能相等。有关对批处理更新的优化支持的更多信息，您可以参考分块插入支持。在编码和利用将来的性能优化时，您仍然应该使用新模型。

注意: 在 JDBC 2.1 规范中，提供了另一个选项来控制对批处理更新的异常条件的处理方式。JDBC 2.1 引入了一个模型，允许正在处理的批处理在批处理项失败后继续。在对每个失败项返回的更新计数整数数组中放置一个特殊的更新计数。这使得大型批处理即使在它们的其中一个项失败时也能够继续处理。有关这两种操作模型的详细信息，参见 JDBC 2.1 或 JDBC 3.0 规范。缺省情况下，本机 JDBC 驱动程序使用 JDBC 2.0 定义。此驱动程序提供了在使用 DriverManager 来建立连接时将要使用的 Connection 属性。此驱动程序还提供了在使用 DataSource 来建立连接时将要使用的 DataSource 属性。这些属性允许应用程序选择要让批处理操作如何处理故障。

分块插入支持:  **分块插入**是 iSeries 服务器上的一种特殊类型的操作，这种操作提供了一种高度优化的方法来一次将若干个行插入数据库表。可以将分块插入想象成批处理更新的子集。批处理更新可以是任何形式的更新请求，但分块插入是特定的。然而，批处理更新的分块插入类型是公共的；本机 JDBC 驱动程序已更改为能够利用此项功能。

由于使用分块插入支持时存在系统限制，所以本机 JDBC 驱动程序的缺省设置是禁用分块插入。通过 Connection 属性或 DataSource 属性可以将其启用。可以代替您检查和处理在使用分块插入时的大多数限制，但有几个限制却不能；因此，这就是缺省情况下关闭分块插入支持的原因。限制的列表如下:

- 所使用的 SQL 语句必须是带有 VALUES 子句的 INSERT 语句，这表示它不是带有 SUBSELECT 的 INSERT 语句。JDBC 驱动程序能够识别此项限制并执行适当的操作过程。
- 必须使用 PreparedStatement，这表示不存在对 Statement 对象的优化支持。JDBC 驱动程序能够识别此项限制并执行适当的操作过程。
- SQL 语句必须对表中的所有列指定参数标记。这表示不能对列使用常量值，也不能允许数据库对任何列插入缺省值。JDBC 驱动程序没有一种机制来处理对 SQL 语句中的特定参数标记的测试。如果将属性设置为执行优化分块插入，并且不在 SQL 语句中避免缺省值或常量，则数据库表中最终的值不正确。
- 必须连接至本地系统。这表示由于 DRDA 不支持分块插入操作，所以使用 DRDA 来访问远程系统的连接是不能使用的。JDBC 驱动程序没有一种机制来处理对面向本地系统的连接的测试。如果将属性设置为执行优化分块插入，并且尝试连接至远程系统，则批处理更新的处理将失败。

此代码示例显示了如何启用对分块插入处理的支持。此代码与不使用分块插入支持的版本之间的唯一区别是对 Connection URL 添加了 `use block insert=true`。

示例: 分块插入处理

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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...
```

在类似的测试案例中，分块插入要比不使用分块插入时执行同一操作快若干倍。例如，在使用分块插入时，对前面的代码执行的测试要快 9 倍。对于只使用原语类型而不使用对象的案例，速度可以快 16 倍。在要执行相当大量工作的应用程序中，应当适当地更改预期。

高级数据类型


在带有 V4R4 e-PACK 的 iSeries 数据库中，提供了若干种称为 SQL3 数据类型的新数据类型。 “Java(TM) 数据库连接”（JDBC）2.0 和更高版本提供了对使用这些作为 SQL99 标准一部分的数据类型的支持。

SQL3 数据类型提供了极大的灵活性。它们适合于存储序列化的 Java 对象、“可扩展标记语言”（XML）文档以及多媒体数据，如歌曲、产品图片、雇员照片和电影剪辑。

单值类型: **单值类型**是基于标准数据库类型的用户定义类型。例如，可以定义在内部作为 CHAR(9) 的“社会保障号”类型 SSN。以下 SQL 语句创建这样的 DISTINCT 类型。

```
CREATE DISTINCT TYPE CUJOSQL.SSN AS CHAR(9)
```

单值类型总是映射至内置数据类型。有关如何以及何时在 SQL 的上下文中使用单值类型的更多信息，请查阅 SQL 参考手册。

要在 JDBC 中使用单值类型，可以采用访问下层类型的方法来访问它们。getUDTs 方法是一个新方法，它允许您查询系统上有哪些单值类型可用。这个示例程序显示了下列各项:

- 单值类型的创建。
- 使用单值类型的表的创建。
- 使用 PreparedStatement 来设置单值类型参数。
- 使用 ResultSet 来返回单值类型。
- 使用对 getUDTs 的元数据“应用程序编程接口”（API）调用来了解关于单值类型的信息。

大对象: 共有三种类型的“大对象”（LOB）:

- 二进制大对象（BLOB）
- 字符大对象（CLOB）
- 双字节字符大对象（DBCLOB）

除字符数据的内部存储表示法之外，DBCLOB 与 CLOB 类似。由于 Java 和 JDBC 将所有字符数据作为 Unicode 来外部化，所以在 JDBC 中只支持 CLOB。从 JDBC 的观点看来，DBCLOB 的工作可以与 CLOB 支持互换。

二进制大对象: 在许多方面，“二进制大对象”（BLOB）列与可以非常大的 CHAR FOR BIT DATA 列类似。可以在这些列中存储任何可以表示为未转换字节的流的内容。通常，使用 BOLB 列来存储已序列化的 Java 对象、图片、歌曲和其它二进制数据。

可以象使用其它标准数据库类型那样使用 BLOB。可以将它们传送至存储过程、在准备的语句中使用它们以及在结果集中更新它们。PreparedStatement 类有一个用于将 BLOB 传送至数据库的 setBlob 方法，而 ResultSet 类添加了用于从数据库中检索 BLOB 的 getBlob 类。在 Java 程序中，使用作为 JDBC 接口的 BLOB 对象来表示 BLOB。

有关如何使用 BLOB 的更多信息，请参考编写使用 BLOB 的代码。

字符大对象: “字符大对象”（CLOB）是 BLOB 的字符数据实现。并非将数据不加转换地存储在数据库中，而是，将数据作为文本存储在数据库中，并按照处理 CHAR 列的方式来处理数据。与 BLOB 一样，JDBC 2.0 提供了用于直接处理 CLOB 的函数。PreparedStatement 接口包含一个 setClob 方法，而 ResultSet 接口包含一个 getClob 方法。

有关如何使用 CLOB 的更多信息，请参考编写使用 CLOB 的代码。

尽管 BLOB 和 CLOB 列的工作方式与 CHAR FOR BIT DATA 和 CHAR 列相似，但这是从外部用户的观点看上去它们在概念上所具有的工作方式。在内部，工作方式并不相同；由于“大对象”（LOB）列有可能具有非常庞大的大小，所以您通常间接地使用数据。例如，当从数据库中提取行块时，不将 LOB 块移至 ResultSet。而是将称为 LOB 定位器的指针（即四个字节的整数）移到 ResultSet 中。然而，在 JDBC 中使用 LOB 时，没必要了解关于定位器的信息。

Datalink: **Datalink** 是封装的值，它包含从数据库到存储于数据库外部的文件的逻辑引用。根据您使用的是 JDBC 2.0 或更新版本还是使用的是 JDBC 3.0 或更新版本，从 JDBC 的观点看来，以两种不同的方式表示和使用 Datalink。

有关如何使用 Datalink 的更多信息，请参考编写使用 Datalink 的代码。

不支持的 SQL3 数据类型: 还有一些其它的 SQL3 数据类型，那些数据类型已定义，并且 JDBC API 对它们提供了支持。这些数据类型是 ARRAY、REF 和 STRUCT。目前，iSeries 服务器不支持这些类型。因此，JDBC 驱动程序没有提供任何形式的对它们的支持。

编写使用 BLOB 的代码:  通过 Java(TM) 数据库连接（JDBC）应用程序编程接口（API），可以对数据库“二进制大对象”（BLOB）列完成许多任务。下列主题简要讨论了这些任务并包括有关如何完成这些任务的示例。

从数据库读取 BLOB 以及将 BLOB 插入数据库: 借助 JDBC API，可使用多种方法从数据库中取出 BLOB 以及将 BLOB 放到数据库中。然而，没有标准化的办法来创建 BLOB 对象。如果数据库已经装满了 BLOB，则这并不是一个问题，但是如果要通过 JDBC 来使用暂存区中的 BLOB，则会引起问题。我们没有为 JDBC API 的 BLOB 和 CLOB 接口定义构造函数，而是提供了用于将 BLOB 作为其它类型来直接放入数据库以及直接从数据库中将它们取出的支持。例如，setBinaryStream 方法可使用 BLOB 类型的数据库列。此示例显示了一些将 BLOB 放入数据库以及从数据库检索它们的常用方法。

使用 BLOB 对象 API: 在 JDBC 中，将 BLOB 定义成接口，各种驱动程序提供了此接口的各种实现。此接口带有一系列可用来与 BLOB 对象交互作用的方法。此示例显示了一些可使用此 API 执行的公共任务。有关 BLOB 对象上的可用方法的完整列表，请查阅 JDBC Javadoc。

使用 JDBC 3.0 支持来更新 BLOB: 在 JDBC 3.0 中，支持更改 LOB 对象。可以将这些更改存储到数据库中的 BLOB 列中。此示例显示了一些借助 JDBC 3.0 中的 BLOB 支持可以执行的任务。

«

示例: BLOB: » 这是有关如何将 BLOB 放入数据库或从数据库检索 BLOB 的示例。

示例: BLOB

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////
// 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.
   }
}
```

示例: 更新 BLOB: 这是有关如何在应用程序中更新 BLOB 的示例。

示例: 更新 BLOB

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////////
// 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.
   }
}
```

示例: 使用 BLOB: 这是有关如何在应用程序中使用 BLOB 的示例。

示例: 使用 BLOB

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
////////////////////////////////////////
// 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.
   }
}
```

编写使用 CLOB 的代码: » 通过 Java(TM) 数据库连接（JDBC）应用程序编程接口（API），可以对数据库 CLOB 和 DBCLOB 列执行许多任务。下列主题简要讨论了这些任务并包括有关如何完成这些任务的示例。

从数据库读取 CLOB 以及将 CLOB 插入数据库: 借助 JDBC API，可使用多种方法从数据库中取出 CLOB 以及将 CLOB 放到数据库中。然而，没有标准化的办法来创建 CLOB 对象。如果数据库已经装满了 CLOB，则这并不是一个问题，但是如果要通过 JDBC 来使用暂存区中的 CLOB，则会引起问题。我们没有为 JDBC API 的 BLOB 和 CLOB 接口定义构造函数，而是提供了用于将 CLOB 作为其它类型来直接放入数据库以及直接从数据库中将它们取出的支持。例如，setCharacterStream 方法可使用 CLOB 类型的数据库列。此示例显示了一些将 CLOB 放入数据库以及从数据库检索它们的常用方法。

使用 CLOB 对象 API: 在 JDBC 中，将 CLOB 定义成接口，各种驱动程序提供了此接口的各种实现。此接口带有一系列可用来与 CLOB 对象交互作用的方法。此示例显示了一些可使用此 API 执行的公共任务。有关 CLOB 对象上的可用方法的完整列表，请查阅 JDBC Javadoc。

使用 JDBC 3.0 支持来更新 CLOB: 在 JDBC 3.0 中，支持更改 LOB 对象。可以将这些更改存储到数据库中的 CLOB 列中。此示例显示了一些借助 JDBC 3.0 中的 CLOB 支持可以执行的任务。

«

示例: CLOB: » 这是有关如何将 CLOB 放入数据库或从数据库检索 CLOB 的示例。

示例: CLOB

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////////
// 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.
   }
}
```

«

示例: 更新 CLOB: » 这是有关如何在应用程序中更新 CLOB 的示例。

示例: 更新 CLOB

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////
// 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.
   }
}
```

示例: 使用 CLOB: » 这是有关如何在应用程序中使用 CLOB 的示例。

示例: 使用 CLOB

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////
// 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.
   }
}
```

编写使用 Datalink 的代码: 您使用 Datalink 的方式取决于您使用的发行版。在 JDBC 3.0 中，支持通过使用 getURL 和 putURL 方法来直接使用 Datalink 列。对于先前的 JDBC 版本，必须将 Datalink 列当作 String 列进行使用。目前，数据库不支持 Datalink 与字符数据类型之间的自动转换。因此，必须在 SQL 语句中执行一些类型强制转换。

此示例显示了一些与使用 Datalink 列相关的基本任务。

«

*示例: **Datalink**:* » 这是有关如何在应用程序中使用 Datalink 的示例。

示例: Datalink

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
////////////////////////////////////////
// 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.
    }
}
```

«

示例: 单值类型: » 这是有关如何使用单值类型的示例。

示例: 单值类型

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
//////////////////////////////////////
// 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

» 最初是将 RowSet 添加到 “Java(TM) 数据库连接（JDBC）2.0 可选包” 中的。与 JDBC 规范的一些大家熟知的接口不同，RowSet 规范设计成更象是一个框架而不是实际的实现。RowSet 接口定义了一组所有 RowSet 都拥有的核心功能。RowSet 实现提供程序具有相当大的自由度来定义在特定问题空间中满足它们的需要所需的功能。

要使用本机 JDBC 驱动程序来实现 Rowset，请查看下列各项:

RowSet 特征

您可以请求 RowSet 要满足特定的属性。公共属性包括结果行集将要支持的接口集。

DB2JdbcRowSet

DB2JdbcRowSet 是已连接的 RowSet，它用作 DB2ResultSet 上的包装器并提供事件处理支持。

DB2CachedRowSet

DB2CachedRowSet 是已断开连接的 RowSet，它允许将 DB2ResultSet 数据存储在对象中。在数据位于对象中之后，就可关闭下层 DB2Connection 对象，并可继续使用 DB2CachedRowSet。请查找下列关于 DB2CachedRowSet 的信息:

- 使用 DB2CachedRowSet
- 创建和植入 DB2CachedRowSet
- 访问 DB2CachedRowSet 数据和游标操纵
- 更改 DB2CachedRowSet 数据并将更改反映回到数据源中
- 其它 DB2CachedRowSet 功能

RowSet 特征: » 您可以请求行集要满足特定的属性。公共属性包括结果行集将要支持的接口集。

RowSet 是 ResultSet: RowSet 接口扩展 ResultSet 接口，这表示 RowSet 有能力执行 ResultSet 可以执行的所有功能。例如，RowSet 可以是可滚动和可更新的。

RowSet 可以与数据库断开连接: RowSet 分为两个类别:

- **已连接**
  在将数据植入已连接的 RowSet 时，它们总是打开与下层数据库的内部连接，并且作为环绕 ResultSet 实现的包装器。
- **已断开连接**
  已断开连接的 RowSet 不需要在所有时候都维护与它们的数据源的连接。可以将已断开连接的 RowSet 与数据库拆离，以各种方式使用它们，然后将它们重新连接至数据库以便镜象对它们所作的任何更改。

RowSet 是 JavaBean 组件: RowSet 具有对基于 JavaBean 事件处理模型的事件处理的支持。它们还具有可以设置的属性。RowSet 可使用这些属性来执行下列操作:

- 建立与数据库的连接。
- 处理 SQL 语句。
- 确定 RowSet 所表示的数据的功能并处理 RowSet 对象的其它内部功能。

RowSet 可序列化: 可以将 RowSet 序列化和取消序列化以允许它们通过网络连接流动以及将它们写至平面文件（即不具有任何字处理或其它结构字符的文本文档）等等。 «

DB2CachedRowSet: » DB2CachedRowSet 对象是断开连接的 RowSet，这表示它可以在不连接至数据库的情况下使用。它的实现紧密遵守 CachedRowSet 的描述。

DB2CachedRowSet 是 ResultSet 中的数据行的容器。DB2CachedRowSet 存放它自己的所有数据，因此，除了对数据库读写数据时显式地维护与数据库的连接之外，其它时候不需要维护该连接。

使用 DB2CachedRowSet

可以使用 DB2CachedRowSet 提供的方法来通过允许若干个人使用同一数据来改进数据库的性能。也可以通过创建不更改的表数据的副本来将公共 ResultSet 交给客户机。

创建和植入 DB2CachedRowSet

通过执行下列任务，了解如何创建 DB2CachedRowSet 以及将数据放入其中:

- 使用 populate 方法
- 使用 DB2CachedRowSet 属性和 DataSource
- 使用 DB2CachedRowSet 属性和 JDBC URL
- 通过使用 setConnection(Connection) 方法来使用现有的数据库连接
- 通过使用 execute(Connection) 方法来使用现有的数据库连接
- 使用 execute(int) 方法来对数据库请求进行分组

访问 DB2CachedRowSet 数据和游标操纵

RowSet 依赖于 ResultSet 方法。对于许多操作，如 DB2CachedRowSet 数据访问和游标移动，使用 ResultSet 与使用 RowSet 在应用程序级别并没有什么不同。

更改 DB2CachedRowSet 数据并将更改反映回到数据源中

DB2CachedRowSet 使用标准 ResultSet 接口所使用的那些方法来更改 RowSet 对象中的数据。DB2CachedRowSet 提供了 acceptChanges 方法，此方法用来将 RowSet 更改反映回到作为数据来源的数据库中。

其它 DB2CachedRowSet 功能

DB2CachedRowSet 类具有一些附加的功能，这些功能使您可以更为灵活地对其进行使用。借助 DB2CachedRowSet 提供的方法，可以执行下列任务:

- 从 DB2CachedRowSet 获取集合
- 创建 RowSet 的副本
- 创建 RowSet 的共享

使用 DB2CachedRowSet:  由于可以将 DB2CachedRowSet 对象断开连接和序列化，所以此对象在并非总是能够实际运行完整 JDBC 驱动程序的环境中（例如在“个人数字助手”（PDA）和启用 Java(TM) 的蜂窝式便携电话上）特别有用。

由于 DB2CachedRowSet 对象包含在内存中并且它的数据总是已知的，所以它可用作应用程序的高度优化形式的可滚动 ResultSet。尽管可滚动 DB2ResultSet 通常会因为它们的随机移动干扰 JDBC 驱动程序的对数据行进行高速缓存的能力而对性能产生负面影响，但 RowSet 没有此问题。

DB2CachedRowSet 上提供了两个用于创建新 RowSet 的方法:

- createCopy 方法创建与复制的 RowSet 完全相同的新 RowSet。
- createShared 方法创建与原始 RowSet 共享同一下层数据的新 RowSet。

可使用 createCopy 方法来将公共 ResultSet 交给客户机。如果表数据未更改，则创建 RowSet 的副本并将其传送给每个客户机要比每一次都对数据库运行查询更有效率。

可以使用 createShared 方法来通过允许若干个人使用同一数据来改进数据库的性能。例如，假定您有一个 Web 站点，当客户连接时，此站点在主页上显示最畅销的 20 件产品。您想定期更新主页上的信息，将每当客户访问主页时都运行查询来获取最畅销物品是不切实际的。通过使用 createShared 方法，可以高效地为每个客户创建“游标”，而不必再次处理查询或在内存中存储大量的信息。在适当的时候，可以再次运行用于查找最畅销产品的查询。新数据可植入用来创建共享游标的 RowSet 以及可使用那些游标的 servlet。

DB2CachedRowSets 提供了延迟处理功能。此功能允许将多个查询请求分组到一起并作为单一请求来对数据库进行处理。这是使用 DB2CachedRowSet 来消除一些计算压力的示例，不然数据库会遭受到这些压力。

由于 RowSet 必须仔细地跟踪对其发生的任何更改以便那些更改可以反映回到数据库中，所以，提供了对用于撤销更改或用于允许您查看已作的所有更改的功能的支持。例如，有一个 showDeleted 方法，此方法可用来告知 RowSet 允许提取已删除的行。另外还有 cancelRowInsert 和 cancelRowDelete 方法，分别用于在进行行插入和删除之后撤销这些操作。

由于 DB2CachedRowSet 对象具有事件处理支持并且它的 toCollection 方法允许将 RowSet 或其一部分转换为 Java 集合，所以 DB2CachedRowSet 对象提供了更好的与其它 Java API 的互操作性。

可以在图形用户界面（GUI）应用程序中使用 DB2CachedRowSet 的事件处理支持来控制显示、在对 RowSet 进行更改时记录关于那些更改的信息或查找关于对除 RowSet 之外的源所作的更改的信息。有关详细信息，参见示例: DB2JdbcRowSet 事件。

有关使用 DB2CachedRowSet 的特定详细信息，参见下列主题:

- 创建和植入 DB2CachedRowSet
- 访问 DB2CachedRowSet 数据和游标操纵
- 更改 DB2CachedRowSet 数据并将更改反映回到数据源中

- 其它 DB2CachedRowSet 功能

由于此项支持对两种类型的 RowSet 能够完全相同地起作用，所以，有关事件模型和事件处理的信息，请查看 DB2JdbcRowSet。 «

创建和植入 DB2CachedRowSet: » 可以采用好几种方法来将数据放到 DB2CachedRowSet 中:

- 使用 populate 方法
- 使用 DB2CachedRowSet 属性和 DataSource
- 使用 DB2CachedRowSet 属性和 JDBC URL
- 通过使用 setConnection(Connection) 方法来使用现有的数据库连接
- 通过使用 execute(Connection) 方法来使用现有的数据库连接
- 使用 execute(int) 方法来对数据库请求进行分组

Warning: Temporary Level 6 Header:

使用 populate 方法: DB2CachedRowSet 有一个 populate 方法，此方法可用来将数据从 DB2ResultSet 对象放到 RowSet 中。以下是这种方法的一个示例。

示例: 使用 populate 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

使用 DB2CachedRowSet 属性和 DataSource: DB2CachedRowSet 具有一些属性，那些属性允许 DB2CachedRowSet 接受 SQL 查询和 DataSource 名称。然后，它们使用 SQL 查询和 DataSource 名称来为它们自己创建数据。以下是这种方法的一个示例。假定对名为 BaseDataSource 的 DataSource 的引用是先前已设置的有效 DataSource。

示例: 使用 DB2CachedRowSet 属性和 DataSource

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

使用 DB2CachedRowSet 属性和 JDBC URL: DB2CachedRowSet 具有一些属性，那些属性允许 DB2CachedRowSet 接受 SQL 查询和 JDBC URL。然后，它们使用该查询和 JDBC URL 来为它们自己创建数据。以下是这种方法的一个示例。

示例: 使用 DB2CachedRowSet 属性和 JDBC URL

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

通过使用 setConnection(Connection) 方法来使用现有的数据库连接: 为了促进重新使用 JDBC Connection 对象，DB2CachedRowSet 提供了一种机制，用于将已建立的 Connection 对象传送至用来填充 RowSet 的 DB2CachedRowSet。如果传入用户提供的 Connection 对象，则 DB2CachedRowSet 在填充其本身之后不断开连接。

示例: 通过使用 setConnection(Connection) 方法来使用现有的数据库连接

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

通过使用 execute(Connection) 方法来使用现有的数据库连接: 为了促进重新使用 JDBC Connection 对象，DB2CachedRowSet 提供了一种机制，用于在调用 execute 方法时将已建立的 Connection 对象传送至 DB2CachedRowSet。如果传入用户提供的 Connection 对象，则 DB2CachedRowSet 在填充其本身之后不断开连接。

示例: 通过使用 execute(Connection) 方法来使用现有的数据库连接

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

使用 execute(int) 方法来对数据库请求进行分组: 为了降低数据库的工作量，DB2CachedRowSet 提供了一种机制，用于将若干个线程的 SQL 语句分组到数据库的一个处理请求中。

示例: 使用 execute(int) 方法来对数据库请求进行分组

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

访问 DB2CachedRowSet 数据和游标操纵: RowSet 依赖于 ResultSet 方法。对于许多操作，如 DB2CachedRowSet 数据访问和游标移动，使用 ResultSet 与使用 RowSet 在应用程序级别并没有什么不同。

访问 DB2CachedRowSet 数据: RowSet 和 ResultSet 以相同的方式访问数据。在下面的示例中，程序使用 JDBC 创建一个表并用各种数据类型对其进行填充。在表就绪之后，创建 DB2CachedRowSet 并用表中的信息对其进行填充。此示例还使用 RowSet 类的各种获取方法。

示例: 访问 DB2CachedRowSet 数据

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
    }
  }
}
```

游标操纵: RowSet 可滚动，它的操作方式与可滚动 ResultSet 很相似。在下面的示例中，程序使用 JDBC 创建一个表并用数据对其进行填充。在表就绪之后，创建 DB2CachedRowSet 对象并用表中的信息对其进行填充。此示例还使用各种游标操纵函数。

示例: 游标操纵

```
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 (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("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());
  }
 }
}
```

«

更改 DB2CachedRowSet 数据并将更改反映回到数据源中: » DB2CachedRowSet 使用标准 ResultSet 接口所使用的那些方法来更改 RowSet 对象中的数据。在应用程序级别，更改 RowSet 的数据与更改 ResultSet 的数据没有区别。DB2CachedRowSet 提供了 acceptChanges 方法，此方法用来将 RowSet 更改反映回到作为数据来源的数据库中。

在 DB2CachedRowSet 中删除、插入和更新行: 可以更新 DB2CachedRowSet。在下面的示例中，程序使用 JDBC 创建一个表并用数据对其进行填充。在表就绪之后，创建 DB2CachedRowSet 并用表中的信息对其进行植入。此示例还使用各个可用来更新 RowSet 的方法并显示了如何使用 showDeleted 属性（即使在删除行之后，此属性也允许应用程序提取那些行）。此外，示例使用 cancelRowInsert 和 cancelRowDelete 方法来允许撤销行的插入或删除。

示例: 在 DB2CachedRowSet 中删除、插入和更新行

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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());
    }
  }
}
```

将 DB2CachedRowSet 更改反映回到下层数据库: 在对 DB2CachedRowSet 作更改之后，只要 RowSet 对象存在这些更改就会存在。也就是说，对断开连接的 RowSet 作更改不会影响数据库。要将 RowSet 的更改反映到下层数据库中，应使用 acceptChanges 方法。此方法告诉已断开连接的 RowSet 重新建立与数据库的连接并尝试对下层数据库进行对 RowSet 已作的更改。如果因为与创建 RowSet 之后发生的其它数据库更改相冲突而不能安全地对数据库进行更改，则将抛出异常并回滚事务。

示例: 将 DB2CachedRowSet 更改反映回到下层数据库

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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());
    }
  }
}
```

«

其它 DB2CachedRowSet 功能: » 除了如若干个示例所显示的那样象 ResultSet 一样工作之外，DB2CachedRowSet 类还具有一些附加的功能，这些功能使您可以更为灵活地对其进行使用。提供了用于调整整个 “Java(TM) 数据库连接” （JDBC）RowSet 或仅仅将其一部分调整为 JDBC 集合的方法。而且，由于它们具有断开连接这一本质，DB2CachedRowSet 与 ResultSet 之间不具有严格的一对一关系。

借助 DB2CachedRowSet 提供的方法，可以执行下列任务:

- 从 DB2CachedRowSet 获取集合
- 创建 RowSet 的副本
- 创建 RowSet 的共享

从 DB2CachedRowSet 获取集合: 有三个方法从 DB2CachedRowSet 对象返回某种形式的集合。它们是:

- **toCollection** 返回向量（即，对每一列有一个项）的 ArrayList（即，对每一行有一个项）。
- **toCollection(int columnIndex)** 返回一个向量，该向量包含每一行的给定列中的值。
- **getColumn(int columnIndex)** 返回一个数组，该数组包含给定列的每一列的值。

toCollection(int columnIndex) 与 getColumn(int columnIndex) 之间的主要区别在于 getColumn 方法可返回原语类型的数组。因此，如果 columnIndex 表示带有整数数据的列，则将返回整数数组而不是包含 java.lang.Integer 对象的数组。

以下示例显示了如何使用这些方法。

示例: 从 DB2CachedRowSet 获取集合

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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());
    }
  }
}
```

创建 RowSet 的副本: createCopy 方法创建 DB2CachedRowSet 的副本。将复制所有与 RowSet 相关联的数据以及所有控制结构、属性和状态标志。

以下示例显示了如何使用此方法。

示例: 创建 RowSet 的副本

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
    }

  }

}
```

创建 RowSet 的共享: createShared 方法创建新的带有高级状态信息的 RowSet 对象，并允许两个 RowSet 对象共享同一下层物理数据。

以下示例显示了如何使用此方法。

示例: 创建 RowSet 的共享

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 是已连接的 RowSet，这表示它只能与下层 Connection 对象、PreparedStatement 对象或 ResultSet 对象的支持配合使用。它的实现紧密遵守 JdbcRowSet 的描述。

使用 DB2JdbcRowSet: 由于 DB2JdbcRowSet 对象支持 "Java(TM) 数据库连接"（JDBC）3.0 规范中对所有 RowSet 描述的事件，所以它可用作本地数据库与其它必须得到数据库数据更改通知的对象之间的中间对象。

作为一个示例，假定您正在一个特定的环境中工作，在这个环境中，有一个主数据库和若干个使用无线协议来连接至该数据库的 "个人数字助手"（PDA）。可使用 DB2JdbcRowSet 对象来移至一行并通过使用在服务器上运行的主控应用程序来更新它。行更新将导致 RowSet 组件生成事件。如果正在运行负责发送出对 PDA 的更新的服务，则它将它自己注册为 RowSet 的 "侦听器"。每次它接收到 RowSet 事件时，它可以生成适当的更新并将其发送出至无线设备。

有关更多信息，请参考示例: DB2JdbcRowSet 事件。

创建 JDBCRowSet: 提供了若干个用于创建 DB2JDBCRowSet 对象的方法。下面对每个方法作了概括。

使用 DB2JdbcRowSet 属性和 DataSource: DB2JdbcRowSet 具有接受 SQL 查询和 DataSource 名称的属性。于是，DB2JdbcRowSet 就可供使用了。以下是这种方法的一个示例。假定对名为 BaseDataSource 的 DataSource 的引用是先前已设置的有效 DataSource。

示例: 使用 DB2JdbcRowSet 属性和 DataSource

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

使用 DB2JdbcRowSet 属性和 JDBC URL: DB2JdbcRowSet 具有接受 SQL 查询和 JDBC URL 的属性。于是，DB2JdbcRowSet 就可供使用了。以下是这种方法的一个示例:

示例: 使用 DB2JdbcRowSet 属性和 JDBC URL

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

通过使用 setConnection(Connection) 方法来使用现有的数据库连接: 为了促进重复使用 JDBC Connection 对象，DB2JdbcRowSet 允许将已建立的连接传送至 DB2JdbcRowSet。DB2JdbcRowSet 使用此连接来准备它自己，以便在调用 execute 方法时进行使用。

示例: 使用 setConnection 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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();
```

访问数据和游标移动: 下层 ResultSet 对象负责处理通过 DB2JdbcRowSet 来操纵游标位置和访问数据库数据。可使用 ResultSet 对象完成的任务也适用于 DB2JdbcRowSet 对象。

更改数据以及将更改反映到下层数据库: 对通过 DB2JdbcRowSet 更新数据库的支持完全是由下层 ResultSet 对象处理的。可使用 ResultSet 对象完成的任务也适用于 DB2JdbcRowSet 对象。 «

DB2JdbcRowSet 事件: » 所有 RowSet 实现都支持其它组件感兴趣的情况的事件处理。此项支持允许应用程序组件在发生事件时相互“交谈”。例如，通过 RowSet 更新数据库行可能会导致“图形用户界面”（GUI）表显示为更新它自己。

在以下示例中，main 方法执行对 RowSet 的更新，这是核心应用程序。侦听器是断开连接的现场客户机所使用的无线服务器的一部分。将商业的这两方面联系到一起而不将两个过程的代码相混合是有可能的。尽管 RowSet 的事件支持主要是为使用数据库数据更新 GUI 而设计的，但它对于此类型的应用程序问题也能非常好地起作用。

示例: DB2JdbcRowSet 事件

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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 JDBC 驱动程序的性能提示

我们已将 IBM Developer Kit for Java(TM) JDBC 驱动程序设计成为高性能的用于使用数据库的 Java 接口。然而，要获得可能的最佳性能，您需要将应用程序构建为充分利用 JDBC 驱动程序必须提供的强度。我们认为下列提示是良好的 JDBC 编程实践。它们中的大多数并非只能用于本机 JDBC 驱动程序。因此，即使与除本机 JDBC 驱动程序之外的 JDBC 驱动程序配合使用，根据这些准则编写的应用程序也会有不错的表现。

- 避免 SELECT * SQL 查询
- 使用 getXXX(int) 而不是 getXXX(String)
- 避免对 Java 原语类型的 getObject 调用
- 使用 PreparedStatement 来代替 Statement
- 避免成本高昂的 DatabaseMetaData 调用
- 使用应用程序的正确提交级别
- 考虑采用 Unicode 来存储数据
- 使用存储过程
- 使用 BigInt 而不是 Numeric/Decimal
- 完成使用 JDBC 资源后显式地将它们关闭
- 使用连接池
- 考虑使用 PreparedStatement 合用
- 使用高效的 SQL

避免 SELECT * SQL 查询

SELECT * FROM... 是在 SQL 中陈述查询的常见方法。然而，通常不需要查询所有字段。对于每个将要返回的列，JDBC 驱动程序必须执行附加的工作来绑定和返回行。即使应用程序从不使用特定的列，JDBC 驱动程序也必须了解该列并且必须保留空间来供其使用。如果不使用的表列比较少，则开销并不显著。然而，对于大量未使用的列，开销可能非常显著。更好的解决方案是个别地列示应用程序所感兴趣的列，如:

```
SELECT COL1, COL2, COL3 FROM...
```

使用 getXXX(int) 而不是 getXXX(String)

请使用接收数值的 ResultSet getXXX 方法而不是使用接收列名的版本。尽管自由地使用列名而不是数字常量似乎是一个优点，但数据库本身只知道如何处理列索引。因此，JDBC 驱动程序必须先对您调用的每个带有列名的 getXXX 方法进行解析，然后才能将其传送给数据库。由于通常在可能要运行数百万次的循环中调用 getXXX 方法，所以这一微小的开销可能会快速地累积。

避免对 Java 原语类型的 getObject 调用

当从数据库获取具有原语类型（int、long 和 float 等等）的值时，使用特定于该原语类型的 get 方法（getInt、getLong 和 getFloat）比使用 getObject 要快。getObject 调用先执行原语类型的 get 的工作，然后创建要返回的对象。这通常是在循环中完成的，从而有可能会创建数百万个具有短暂寿命的对象。对基元命令使用 getObject 还具有需要频繁激活垃圾收集器这一缺点，从而进一步降低性能。

使用 PreparedStatement 来代替 Statement

如果正在编写需要多次使用的 SQL 语句，则它作为 PreparedStatement 的执行效果要比作为 Statement 对象的执行效果好。每次运行语句时，都会经过一个由两个步骤组成的过程: 准备语句，然后处理语句。使用已准备的语句时，只在构造语句时才准备该语句，而不会在每次运行该语句时这样做。尽管 PreparedStatement 的执行

速度比 Statement 快这一点是公认的，但程序员经常忽略此项优点。由于 PreparedStatement 能够大幅提高性能，所以明智的做法是在应用程序中尽可能地使用它们（参见 PreparedStatement 合用）。

避免 DatabaseMetaData 调用

您要知道，某些 DatabaseMetaData 调用的成本可能十分高昂。特别是，getBestRowIdentifier、getCrossReference、getExportedKeys 和 getImportedKeys 方法的成本可能非常高昂。某些 DataBaseMetaData 调用涉及基于系统级表的复杂连接条件。敬请仅当需要它们的信息时才使用它们，而不是仅仅为了方便而使用它们。

使用应用程序的正确提交级别

JDBC 提供了若干个提交级别，这些提交级别确定对系统执行的多个事务是如何相互影响的（有关更多的详细信息，参见事务）。缺省情况是使用最低的提交级别。这表示事务可通过提交边界查看其它每个事务的一些工作。这就引入了特定数据库反常的可能性。一些程序员会提高提交级别，从而使他们不必担心会发生这些反常情况。您要知道，提交级别越高，就会导致数据库挂起在越多的短暂锁定上。这将限制系统可以具有的并行度，并严重拖慢某些应用程序的性能。通常，我们通过在一开始对应用程序进行精心设计来杜绝反常情况的发生。请花费点时间来理解您正在尝试完成的任务并将事务隔离级别限制为可以安全使用的最低级别。

考虑采用 Unicode 来存储数据

Java 要求它使用的所有字符数据（String）都采用 Unicode。因此，在将任何不包含 Unicode 数据的表放入数据库以及从数据库中检索出该表时，都要求 JDBC 驱动程序来回转换数据。如果表已采用 Unicode，则 JDBC 驱动程序就不需要转换数据，从而可以更快地将数据放入数据库。注意，Unicode 数据不能与不知道如何处理 Unicode 的非 Java 应用程序配合使用。并且请记住，由于不转换非字符数据，所以这种数据的执行速度不会有任何提高。另一个注意事项是，采用 Unicode 存储的数据所占用的空间是单字节数据的两倍。然而，如果有很多要读取许多次的字符列，则通过采用 Unicode 存储数据所获得的性能增益就会很显著。

使用存储过程

Java 支持使用存储过程。存储过程通过允许 JDBC 驱动程序运行静态 SQL 而不是动态 SQL 来提高执行速度。请不要为程序中运行的每一个个别 SQL 语句创建存储过程。然而，请尽可能地创建运行一组 SQL 语句的存储过程。

使用 BigInt 而不是 Numeric 或 Decimal

请不要使用标度为 0 的 Numeric 或 Decimal 字段，而是使用 BigInt 数据类型。BigInt 将直接转换为 Java 原语类型 Long，而 Numeric 或 Decimal 数据类型将转换成 String 或 BigDecimal 对象。如 避免对 Java 原语类型的 getObject 调用所述，使用基元数据类型比使用要求创建对象的类型更为可取。

完成使用 JDBC 资源后显式地将它们关闭

当不再需要 ResultSet、Statement 和 Connection 时，应用程序应显式地将它们关闭。这样就能够以最有效率的方式清理资源并可以提高性能。此外，未显式关闭的数据库资源可能会导致资源泄漏以及无必要地长时间挂起数据库锁。这会引起应用程序故障或降低应用程序并行度。

使用连接池

连接池是一种策略，通过这种策略，对多个用户重复使用 JDBC Connection 对象，而不是让每个用户请求都创建它自己的 Connection 对象。Connection 对象的创建成本十分高昂。并不应该让每个用户都创建新的 Connection 对象，而是，应该在性能比较关键的应用程序中共享 Connection 对象的池。许多产品（如 WebSphere）提供了

Connection 合用支持，在使用这些合用支持时，用户方只需执行很少的附加工作。即使您不想使用具有连接池支持的产品，或者希望构建自己的连接池以便更好地控制池的工作和执行方式，进行起来也相当容易。

考虑使用 PreparedStatement 合用

Statement 合用的工作方式与 Connection 合用类似。并不是仅仅将 Connection 放到池中，而是将包含 Connection 和 PreparedStatement 的对象放到池中。然后，检索该对象并访问您要使用的特定语句。这可以显著提高性能。

使用高效的 SQL

由于 JDBC 是基于 SQL 构建的，所以几乎任何有利于提高 SQL 效率的措施对提高 JDBC 的效率也有利。因而，优化的查询、明智选择的索引以及良好 SQL 设计的其它方面都会使 JDBC 受益。

使用 IBM Developer Kit for Java DB2 SQLJ 支持来访问数据库

DB2 的 “Java(TM) 结构化查询语言”（SQLJ）支持基于 SQLJ ANSI 标准。DB2 SQLJ 支持包含在 IBM Developer Kit for Java 中。DB2 SQLJ 支持允许您创建、构建及运行 Java 嵌入式 SQL 应用程序。

IBM Developer Kit for Java 提供的 SQLJ 支持包括 SQLJ 运行时类，它可以从 /QIBM/ProdData/Java400/ext/runtime.zip 中获得。有关 SQLJ 运行时类的更多信息，请参考 www.sqlj.org 上的 “实现” 中提供的 “运行时 API” 文档。

SQLJ 工具

IBM Developer Kit for Java 提供的 SQLJ 支持还包括下列工具:

- SQLJ 转换程序（sqlj），它使用 Java 源语句来替换 SQLJ 程序中的嵌入式 SQL 语句并生成序列化的概要文件，此概要文件包含有关 SQLJ 程序中的操作的信息。
- DB2 SQLJ 概要文件定制程序（db2profc），它对存储在已生成的概要文件中的 SQL 语句进行预编译，并在 DB2 数据库中生成包。
- DB2 SQLJ 概要文件打印程序（db2profp），它以纯文本格式打印 DB2 定制概要文件的内容。
- SQLJ 概要文件审查程序安装器（profdb），它用来在现有二进制概要文件集中安装和卸载调试类审查程序。
- SQLJ 概要文件转换工具（profconv），它将序列化的概要文件实例转换为 Java 类格式。

注意: 这些工具必须在 Qshell Interpreter 中运行。

DB2 SQLJ 限制

当使用 SQLJ 来创建 DB2 应用程序时，您应了解下列限制:

- DB2 SQLJ 支持符合有关发出 SQL 语句的标准 “DB2 通用数据库” 限制。
- DB2 SQLJ 概要文件定制程序只应该对与本地数据库的连接相关联的概要文件运行。
- “SQLJ 参考实现” 需要 JDK 1.1 或更高版本。有关运行多个版本的 Java Development Kit 的更多信息，参见支持多个 Java Development Kit（JDK）。

有关在 Java 应用程序中使用 SQL 的信息，参见 Java 应用程序中的嵌入式 SQL 语句和编译和运行 SQLJ 程序。

Java 结构化查询语言概要文件

在转换 SQLJ 源文件时，“SQLJ 转换程序”（sqlj）会生成概要文件。概要文件是序列化的二进制文件。这就是为什么这些文件带有扩展名 .ser 的原因。这些文件包含相关联的 SQLJ 源文件中的 SQL 语句。

要根据 SQLJ 源代码来生成概要文件，请对 .sqlj 文件运行 SQLJ 转换程序 sqlj。

有关更多信息，参见编译和运行 SQLJ 程序。

Java 结构化查询语言（SQLJ）转换程序（sqlj）

SQLJ 转换程序（sqlj）生成序列化的概要文件，此概要文件包含有关 SQLJ 程序中的 SQL 操作的信息。SQLJ 转换程序使用 /QIBM/ProdData/Java400/ext/translator.zip 文件。

有关 sqlj 命令行选项的更多信息，请参考 www.sqlj.org  中的“实现”中提供的“SQLJ 用户指南与参考”。

使用 DB2 SQLJ 概要文件定制程序 db2profc 来预编译概要文件中的 SQL 语句

可以使用“DB2 SQLJ 概要文件定制程序”（db2profc）来使 Java(TM) 应用程序更有效地使用数据库。

“DB2 SQLJ 概要文件定制程序”执行下列操作:

- 对存储在概要文件中的 SQL 语句进行预编译，并在 DB2 数据库中生成包。
- 通过将 SQL 语句替换为对所创建的包中的关联语句的引用来定制 SQLJ 概要文件。

要对概要文件中的 SQL 语句进行预编译，请在 Qshell 命令提示处输入以下命令:

```
db2profc MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 是要预编译的概要文件的名称。

“DB2 SQLJ 概要文件定制程序”的用法及语法

db2profc[*options*] <*SQLJ_profile_name*>

其中，*SQLJ_profile_name* 是要打印的概要文件的名称，而 *options* 是您所要的选项列表。

db2profp 的可用选项如下:

- -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*>

以下是这些选项的描述:

-URL=<*JDBC_URL***>**

其中，*JDBC_URL* 是 JDBC 连接的 URL。URL 的语法是:

```
"jdbc:db2:systemName"
```

有关更多信息，参见使用 IBM Developer Kit for Java JDBC 驱动程序来访问 iSeries 数据库。

-user=<*username*>

其中，*username* 是您的用户名。缺省值是已对本地连接注册的当前用户的用户标识。

-password=<*password*>

其中，*password* 是密码。缺省值是已对本地连接注册的当前用户的密码。

-package=<*library name/package name*>

其中，*library name* 是存放包的库，而 *package name* 是要生成的包的名称。缺省库名是 QUSRSYS。缺省包名是根据概要文件的名称生成的。包名的最大长度为 10 个字符。因为 SQLJ 概要文件名的长度总是超过 10 个字符，所以构造的缺省包名将与概要文件名不同。通过将概要文件名的前几个字母与概要文件键号并置来构造缺省包名。如果概要文件键号的长度超过 10 个字符，则将概要文件键号的后 10 个字符用作缺省包名。例如，下表显示了一些概要文件名及其缺省包名:

概要文件名	缺省包名
App_SJProfile0	App_SJPro0
App_SJProfile01234	App_S01234
App_SJProfile012345678	A012345678
App_SJProfile01234567891	1234567891

-commitctrl=<*commitment_control*>

其中，*commitment_control* 是您所要的提交控制的级别。提交控制可以是下列字符值中的任何一个:

值	定义
C	*CHG。脏读取、不可重复读取和幻象读取都是可能的。
S	*CS。脏读取是不可能的，但不可重复读取和幻象读取是可能的。
A	*ALL。脏读取和不可重复读取是不可能的，但幻象读取是可能的。
N	*NONE。脏读取、不可重复读取和幻象读取都是不可能的。这是缺省值。

-datefmt=<*date_format*>

其中，*date_format* 是您所要的日期格式类型。日期格式可以是下列值中的任何一个:

值	定义
USA	IBM 美国标准（mm.dd.yyyy,hh:mm a.m., hh:mm p.m.）
ISO	国际标准化组织（yyyy-mm-dd, hh.mm.ss）, 这是缺省值。
EUR	IBM 欧洲标准（dd.mm.yyyy, hh.mm.ss）
JIS	日本工业标准公元历（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	儒略历（yy/ddd）

当访问日期结果列时，将使用到日期格式。所有输出日期字段都以指定格式返回。对于输入日期字符串，指定的值用来确定日期是否是以有效格式指定的。缺省值为 ISO。

-datesep=<*date_separator*>

其中，*date_separator* 是您想要使用的分隔符的类型。当访问日期结果列时，将使用到日期分隔符。日

期分隔符可以是下列值中的任何一个:

值	定义
/	使用斜杠。
.	使用句点。
,	使用逗号。
-	使用连字符。这是缺省值。
空白	使用空格。

-timefmt=<*time_format***>**

其中，*time_format* 是您想要用来显示时间字段的格式。当访问时间结果列时，将使用到时间格式。对于输入时间字符串，指定的值用来确定时间是否是以有效格式指定的。时间格式可以是下列值中的任何一个:

值	定义
USA	IBM 美国标准（mm.dd.yyyy,hh:mm a.m., hh:mm p.m.）
ISO	国际标准化组织（yyyy-mm-dd, hh.mm.ss），这是缺省值。
EUR	IBM 欧洲标准（dd.mm.yyyy, hh.mm.ss）
JIS	日本工业标准公元历（yyyy-mm-dd, hh:mm:ss）
HMS	小时／分钟／秒（hh:mm:ss）

-timesep=<*time_separator***>**

其中，*time_separator* 是要用来访问时间结果列的字符。时间分隔符可以是下列值中的任何一个:

值	定义
:	使用冒号。
.	使用句点。这是缺省值。
,	使用逗号。
空白	使用空格。

-decimalpt=<*decimal_point***>**

其中，*decimal_point* 是要使用的小数点。小数点用于 SQL 语句中的数字常量。小数点可以是下列值中的任何一个:

值	定义
.	使用句点。这是缺省值。
,	使用逗号。

-stmtCCSID=<*CCSID***>**

其中，*CCSID* 是所准备的要放入包的 SQL 语句的编码字符集标识符。作业在定制期间的值就是缺省值。

-sorttbl=<*library_name/sort_sequence_table_name***>**

其中，*library_name/sort_sequence_table_name* 是要使用的排序顺序表的位置和表名。排序顺序表用于 SQL 语句中的字符串比较。库名和排序顺序表名都限长 10 个字符。缺省值取自定制期间的作业。

-langID=<*language_identifier*>

其中，*language_identifier* 是要使用的语言标识符。语言标识符的缺省值取自定制期间的当前作业。语言标识符与排序顺序表配合使用。

有关上述任何字段的更详细信息，参见 DB2 for iSeries SQL Programming Concepts（SC41-5611）  。

打印 DB2 SQLJ 概要文件（db2profp 和 profp）的内容

“DB2 SQLJ 概要文件打印程序”（db2profp）以纯文本格式打印 DB2 定制概要文件的内容。“概要文件打印程序”（profp）以纯文本格式打印由 SQLJ 转换程序生成的概要文件的内容。

要以纯文本格式打印由 SQLJ 转换程序生成的概要文件的内容，请按如下方式使用 profp 实用程序:

```
profp MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 是要打印的概要文件的名称。

要以纯文本格式打印概要文件的 DB2 定制版本的内容，请按如下方式使用 db2profp 实用程序:

```
db2profp MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 是要打印的概要文件的名称。

注意: 如果对未定制的概要文件运行 db2profp，则它会告诉您概要文件未经定制。如果对已定制的概要文件运行 profp，则它将显示概要文件的内容，而不是定制内容。

“DB2 SQLJ 概要文件打印程序”的用法及语法:

```
db2profp [options] <SQLJ_profile_name>
```

其中，*SQLJ_profile_name* 是要打印的概要文件的名称，而 *options* 是您所要的选项列表。

db2profp 的可用选项如下:

-URL=<*JDBC_URL*>

其中 *JDBC_URL* 是要连接至的 URL。有关更多信息，参见使用 IBM Developer Kit for Java JDBC 驱动程序来访问 iSeries 数据库。

-user=<*username*>

其中，*username* 是用户概要文件的用户名。

-password=<*password*>

其中，*password* 是用户概要文件的密码。

SQLJ 概要文件审查程序安装器（profdb）

SQLJ 概要文件审查程序安装器（profdb）用来安装和卸装调试类审查程序。将把调试类审查程序安装到现有的二进制概要文件集内。在安装调试类审查程序之后，便将应用程序运行时期间进行的所有 RTStatement 和 RTResultSet 调用记录下来。可以将它们记录到文件或标准输出。然后，可以检查这些作业记录来验证应用程序的行为和跟踪应用程序的错误。注意，只审查在运行时对下层 RTStatement 和 RTResultSetcall 接口所作的调用。

要安装调试类审查程序，请在 Qshell 命令提示处输入以下命令:

```
profdb MyClass_SJProfile0.ser
```

其中，*MyClass_SJProfile0.ser* 是“SQLJ 转换程序”生成的概要文件的名称。

要卸载调试类审查程序，请在 Qshell 命令提示处输入以下命令:

```
profdb -Cuninstall MyClass_SJProfile.ser
```

其中，*MyClass_SJProfile0.ser* 是 “SQLJ 转换程序” 生成的概要文件的名称。

有关 profdb 命令行选项的更多信息，请访问 www.sqlj.org，选择 “实现” 类别并转至 “运行时 API 文档” 中的 sqlj.runtime.profile.util.AuditorInstaller 类。

使用 SQLJ 概要文件转换工具（profconv）来将序列化的概要文件实例转换为 Java 类格式

SQLJ 概要文件转换工具（profconv）将序列化的概要文件实例转换为 Java(TM) 类格式。因为某些浏览器不支持从与 applet 相关联的资源文件装入序列化的对象，所以您需要 profconv 工具。请运行 profconv 实用程序来执行转换。

要运行 profconv 实用程序，请在 Qshell 命令行上输入以下命令:

```
profconv MyApp_SJProfile0.ser
```

其中，*MyApp_SJProfile0.ser* 是您想要转换的概要文件实例的名称。

profconv 工具调用 sqlj -ser2class。请查看 sqlj 以了解命令行选项。

Java 应用程序中的嵌入式 SQL 语句

SQLJ 中的静态 SQL 语句以 SQLJ 子句形式出现。SQLJ 子句以 #sql 开始，并以分号（;）字符结束。

在 Java(TM) 应用程序中创建任何 SQLJ 子句之前，需导入下列包:

- import java.sql.*;
- import sqlj.runtime.*;
- import sqlj.runtime.ref.*;

最简单的 SQLJ 子句是可处理的子句，由 #sql 记号以及随后的 SQL 语句（括在花括号中）组成。例如，以下 SQLJ 子句可以出现在 Java 语句可以合法出现的任何地方:

```
#sql { DELETE FROM TAB };
```

上一示例删除名为 TAB 的表中的所有行。

注意: 有关编译和运行 SQLJ 应用程序的信息，参见编译和运行 SQLJ 程序。

在 SQLJ 处理子句中，在花括号中出现的记号是 SQL 记号或主机变量。所有主机变量都通过冒号（:）字符进行区别。SQL 记号从不出现在 SQLJ 处理子句的花括号外面。例如，以下 Java 方法将其自变量插入到 SQL 表中:

```
public void insertIntoTAB1 (int x, String y, float z) throws SQLException
{
    #sql { INSERT INTO TAB1 VALUES (:x, :y, :z) };
}
```

方法主体由包含主机变量 x、y 和 z 的 SQLJ 处理子句组成。有关主机变量的更多信息，参见 SQLJ 中的主机变量。

通常，SQL 记号不区分大小写（通过双引号定界的标识符除外），可以书写成大写、小写或混合大小写。然而，*Java* 记号却区分大小写。为了使示例更为清晰，不区分大小写的 SQL 记号是大写的，而 Java 记号是小写的或混合大小写的。贯穿本主题，小写 `null` 都用来表示 Java 的“空”值，而大写 `NULL` 用来表示 SQL 的“空”值。

SQLJ 程序中可能会出现下列类型的 SQL 结构:

- 查询
  例如，SELECT 语句和表达式。
- “SQL 数据更改”语句（DML）
  例如，INSERT、UPDATE 和 DELETE。
- 数据语句
  例如，FETCH 和 SELECT..INTO。
- “事务控制”语句
  例如，COMMIT 和 ROLLBACK，等等。
- “数据定义语言”（DDL，也称为“模式操作语言”）语句
  例如，CREATE、DROP 和 ALTER。
- 存储过程调用
  例如，CALL MYPROC(:x, :y, :z)
- 存储函数调用
  例如，VALUES(MYFUN(:x))

有关嵌入式 SQLJ 的示例，参见示例: Java 应用程序中的嵌入式 SQL 语句。

Java 结构化查询语言中的主机变量: 嵌入式 SQL 语句的自变量是通过主机变量传送的。主机变量是主机语言的变量，它们可以出现在 SQL 语句中。主机变量最多可以有三个部分:

- 冒号（:）前缀。
- 作为参数、变量或字段的 Java 标识符的 Java(TM) 主机变量。
- 可选的参数方式标识符。

  此方式标识符可是下列其中一项:
  IN、OUT 或 INOUT。

Java 标识符的赋值对 Java 程序没有副作用，所以它可以为替换 SQLJ 子句而在生成的 Java 代码中出现多次。

以下查询包含主机变量 `:x`。这个主机变量是在包含该查询的作用域中可见的 Java 变量、字段或参数 `x`。

```
SELECT COL1, COL2 FROM TABLE1 WHERE :x > COL3
```

示例: Java 应用程序中的嵌入式 SQL 语句: 以下示例 SQLJ 应用程序（App.sqlj）使用静态 SQL 来在 DB2 样本数据库的 EMPLOYEE 表中检索和更新数据。

示例: Java(TM) 应用程序中的嵌入式 SQL 语句:

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
    }
  }
}
```

1. **声明迭代器。**本节声明了两种类型的迭代器:

   App_Cursor1
   声明列数据类型和名称，并根据列名返回列值（列的命名绑定）。

   App_Cursor2
   声明列数据类型，并按列位置返回列值（列的位置绑定）。
2. **初始化迭代器。**迭代器对象 cursor1 使用查询结果进行初始化。此查询将结果存储在 cursor1 中。
3. **使迭代器前进至下一行。**如果没有更多的行可供检索，则 cursor1.next() 方法返回布尔值 false。
4. **移动数据。**命名访问者方法 empno() 返回当前行上名为 empno 的列的值。命名访问者方法 firstnme() 返回当前行上名为 firstnme 的列的值。
5. **将数据查询（SELECT）到主机变量中。**SELECT 语句将表中的行传送到主机变量 count1 中。
6. **初始化迭代器。**迭代器对象 cursor2 使用查询结果进行初始化。此查询将结果存储在 cursor2 中。
7. **检索数据。**FETCH 语句将 ByPo 游标中声明的首列的当前值从结果表返回到主机变量 str2 中。
8. **检查 FETCH..INTO 语句成功与否。**如果迭代器未定位在行上（即上一次提取行的尝试失败），则 endFetch() 方法返回布尔值 true。如果上一次提取行成功，则 endFetch() 方法返回 false。当调用 next() 方法时，DB2 便尝试提取行。FETCH...INTO 语句隐式地调用 next() 方法。
9. **关闭迭代器。**close() 方法释放被迭代器挂起的任何资源。您应显式关闭迭代器才能确保及时释放系统资源。

有关本示例的背景信息，参见 Java 应用程序中的嵌入式 SQL 语句。

编译和运行 SQLJ 程序

如果 Java(TM) 程序带有嵌入式 SQLJ 语句，则需要遵循特殊的过程才能编译和运行该程序。

注意: 在开始之前，请将 CLASSPATH 设置为包含下列各项:

- /QIBM/ProdData/Os400/Java400/ext/sqlj_classes.jar

- /QIBM/ProdData/Os400/Java400/ext/translator.zip
- /QIBM/ProdData/Os400/Java400/ext/runtime.zip

要编译和运行 Java 结构化查询语言（SQLJ）程序，请遵循下面这些步骤:

1. 对带有嵌入式 SQL 的 Java 源代码 使用 SQLJ 转换程序（sqlj），来生成 Java 源代码和相关联的概要文件。将对每个连接生成一个概要文件。

   例如，输入以下命令:

   ```
   sqlj MyClass.sqlj
   ```

   其中，*MyClass.sqlj* 是 SQLJ 文件的名称。

   在本示例中，SQLJ 转换程序将生成 MyClass.java 源代码文件和任何相关联的概要文件。相关联的概要文件名为 MyClass_SJProfile0.ser、MyClass_SJProfile1.ser 以及 MyClass_SJProfile2.ser，等等。

   注意: 除非使用 -compile=false 子句显式关闭编译选项，否则 SQLJ 转换程序将自动把转换后的 Java 源代码编译成类文件。
2. 使用 “SQLJ 概要文件定制程序” 工具（db2profc）来对生成的概要文件安装 “DB2 SQLJ 定制程序”，并在本地系统上创建 DB2 包。

   例如，输入命令:

   ```
   db2profc MyClass_SJProfile0.ser
   ```

   其中，*MyClass_SJProfile0.ser* 是要对其运行 “DB2 SQLJ 定制程序” 的概要文件的名称。

   注意: 此步骤是可选的，但建议您执行此步骤，以提高运行时性能。
3. 象运行任何其它 Java 类文件一样 运行 Java 类文件。

   例如，输入命令:

   ```
   java MyClass
   ```

   其中，*MyClass* 是 Java 类文件的名称。

Java SQL 例程

iSeries 服务器提供了从 SQL 语句和程序访问 Java(TM) 程序的能力。这可使用 Java 存储过程和 Java 用户定义函数（UDF）来完成。iSeries 服务器支持用于调用 Java 存储过程和 Java UDF 的 DB2 和 SQLJ 约定。Java 存储过程和 Java UDF 都可使用存储在 JAR 文件中的 Java 类。iSeries 服务器使用 *SQLJ 第 1 部分*标准定义的存储过程来对数据库注册 JAR 文件。

要从 SQL 语句和程序访问 Java 应用程序，请查看下列各项:

使用 Java SQL 例程

执行下列步骤来使用 Java SQL 例程:

- 为例程编写 Java 方法。
- 编译 Java 类。
- 使数据库所使用的 Java 虚拟机可访问经编译的类。
- 对数据库注册例程。
- 使用 Java SQL 过程。

Java 存储过程

当使用 Java 来编写存储过程时，可使用下列可能的参数传送样式:

- JAVA 参数样式
- DB2GENERAL 参数样式

Java 用户定义标量函数

Java 标量函数从 Java 程序返回一个值给数据库。与 Java 存储过程相似，Java 标量函数使用两种参数样式（JAVA 和 DB2GENERAL）之一。

Java 用户定义表函数

DB2 提供了让函数返回表的能力。这对于以表格式将数据库外的信息透露给数据库非常有用。

用于操纵 JAR 文件的 SQLJ 过程

Java 存储过程和 Java UDF 都可使用存储在 Java JAR 文件中的 Java 类。请查找下列关于操纵 JAR 文件的 SQLJ 过程的信息:

- SQLJ.INSTALL_JAR
- SQLJ.REMOVE_JAR
- SQLJ.REPLACE_JAR
- SQLJ.UPDATEJARINFO
- SQLJ.RECOVERJAR

Java 存储过程和 UDF 的参数传送约定

本节描述如何在 Java 存储过程和 UDF 中表示 SQL 数据类型。

使用 Java SQL 例程

可以从 SQL 语句和程序访问 Java(TM) 程序。这可使用 Java 存储过程和 Java 用户定义函数（UDF）来完成。

要使用 Java SQL 例程，必须执行下列步骤:

1. 为例程编写 Java 方法。

   Java SQL 例程从 SQL 中处理 Java 方法。必须使用 DB2(R) 或 SQLJ 参数传送约定来编写此方法。有关编码 Java SQL 例程所使用的方法的更多信息，参见 Java 存储过程、Java用户定义函数和 Java 用户定义表函数。

2. 编译 Java 类。

   可以在不执行任何附加设置的情况下编译使用 Java 参数样式编写的 Java SQL 例程。然而，使用 DB2GENERAL 参数样式的 Java SQL 例程必须扩展 com.ibm.db2.app.UDF 类或 com.ibm.db2.app.StoredProc 类。这些类包含在 JAR 文件 /QIBM/ProdData/Java400/ext/db2routines_classes.jar 中。当使用 javac 来编译这些例程时，这个 JAR 文件必须存在于 CLASSPATH 中。例如，以下命令用于编译包含使用 DB2GENERAL 参数样式的例程的 Java 源文件:

   ```
   javac -DCLASSPATH=/QIBM/ProdData/Java400/ext/db2routines_classes.jar
   source.java
   ```

3. 使数据库所使用的 JVM 可访问经编译的类。

   数据库 Java 虚拟机（JVM）所使用的用户定义类可以驻留在 /QIBM/UserData/OS400/SQLLib/Function 目录中或驻留在已对数据库注册的 JAR 文件中。

/QIBM/UserData/OS400/SQLLib/Function 是 /sqllib/function 的 iSeries 等价项，也就是 DB2 UDB 在其它平台上存储 Java 存储过程和 Java UDF 的目录。如果类是某个 Java 包的一部分，则它必须驻留在适当的子目录中。例如，如果将 runit 类作为 foo.bar 包的一部分来创建，则 runnit.class 文件应位于集成文件系统目录 /QIBM/ProdData/OS400/SQLLib/Function/foo/bar 中。

还可以将类文件放在已对数据库注册的 JAR 文件中。请使用 SQLJ.INSTALL_JAR 存储过程来注册 JAR 文件。此存储过程用来对 JAR 文件指定 JAR 标识。这个 JAR 标识用来标识类文件驻留所在的 JAR 文件。有关 SQLJ.INSTALL_JAR 以及其它用于操纵 JAR 文件的存储过程的更多信息，参见用于操纵 JAR 文件的 SQLJ 过程。

4. 对数据库注册例程。

   请使用 CREATE PROCEDURE 和 CREATE FUNCTION SQL 语句来对数据库注册 Java SQL 例程。这些语句包含下列元素:

   CREATE 关键字
   用于创建 Java SQL 例程的 SQL 语句以 CREATE PROCEDURE 或 CREATE STATEMENT 开始。

   例程的名称
   然后，SQL 语句标识数据库所知的例程名称。这就是用来从 SQL 中访问 Java 例程的名称。

   参数和返回值
   然后，SQL 语句标识 Java 例程的参数和返回值（如果适用的话）。

   LANGUAGE JAVA
   SQL 语句使用关键字 LANGUAGE JAVA 来指示例程是使用 Java 编写的。

   PARAMETER STYLE KEYWORDS
   然后，SQL 语句使用关键字 PARAMETER STYLE JAVA 或 PARAMETER STYLE DB2GENERAL 来标识参数样式。

   外部名称
   然后，SQL 语句标识要作为 Java SQL 例程进行处理的 Java 方法。外部名称具有以下两种格式之一:

   - 如果该方法包含在位于 /QIBM/UserData/OS400/SQLLib/Function 目录下的类文件中，则使用格式 *classname.methodname* 来标识该方法，其中 *classname* 是类的全限定名称，而 *methodname* 是方法的名称。
   - 如果该方法位于已对数据库注册的 JAR 文件中，则使用格式 *jarid*:*classname.methodname* 来标识该方法，其中 *jarid* 是已注册的 JAR 文件的 JAR 标识，*classname* 是类的名称，而 *methodname* 是方法的名称。

   可使用 “iSeries 导航器” 来创建使用 Java 参数样式的存储过程或用户定义函数。

5. 使用 Java 过程。

   使用 SQL CALL 语句来调用 Java 存储过程。Java UDF 是作为另一个 SQL 语句的一部分来进行调用的函数。

Java 存储过程

当使用 Java(TM) 来编写存储过程时，可使用两种可能的参数传送样式。建议的样式是 JAVA 参数样式，此样式与 “SQLj: SQL” 例程标准中指定的参数样式相匹配。第二种样式 DB2GENERAL 是 DB2(R) UDB 定义的参数样式。参数样式还确定了在编码 Java 存储过程时必须使用的约定。

另外，您还应了解一些对 Java 存储过程施加的限制。

JAVA 参数样式: 在编码使用 JAVA 参数样式的 Java(TM) 存储过程时，必须使用下列约定:

- Java 方法必须是 public void static（非实例）方法。
- Java 方法的参数必须具有 SQL 兼容类型。
- 当参数具有可为空类型（如 String）时，Java 方法可测试 SQL NULL 值。
- 通过使用单一元素数组来返回输出参数。
- Java 方法可使用 getConnection 方法来访问当前数据库。

使用 JAVA 参数样式的 Java 存储过程是公用静态方法。在类中，通过存储过程的方法名和特征符来对它们进行标识。调用存储过程时，它的特征符是根据 CREATE PROCEDURE 语句定义的变量类型动态生成的。

如果在允许空值的 Java 类型中传送参数，则 Java 方法可将该参数与空作比较以确定输入参数是否为 SQL NULL。

下列 Java 类型不支持空值:

- short
- int
- long
- float
- double

如果将空值传送给不支持空值的 Java 类型，则将返回错误代码为 -20205 的 SQL Exception。

输出参数是作为包含一个元素的数组传送的。Java 存储过程可设置该数组的第一个元素以设置输出参数。

使用以下 “Java 数据库连接”（JDBC）调用来访问与嵌入应用程序上下文的连接:

```
connection=DriverManager.getConnection("jdbc:default:connection");
```

此连接接着使用 JDBC API 来运行 SQL 语句。

以下是一个小型的存储过程，它带有一个输入和两个输出。它运行给定的 SQL 查询，并返回结果中的行数和 SQLSTATE。

示例: 带有一个输入和两个输出的存储过程

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
       }
      }
     }
```

在 SQLj 标准中，要在使用 JAVA 参数样式的例程中返回结果集，必须显式地设置结果集。在创建返回结果集的过程时，将把附加的结果集参数添加至参数列表的末尾。例如，以下语句

```
CREATE PROCEDURE RETURNTWO()
DYNAMIC RESULT SETS 2
LANGUAGE JAVA
PARAMETER STYLE JAVA
EXTERNAL NAME 'javaClass!returnTwoResultSets'
```

将调用具有特征符 `public static void returnTwoResultSets(ResultSet[] rs1, ResultSet[] rs2)` 的 Java 方法。

如以下示例所示，必须显式地设置结果集的输出参数。与 DB2GENERAL 样式相同，不应关闭结果集和相应的语句。

示例: 返回两个结果集的存储过程

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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();
  Stringsql2 = "select value from table01 where index=2";
  rs2[0] = stmt2.executeQuery(sql2);
  }
}
```

在服务器上，并不会检查附加的结果集参数来确定结果集的定序。服务器上的结果集按照它们的打开次序返回。为了确保与 SQLj 标准兼容，如前面所示，应按照打开结果的次序来指定它们。

«

DB2GENERAL 参数样式: » 在编码使用 DB2GENERAL 参数样式的 Java(TM) 存储过程时，必须使用下列约定:

- 定义 Java 存储过程的类必须扩展 Java com.ibm.db2.app.StoredProc 类或是它的子类。
- Java 方法必须是 public void 实例方法。
- Java 方法的参数必须具有 SQL 兼容类型。
- Java 方法可使用 isNull 方法来测试 SQL NULL 值。
- Java 方法必须使用 set 方法显式地设置返回参数。
- Java 方法可使用 getConnection 方法来访问当前数据库。

包含 Java 存储过程的类必须扩展 com.ibm.db2.app.StoredProc 类。Java 存储过程是公用实例方法。在类中，通过存储过程的方法名和特征符来对它们进行标识。调用存储过程时，它的特征符是根据 CREATE PROCEDURE 语句定义的变量类型动态生成的。

com.ibm.db2.app.StoredProc 类提供了 isNull 方法，这个方法允许 Java 方法确定输入参数是否是 SQL NULL。com.ibm.db2.app.StoredProc 类还提供了用于设置输出参数的 set...( ) 方法。必须使用这些方法来设置输出参数。如果不设置输出参数，则输出参数返回 SQL NULL 值。

com.ibm.db2.app.StoredProc 类提供了以下例程来将 JDBC 连接提取至嵌入应用程序上下文。使用以下 JDBC 调用来访问与嵌入应用程序上下文的连接:

```
public Java.sql.Connection getConnection( )
```

此连接接着使用 JDBC API 来运行 SQL 语句。

以下是一个小型的存储过程，它带有一个输入和两个输出。它处理给定的 SQL 查询，并返回结果中的行数和 SQLSTATE。

示例: 带有一个输入和两个输出的存储过程

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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());
    }
  }
}
```

要在使用 DB2GENERAL 参数样式的过程中返回结果集，在该过程结束时必须保持结果集和相应语句处于打开状态。客户机应用程序必须关闭所返回的结果集。如果返回多个结果集，则它们按照它们的打开次序返回。例如，以下存储过程返回两个结果集。

示例: 返回两个结果集的存储过程

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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);
}
```

«

有关 Java 存储过程的限制: » 下列限制适用于 Java(TM) 存储过程:

- Java 存储过程不应创建附加的线程。仅当作业具有多线程能力时才能在作业中创建附加的线程。由于不保证调用 SQL 存储过程的作业具有多线程能力，所以 Java 存储过程不应创建附加的线程。
- 不能使用沿用权限来访问 Java 类文件。
- Java 存储过程始终使用系统上安装的最新版本的 Java Development Kit。
- 由于 Blob 和 Clob 类同时驻留在 java.sql 和 com.ibm.db2.app 包中，所以，如果在同一程序中同时使用这两个类，则程序员必须使用这些类的完整名称。程序必须确保使用 com.ibm.db2.app 中的 Blob 和 Clob 类来作为传送给存储过程的参数。
- 当创建 Java 存储过程时，系统在库中生成服务程序。这个服务程序用来存储过程定义。服务程序具有系统生成的名称。可通过检查创建存储过程的作业的作业记录来获取此名称。如果先保存并接着恢复服务程序对象，则将恢复过程定义。如果将要把 Java 存储过程从一个系统移至另一系统，则您负责移动包含过程定义的程序以及包含 Java 类的集成文件系统文件。
- Java 存储过程不能设置用来连接至数据库的 JDBC 连接的属性（例如 system naming）。始终使用缺省 JDBC 连接属性，即使在禁用预提取时亦如此。 «

Java 用户定义标量函数

» **Java(TM) 标量函数**从 Java 程序返回一个值给数据库。例如，可以创建返回两个数之和的标量函数。与 Java 存储过程相似，Java 标量函数使用两种参数样式（Java 和 DB2GENERAL）之一。在编码 Java 用户定义函数（UDF）时，您必须了解对创建 Java 标量函数施加的限制。

参数样式 Java: Java 参数样式是由 *SQLJ 第 1 部分: SQL 例程*标准指定的样式。在编码 Java UDF 时，请使用下列约定。

- Java 方法必须是 public static 方法。
- Java 方法必须返回 SQL 兼容类型。返回值是该方法的结果。
- Java 方法的参数必须具有 SQL 兼容类型。
- Java 方法可以对允许空值的 Java 类型测试 SQL NULL。

例如，对于名为 sample!test3 的给定 UDF，如果它返回 INTEGER 并接收 CHAR(5)、BLOB(10K) 和 DATE 类型的自变量，则 DB2 期望该 UDF 的 Java 实现具有以下特征符:

```
import com.ibm.db2.app.*;
public class sample {
   public static int test3(String arg1, Blob arg2, Date arg3) { ... }
}
```

Java 方法的参数必须具有 SQL 兼容类型。例如，如果将某个 UDF 声明为接收 SQL 类型为 t1、t2 和 t3 的自变量并返回类型 t4，则将其称为具有期望的 Java 特征符的 Java 方法:

```
public static T4 name (T1 a, T2 b, T3 c)  { .....}
```

其中:

- *name* 是方法名
- T1 至 T4 是与 SQL 类型 t1 至 t4 相对应的 Java 类型。
- *a*、*b* 和 *c* 是输入自变量的任意变量名。

可以在存储过程和 UDF 的参数传送约定中找到 SQL 类型与 Java 类型之间的关联。

SQL NULL 值由尚未初始化的 Java 变量表示。如果这些变量具有对象类型，则它们具有 Java 空值。如果将 SQL NULL 传送给 Java 标量数据类型，如 int，则发生异常条件。

在使用 JAVA 参数样式时，要从 Java UDF 返回结果，只需从方法返回结果即可。

```
{ ....
  return value;
}
```

与 UDF 和存储过程中使用的 C 模块相似，不能在 Java UDF 中使用 Java 标准 I/O 流（System.in、System.out 和 System.err）。

参数样式 DB2GENERAL: Java UDF 可使用参数样式 DB2GENERAL。在此参数样式中，将返回值作为函数的最后一个参数传送，并且必须使用 com.ibm.db2.app.UDF 类的 *set* 方法来设置返回值。

在编码 Java UDF 时，必须遵循下列约定:

- 包含 Java UDF 的类必须扩展 Java com.ibm.db2.app.UDF 类或者是它的子类。
- 对于 DB2GENERAL 参数样式，Java 方法必须是 public void 实例方法。
- Java 方法的参数必须具有 SQL 兼容类型。
- Java 方法可使用 isNull 方法来测试 SQL 空值。
- 对于 DB2GENERAL 参数样式，Java 方法必须使用 set() 方法来显式地设置返回参数。

包含 Java UDF 的类必须扩展 Java 类 com.ibm.db2.app.UDF。使用 DB2GENERAL 参数样式的 Java UDF 必须是 Java 类的空实例方法。例如，对于名为 sample!test3 的给定 UDF，如果它返回 INTEGER 并接收 CHAR(5)、BLOB(10K) 和 DATE 类型的自变量，则 DB2 期望该 UDF 的 Java 实现具有以下特征符:

```
import com.ibm.db2.app.*;
public class sample extends UDF {
   public void test3(String arg1, Blob arg2, String arg3, int result) { ... }
}
```

Java 方法的参数必须具有 SQL 类型。例如，如果将某个 UDF 声明为接收 SQL 类型为 t1、t2 和 t3 的自变量并返回类型 t4，则将其称为具有期望的 Java 特征符的 Java 方法:

```
public void name (T1 a, T2 b, T3 c, T4 d)  { .....}
```

其中:

- *name* 是方法名
- T1 至 T4 是与 SQL 类型 t1 至 t4 相对应的 Java 类型。
- *a*、*b* 和 *c* 是输入自变量的任意变量名。
- *d* 是任意的变量名，表示所计算的 UDF 结果。

SQL 类型与 Java 类型之间的关联在存储过程和 UDF 的参数传送约定一节中给出。

SQL NULL 值由尚未初始化的 Java 变量表示。根据 Java 规则，如果这些变量具有原语类型，则它们具有零值，如果它们具有对象类型，则具有 Java 空值。为了将 SQL NULL 与普通的零区分开来，可以对任何输入自变量调用 isNull 方法。

```
   { ....
if (isNull(1)) { /* argument #1 was a SQL NULL */ }
else           { /* not NULL */ }
   }
```

在上一个示例中，自变量编号从 1 开始。与下面的其它函数一样，isNull() 函数从 com.ibm.db2.app.UDF 类继承。在使用 DB2GENERAL 参数样式时，要从 Java UDF 返回结果，请在 UDF 中使用 set() 方法，如下所示:

```
   { ....
set(2, value);
   }
```

其中，2 是输入自变量的索引，而 *value* 是具有兼容类型的文字或变量。自变量号是所选输出在自变量列表中的索引。在本节的第一个示例中，int 结果变量的索引为 4。在 UDF 返回之前未设置的输出自变量具有空值。

与 UDF 和存储过程中使用的 C 模块相似，不能在 Java UDF 中使用 Java 标准 I/O 流（System.in、System.out 和 System.err）。

通常，DB2 要将 UDF 调用许多次，即对一个查询中的输入或结果集的每一行调用一次。如果在 UDF 的 CREATE FUNCTION 语句中指定了 SCRATCHPAD，则 DB2 就会认识到在该 UDF 的连续调用之间需要一些“连续性”，因此，对于 DB2GENERAL 参数样式函数，并不是对每个调用将所实现的 Java 类实例化，而通常称对每个语句的每个 UDF 引用实例化一次。但是，如果对 UDF 指定了 NO SCRATCHPAD，则通过调用类构造函数来对每一个对该 UDF 的调用将一个清理实例实例化。

暂存区对于跨多个 UDF 调用保存信息而言非常有用。Java UDF 可以使用实例变量或设置暂存区来获得调用之间的连续性。Java UDF 使用 com.ibm.db2.app.UDF 中的 getScratchPad 和 setScratchPad 方法来访问暂存区。查询结束时，如果在 CREATE FUNCTION 语句上指定了 FINAL CALL 选项，则调用对象的 public void close() 方法（对于 DB2GENERAL 参数样式函数）。如果未定义此方法，则一个存根函数获取控制权并忽略该事件。com.ibm.db2.app.UDF 类包含许多有用的变量和方法，可以在 DB2GENERAL 参数样式 UDF 中使用这些变量和方法。下表对这些变量和方法作了解释。

变量和方法	描述
public static final int SQLUDF_FIRST_CALL = -1; **public static final int SQLUDF_NORMAL_CALL = 0;** **» public static final int SQLUDF_TF_FIRST = -2;** **public static final int SQLUDF_TF_OPEN = -1;** **public static final int SQLUDF_TF_FETCH = 0;** **public static final int SQLUDF_TF_CLOSE = 1;** **public static final int SQLUDF_TF_FINAL = 2; «**	对于标量 UDF，这些是用于确定调用是第一个调用还是正常调用的常量。对于表 UDF，这些是用于确定调用是第一个调用、打开调用、提取调用、关闭调用还是最终调用的常量。
public Connection getConnection();	该方法获取此存储过程调用的 JDBC 连接句柄并返回一个 JDBC 对象，该对象表示调用应用程序与数据库的连接。这与 C 存储过程中的空 SQLConnect() 调用的结果类似。
public void close();	如果 UDF 是使用 FINAL CALL 选项创建的，则数据库在 UDF 求值结束时将调用此方法。这与 C UDF 的最终调用类似。如果 Java UDF 类未实现此方法，则忽略此事件。
public boolean isNull(int i)	此方法测试具有给定索引的输入参数是否是 SQL NULL。

变量和方法	描述
public void set(int i, short s); **public void set(int i, int j);** **public void set(int i, long j);** **public void set(int i, double d);** **public void set(int i, float f);** **public void set(int i, BigDecimal bigDecimal);** **public void set(int i, String string);** **public void set(int i, Blob blob);** **public void set(int i, Clob clob);** **public boolean needToSet(int i);**	这些方法将输出自变量设置为给定的值。如果发生任何问题，则抛出异常，这些问题包括: • UDF 调用未在进行中 • 索引未引用有效的输出自变量 • 数据类型不匹配 • 数据长度不匹配 • 发生代码页转换错误
public void setSQLstate(String string);	可以从 UDF 中调用此方法来设置要从此调用中返回的 SQLSTATE。如果不能接受该字符串作为 SQLSTATE，则抛出异常。用户可在外部程序中设置 SQLSTATE 来从函数中返回错误或警告。在此情况下，SQLSTATE 必须包含下列其中一项: • “00000”，指示成功 • “01Hxx”，其中 xx 是任何两个数字或大写字母，指示警告 • “38yxx”，其中 y 是 “I” 与 “Z” 之间的大写字母，而 xx 是任何两个数字或大写字母，指示错误
public void setSQLmessage(String string);	此方法与 setSQLstate 方法类似。此方法设置 SQL 消息结果。如果不能接受该字符串（例如，长于 70 个字符），则抛出异常。
public String getFunctionName();	此方法返回正在处理的 UDF 的名称。
public String getSpecificName();	此方法返回正在处理的 UDF 的特定名称。
public byte[] getDBinfo();	此方法将正在处理的 UDF 的未经处理的原始 DBINFO 结构作为字节数组返回。必须已使用 DBINFO 选项注册该 UDF（使用 CREATE FUNCTION）。
public String getDBname(); **public String getDBauthid();** **public String getDBver_rel();** **public String getDBplatform();** **public String getDBapplid();** **public String getDBapplid();** **public String getDBtbschema();** **public String getDBtbname();** **public String getDBcolname();**	这些方法返回正在处理的 UDF 的 DBINFO 结构中的适当字段的值。必须已使用 DBINFO 选项注册该 UDF（使用 CREATE FUNCTION）。如果在 UPDATE 语句中的 SET 子句的右边指定了用户定义函数，则 getDBtbschema()、getDBtbname() 和 getDBcolname() 方法只返回有意义的信息。
public int getCCSID();	此方法返回作业的 CCSID。
public byte[] getScratchpad();	此方法返回当前正在处理的 UDF 的暂存区的副本。首先必须使用 SCRATCHPAD 选项声明该 UDF。
public void setScratchpad(byte ab[]);	此方法使用给定字节数组的内容来覆盖当前正在处理的 UDF 的暂存区。首先必须使用 SCRATCHPAD 选项声明该 UDF。该字节数组必须具有 getScratchpad() 所返回的大小。

变量和方法	描述
public int getCallType();	此方法返回当前正在进行的调用的类型。这些值与 sqludf.h 中定义的 C 值相对应。可能的返回值包括: • SQLUDF_FIRST_CALL • SQLUDF_NORMAL_CALL • » SQLUDF_TF_FIRST • SQLUDF_TF_OPEN • SQLUDF_TF_FETCH • SQLUDF_TF_CLOSE • SQLUDF_TF_FINAL «

«

有关 Java 用户定义函数的限制: » 下列限制适用于 Java(TM) 用户定义函数（UDF）:

- Java UDF 不应创建附加的线程。仅当作业具有多线程能力时才能在作业中创建附加的线程。由于不能保证调用 SQL 存储过程的作业具有多线程能力，所以 Java 存储过程不应创建附加的线程。
- 对数据库定义的 Java 存储过程的完整名称限长 279 个字符。此项限制是 EXTERNAL_NAME 列的结果，该列的最大宽度为 279 个字符。
- 不能使用沿用权限来访问 Java 类文件。
- Java UDF 始终使用系统上安装的最新版本的 JDK。
- 由于 Blob 和 Clob 类同时驻留在 java.sql 和 com.ibm.db2.app 包中，所以，如果在同一程序中同时使用这两个类，则程序员必须使用这些类的完整名称。程序必须确保使用 com.ibm.db2.app 中的 Blob 和 Clob 类来作为传送给存储过程的参数。
- 与有源函数相似，在创建 Java UDF 时，使用库中的服务程序来存储函数定义。服务程序的名称由系统生成，可以在创建函数的作业的作业记录中找到该名称。如果保存此对象并接着将其恢复至另一个系统，则将恢复函数定义。如果将要把 Java UDF 从一个系统移至另一系统，则您负责移动包含函数定义的服务程序以及包含 Java 类的集成文件系统文件。
- Java UDF 不能设置用来连接至数据库的 JDBC 连接的属性（例如 system naming）。除了在禁用预提取时外，始终使用缺省 JDBC 连接属性。 «

Java 用户定义表函数: » DB2 提供了让函数返回表的能力。这对于以表格式将数据库外的信息透露给数据库非常有用。例如，可以创建一个表来透露在 Java(TM) 虚拟机（JVM）中设置的用于 Java 存储过程和 Java UDF（表和标量）的属性。

*SQLJ 第 1 部分: SQL 例程*标准支持表函数。因此，只有使用参数样式 DB2GENERAL 的表函数可用。

可对表函数进行五种不同类型的调用。下表说明了这些调用。这些调用假定已在创建函数 SQL 语句上指定了暂存区。

扫描时间中的点	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
在表函数的第一个 OPEN 之前	没有调用	调用类构造函数（表示新建暂存区）。使用 FIRST 调用来调用 UDF 方法。
在表函数的每个 OPEN 时。	调用类构造函数（表示新建暂存区）。使用 OPEN 调用来调用 UDF 方法。	使用 OPEN 调用来调用 UDF 方法。

扫描时间中的点	NO FINAL CALL LANGUAGE JAVA SCRATCHPAD	FINAL CALL LANGUAGE JAVA SCRATCHPAD
在表函数数据的新行的每个 FETCH 时。	使用 FETCH 调用来调用 UDF 方法。	使用 FETCH 调用来调用 UDF 方法。
在表函数的每个 CLOSE 时	使用 CLOSE 调用来调用 UDF 方法。还将调用 close() 方法（如果存在此方法的话）。	使用 CLOSE 调用来调用 UDF 方法。
在表函数的最后一个 CLOSE 之后。	没有调用	使用 FINAL 调用来调用 UDF 方法。还将调用 close() 方法（如果存在此方法的话）。

示例: Java 表函数: 下面是一个 Java 表函数的示例，这个表函数确定用来运行 Java 用户定义表函数的 JVM 中设置的属性。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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);
   }
 }
}
```

在编译表函数并将它的类文件复制到 /QIBM/UserData/OS400/SQLLib/Function 之后，可使用以下 SQL 语句来向数据库注册该函数。

```
create function properties()
returns table (property varchar(500), value varchar(500))
external name 'JVMProperties.dump' language java
parameter style db2general fenced no sql
disallow parallel scratchpad
```

在注册该函数之后，可将其用作 SQL 语句的一部分。例如，以下 SELECT 语句返回表函数所生成的表。

```
SELECT * FROM TABLE(PROPERTIES())
```

用于操纵 JAR 文件的 SQLJ 过程

Java(TM) 存储过程和 Java UDF 都可使用存储在 Java JAR 文件中的 Java 类。要使用 JAR 文件，必须将 *jar 标识*与 JAR 文件相关联。系统在 SQLJ 模式中提供了允许操纵 *jar 标识*和 JAR 文件的存储过程。这些过程允许安装、替换和除去 JAR 文件。它们还提供了使用和更新与 JAR 文件相关联的 SQL 类别的能力。

有关更多信息，参见下列主题:

- SQLJ.INSTALL_JAR
- SQLJ.REMOVE_JAR
- SQLJ.REPLACE_JAR
- SQLJ.UPDATEJARINFO
- SQLJ.RECOVERJAR

SQLJ.INSTALL_JAR: SQLJ.INSTALL_JAR 存储过程将 JAR 文件安装到数据库系统中。可以在后续的 CREATE FUNCTION 和 CREATE PROCEDURE 语句中使用这个 JAR 文件。

权限: CALL 语句的授权标识对 SYSJAROBJECTS 和 SYSJARCONTENTS 目录表拥有的特权必须至少包括下列其中一项:

- 下列系统权限:
  - 对表的 INSERT 和 SELECT 特权
  - 对 QSYS2 库的系统权限 *EXECUTE
- 管理权限

CALL 语句的授权标识所拥有的特权还必须具有下列权限:

- 对 *jar-url* 参数中指定的正在安装的 JAR 文件的“读取”（*R）访问权。
- 对安装 JAR 文件的目录的“写”、“执行”和“读取”（*RWX）访问权。此目录是 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*，其中 *schema* 是 *jar-id* 的模式。

不能将沿用权限用于这些权限。

SQL 语法:

```
>>-CALL--SQLJ.INSTALL_JAR-- (--'jar-url'--,--'jar-id'--,--deploy--)-->

>-------------------------------------------------------------><
```

描述:

jar-url 包含将要安装或替换的 JAR 文件的 URL。支持的唯一 URL 方案是“file:”。

jar-id 数据库中要与 *jar-url* 指定的文件相关联的 JAR 标识符。*jar-id* 使用 SQL 命名，并且将把 JAR 文件安装在由隐式或显式限定符指定的模式或库中。

deploy

用来描述部署描述符文件的 install_action 的值。如果此整数是非零值，则应在 install_jar 过程结束时运行部署描述符文件的 install_action。DB2 UDB for iSeries 的当前版本只支持零值。

使用注意事项: 当安装 JAR 文件时，DB2 UDB for iSeries 在 SYSJAROBJECTS 系统类别中注册 JAR 文件。它还从 JAR 文件中抽取 Java(TM) 类文件的名称并在 SYSJARCONTENTS 系统目录中注册每个类。DB2 UDB iSeries 版将 JAR 文件复制到 /QIBM/UserData/OS400/SQLLib/Function 目录的 jar/schema 子目录。DB2 UDB for iSeries 对该 JAR 文件的新副本指定 *jar-id* 子句中给出的名称。不应更改由 DB2 UDB for iSeries 安装到 /QIBM/UserData/OS400/SQLLib/Function/jar 的子目录中的 JAR 文件。而是，应使用 CALL SQLJ.REMOVE_JAR 和 CALL SQLJ.REPLACE_JAR SQL 命令来除去或替换已安装的 JAR 文件。

示例: 从 SQL 交互式会话发出以下命令。

```
CALL SQLJ.INSTALL_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar', 0)
```

将把位于 file:/home/db2inst/classes/ 目录中的 Proc.jar 文件安装到 DB2 UDB for iSeries 中，并具有名称 myproc_jar。需要使用 Procedure.jar 文件的后续 SQL 命令将使用名称 myproc_jar 来引用它。 «

SQLJ.REMOVE_JAR: » SQLJ.REMOVE_JAR 存储过程从数据库系统中除去 JAR 文件。

授权: CALL 语句的授权标识对 SYSJARCONTENTS 和 SYSJAROBJECTS 目录表拥有的特权必须至少包括下列其中一项:

- 下列系统权限:
  - 对表的 SELECT 和 DELETE 特权
  - 对 QSYS2 库的系统权限 *EXECUTE
- 管理权限

CALL 语句的授权标识所拥有的特权还必须具有以下权限。

- 对正在除去的 JAR 文件的 *OBJMGT 权限。该 JAR 文件名为 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile。

不能将沿用权限用于此权限。

语法:

```
>>-CALL--SQLJ.REMOVE_JAR--(--'jar-id'--,--undeploy--)----------><
```

描述:

jar-id 将要从数据库中除去的 JAR 文件的 JAR 标识符。

undeploy
用来描述部署描述符文件的 remove_action 的值。如果此整数是非零值，则应在 install_jar 过程结束时运行部署描述符文件的 remove_action。DB2 UDB for iSeries 的当前版本只支持零值。

示例: 从 SQL 交互式会话发出以下命令:

```
CALL SQLJ.REMOVE_JAR('myProc_jar', 0)
```

将从数据库中除去 JAR 文件 myProc_jar。 «

SQLJ.REPLACE_JAR: » SQLJ.REPLACE_JAR 存储过程将 JAR 文件替换到数据库系统中。

授权: CALL 语句的授权标识对 SYSJAROBJECTS 和 SYSJARCONTENTS 目录表拥有的特权必须至少包括下列其中一项:

- 下列系统权限:
  - 对表的 SELECT、INSERT 和 DELETE 特权
  - 对 QSYS2 库的系统权限 *EXECUTE

- 管理权限

CALL 语句的授权标识所拥有的特权还必须具有下列权限:

- 对 *jar-url* 参数指定的正在安装的 JAR 文件的“读取”（*R）访问权。
- 对正在除去的 JAR 文件的 *OBJMGT 权限。该 JAR 文件名为 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile。

不能将沿用权限用于这些权限。

语法:

```
>>-CALL--SQLJ.REPLACE_JAR--(--'jar-url'--,--'jar-id'--)--------><
```

描述:

jar-url 包含将要替换的 JAR 文件的 URL。支持的唯一 URL 方案是“file:”。

jar-id 数据库中要与 *jar-url* 指定的文件相关联的 JAR 标识符。*jar-id* 使用 SQL 命名，并且将把 JAR 文件安装在由隐式或显式限定符指定的模式或库中。

使用注意事项: SQLJ.REPLACE_JAR 存储过程替换先前使用 SQLJ.INSTALL_JAR 安装在数据库中的 JAR 文件。

示例: 从 SQL 交互式会话发出以下命令:

```
CALL SQLJ.REPLACE_JAR('file:/home/db2inst/classes/Proc.jar' , 'myproc_jar')
```

将把 *jar-id* myproc_jar 所引用的当前 JAR 文件替换为位于 file:/home/db2inst/classes/ 目录中的 Proc.jar 文件。

«

SQLJ.UPDATEJARINFO: » SQLJ.UPDATEJARINFO 更新 SYSJARCONTENTS 目录表的 CLASS_SOURCE 列。此过程不是 SQLJ 标准的一部分，但 DB2 UDB for iSeries 存储过程构建器使用此过程。

授权: CALL 语句的授权标识对 SYSJARCONTENTS 目录表拥有的特权必须至少包括下列其中一项:

- 下列系统权限:
  - 对表的 SELECT 和 UPDATEINSERT 特权
  - 对 QSYS2 库的系统权限 *EXECUTE
- 管理权限

运行 CALL 语句的用户还必须具有下列权限:

- 对 *jar-url* 参数中指定的 JAR 文件的“读取”（*R）访问权。对正在安装的 JAR 文件的“读取”（*R）访问权。
- 对安装 JAR 文件的目录的“写”、“执行”和“读取”（*RWX）访问权。此目录是 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*，其中 *schema* 是 *jar-id* 的模式。

不能将沿用权限用于这些权限。

语法:

```
>>-CALL--SQLJ.UPDATEJARINFO--(--'jar-id'--,--'class-id'--,--'jar-url'--)-->

>-----------------------------------------------------------><
```

描述:

jar-id 数据库中要更新的 JAR 标识符。

class-id
要更新的类的包限定类名。

jar-url 包含要用来更新 JAR 文件的类文件的 URL。支持的唯一 URL 方案是 “file:”。

示例: 从 SQL 交互式会话发出以下命令:

```
CALL SQLJ.UPDATEJARINFO('myproc_jar', 'mypackage.myclass',
                        'file:/home/user/mypackage/myclass.class')
```

将使用 mypackage.myclass 类的新版本来更新与 *jar-id* myproc_jar 相关联的 JAR 文件。将从文件 /home/user/mypackage/myclass.class 获取那个类的新版本。 «

SQLJ.RECOVERJAR: » SQLJ.RECOVERJAR 过程获取存储在 SYSJAROBJECTS 目录中的 JAR 文件并将其恢复至 /QIBM/UserData/OS400/SQLLib/Function/jar/*jarschema*/*jar_id*.jar 文件。

授权: CALL 语句的授权标识对 SYSJAROBJECTS 目录表拥有的特权必须至少包括下列其中一项:

- 下列系统权限:
  - 对表的 SELECT 和 UPDATEINSERT 特权
  - 对 QSYS2 库的系统权限 *EXECUTE
- 管理权限

运行 CALL 语句的用户还必须具有下列权限:

- 对安装 JAR 文件的目录的 “写”、“执行” 和 “读取” (*RWX) 访问权。此目录是 /QIBM/UserData/OS400/SQLLib/Function/jar/*schema*, 其中 *schema* 是 *jar-id* 的模式。
- 对正在除去的 JAR 文件的 *OBJMGT 权限。该 JAR 文件名为 /QIBM/UserData/OS400/SQLLib/Function/jar/schema/jarfile。

语法:

```
>>-CALL--SQLJ.RECOVERJAR--(--'jar-id'--)----------------------><
```

描述:

jar-id 数据库中要恢复的 JAR 标识符。

示例: 从 SQL 交互式会话发出以下命令:

```
CALL SQLJ.UPDATEJARINFO('myproc_jar')
```

将使用 SYSJARCONTENT 表中内容来更新与 myproc_jar 相关联的 JAR 文件。将把该文件复制到 /QIBM/UserData/OS400/SQLLib/Function/jar/jar_schema myproc_jar.jar。 «

Java 存储过程和 UDF 的参数传送约定

» 下表列示了在 Java(TM) 存储过程和 UDF 中表示 SQL 数据类型的方式。

SQL 数据类型	Java 参数样式 JAVA	Java 参数样式 DB2GENERAL
SMALLINT	short	short
INTEGER	int	int
BIGINT	long	long
DECIMAL(p,s)	BigDecimal	BigDecimal

SQL 数据类型	Java 参数样式 JAVA	Java 参数样式 DB2GENERAL
NUMERIC(p,s)	BigDecimal	BigDecimal
REAL 或 FLOAT(p)	float	float
DOUBLE PRECISION、FLOAT 或 FLOAT(p)	double	double
CHARACTER(n)	String	String
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
指示符变量	-	-
CLOB	-	com.ibm.db2.app.Clob
BLOB	-	com.ibm.db2.app.Blob
DBCLOB	-	com.ibm.db2.app.Clob
DataLink	-	-

Java 与其它编程语言

对于 Java(TM)，您有多种方法来调用以非 Java 语言编写的代码。

Java 本机接口

其中一种调用用另一语言编写的代码的方法是将所选 Java 方法作为“本机方法”实现。本机方法是用另一语言编写的过程，它提供 Java 方法的实际实现。本机方法可以使用“Java 本机接口”（JNI）来访问 Java 虚拟机。这些本机方法在 Java 线程（它是内核线程）下运行，因此必须具有线程安全性。如果可以在同一进程的多个线程中同时启动一个函数，则该函数具有线程安全性。当且仅当一个函数所调用的所有函数都具有线程安全性时，该函数才具有线程安全性。

本机方法是访问 Java 中不直接支持的系统函数或与现有用户代码接口的“桥梁”。使用本机方法时请务必谨慎，因为所调用的代码可能不具有线程安全性。  有关 JNI 和 ILE 本机方法的更多信息，参见将 Java 本机接口用于本机方法。 

OS/400 PASE 本机方法

iSeries Java 虚拟机（JVM）现在支持使用在 OS/400(R) PASE 环境中运行的本机方法。Java 的 OS/400 PASE 本机方法使您能够容易地将在 AIX(R) 中运行的 Java 应用程序移植到 iSeries 服务器中。可以将类文件和 AIX 本机方法库复制到 iSeries 上的集成文件系统中并从任何控制语言（CL）、Qshell 或 OS/400 PASE 终端会话命令提示符下运行它们。 

java.lang.Runtime.exec()

您可以使用 java.lang.Runtime.exec() 来从 Java 程序中调用程序或命令。exec() 方法启动另一个进程，任何 iSeries 程序或命令都可以在该进程中运行。在此模型中，可使用子进程的标准输入、标准输出和标准错误来进行进程间通信。

进程间通信

一种选择是使用套接字来进行父进程和子进程之间的进程间通信。

还可使用流文件来在程序之间进行通信。或者，参见进程间通信示例以获得与运行于另一进程中的程序通信时可供选择的方法的概述。

» 要从其它语言调用 Java，参见示例：从 C 调用 Java 或示例：从 RPG 调用 Java 以获取更多信息。 «

还可以使用 IBM Toolbox for Java 来调用 iSeries 服务器上的现有程序和命令。对于 IBM Toolbox for Java，通常使用数据队列和 iSeries 消息来进行进程间通信。

注意：使用 Runtime.exec()、IBM Toolbox for Java 或 JNI 可能会有损 Java 程序的可移植性。在“纯”Java 环境中，应该避免使用这些方法。

Java“调用”API

通过 Java 调用 API（它也是“Java 本机接口”（JNI）规范的一部分），允许非 Java 应用程序使用 Java 虚拟机。它还允许将 Java 代码用作应用程序的扩展。

将 Java 本机接口用于本机方法

只有在纯 Java(TM) 不能满足编程需要的情况下，才应使用本机方法。通过仅在下列情况下使用本机方法来限制本机方法的使用:

- 要访问那些使用纯 Java 无法得到的系统函数。
- 要实现那些可从本机实现中显著获益的对性能影响很大的方法。
- 要与允许 Java 调用其他 API 的现有应用程序接口（API）连接。

要将“Java 本机接口”（JNI）用于本机方法，请执行下面这些步骤:

1. 通过用标准 Java 语言语法指定哪些方法是本机方法来设计类。
2. 决定包含本机方法实现的服务程序（*SRVPGM）的库和程序名。当在类的静态初始化程序中编码 System.loadLibrary() 方法调用时，指定服务程序的名称。
3. 使用 `javac` 工具来将 Java 源编译成类文件。
4. 使用 `javah` 工具来创建头文件（.h）。这个头文件包含用于创建本机方法实现的精确原型。-d 选项指定应在哪个目录中创建头文件。
5. 通过使用“从流文件复制”（CPYFRMSTMF）命令将头文件从集成文件系统复制到源文件的成员中。必须将头文件复制到源文件成员中，这样 C 编译器才能使用它。使用新的流文件支持，以便“创建绑定 ILE C/400 程序”（CRTCMOD）命令将 C 源文件和 C 头文件保留在集成文件系统中。有关 CRTCMOD 命令以及流文件的使用的更多信息，参见 WebSphere Development Studio: ILE C/C++ Programmer’s Guide（SC09-2712） 。
6. 编写本机方法代码。有关用于本机方法的语言和函数的详细信息，参见 Java 本机方法和线程注意事项。
   a. 包括先前步骤中创建的头文件。
   b. 与头文件中的原型精确匹配。

c. 如果要将字符串传送给 Java 虚拟机，则将该字符串转换成“美国信息交换标准代码”（ASCII）。有关更多信息，参见 Java 字符编码。

7. 如果本机方法必须与 Java 虚拟机交互作用，则使用 JNI 附带提供的函数。
8. 使用 CRTCMOD 来将 C 源代码编译成模块（*MOD）对象。
9. 通过使用“创建服务程序”（CRTSRVPGM）命令，将一个或多个模块对象绑定成一个服务程序（*SRVPGM）。此服务程序的名称必须与 System.load() 或 System.loadLibrary() 函数调用中的 Java 代码中提供的名称相匹配。
10. 如果在 Java 代码中使用 System.loadLibrary() 调用，则执行下列其中一项。

    如果是在使用比 J2SDK 旧的版本:
    将包含新服务程序的 iSeries 库添加到 iSeries 库列表中。要添加库，请使用“添加库列表项”（ADDLIBLE）命令。这使 Java 程序在处理 System.loadLibrary() 函数时能够找到该服务程序。

    如果使用的是 J2SDK V1.2 或更高版本:
    您无需更改库列表。而是，您可以:

    - 将您需要的库列表包括在 LIBPATH 环境变量中。可以在 QShell 中以及从 iSeries 命令行更改 LIBPATH 环境变量。
      - 在 Qshell 命令提示处，输入:
        ```
        export LIBPATH=/QSYS.LIB/MYLIB.LIB
        java -Djava.version=1.4 myclass
        ```
      - 或者，从命令行输入:
        ```
        ADDENVVAR LIBPATH '/QSYS.LIB/MYLIB.LIB'
        JAVA PROP((java.version 1.4)) myclass
        ```
    - 或者，在 **java.library.path** 属性中提供该列表。可以在 QShell 中以及从 iSeries 命令行更改 java.library.path 属性。
      - 在 Qshell 命令提示处，输入:
        ```
        java -Djava.library.path=/QSYS.LIB/MYLIB.LIB -Djava.version=1.4 myclass
        ```
      - 或者，从 iSeries 命令行中输入:
        ```
        JAVA PROP((java.library.path '/QSYS.LIB/MYLIB.LIB') (java.version '1.4')) myclass
        ```

    其中，*/QSYS.LIB/MYLIB.LIB* 是您想使用 System.loadLibrary() 调用来装入的库，*myclass* 是 Java 应用程序的名称。
11. System.load(String patches) 的 patches 的语法可以是下列任何一项:
    - “路径”（指定服务程序提供的库的集成文件系统文件名），它是指向 *SRVPGM 的符号链接，如“/qsys.lib/mylib.lib/myNMsp.srvpgm”
    - /qsys.lib/sysNMsp.srvpgm
    - /qsys.lib/mylib.lib/myNMsp.srvpgm
    - 如果正在使用的版本比 J2SDK 旧: /qsys.lib/%libl%.lib/myNMsp.srvpgm

      注意: 这与使用 System.loadLibrary(″myNMsp″) 方法等效。

    注意: 如果将 pathname 用作字符串文字，则必须用引号将它括起来。例如，System.load(″/qsys.lib/mylib.lib/myNMsp.srvpgm″)。
12. System.loadLibrary(String libya) 的 “libya” 的语法是 mysp。系统通过使用 *LIBL 来查找 mysp。例如，loadLibrary(″myNMsp″) 与 System.load(″/qsys.lib/%libl%.lib/myNMsp.srvpgm″) 等效。如果将 “pathname” 用作字符串文字，则必须用引号将库名括起来。

    注意: J2SDK 不支持 %libl% 语法。

有关 JNI 的完整描述，请参考 Sun Microsystems 的 Java 本机接口和 The Source for Java Technology java.sun.com 。

有关如何将 JNI 用于本机方法的示例，参见示例：将 Java 本机接口用于本机方法。

Java 调用 API

“调用” API 是 “Java(TM) 本机接口” （JNI）的一部分，它允许非 Java 代码创建 Java 虚拟机以及装入和使用 Java 类。此功能允许多线程程序在多个线程中使用正在单个 Java 虚拟机中运行的 Java 类。

Java 虚拟机由应用程序控制。应用程序可以创建 Java 虚拟机、调用 Java 方法（与应用程序调用子例程的方式相类似）以及破坏 Java 虚拟机。在创建 Java 虚拟机之后，它就保持准备好在进程中运行，直到应用程序显式地破坏它为止。在被破坏时，Java 虚拟机执行清理操作，例如运行终止程序、结束 Java 虚拟机线程，以及释放 Java 虚拟机资源。

有了已准备好运行的 Java 虚拟机，用 C 语言编写的应用程序就可以调用至 Java 虚拟机中，以执行任何函数。它还可以从 Java 虚拟机返回至 C 应用程序，然后再调用至 Java 虚拟机中，如此反复。Java 虚拟机只需创建一次，在调用至 Java 虚拟机中以运行或多或少的 Java 代码之前，不必重建。

在使用 “调用” API 来运行 Java 程序时，STDOUT 和 STDERR 的目的地受环境变量 QIBM_USE_DESCRIPTOR_STDIO 的使用的控制。如果将此环境变量设置为 Y 或 I（例如，QIBM_USE_DESCRIPTOR_STDIO=Y），则 Java 虚拟机使用 STDIN、STDOUT 和 STDERR 的文件描述符（分别是 fd 0、fd 1 和 fd 2）。在此情况下，程序必须将这些文件描述符设置为有效值，具体方式是将它们作为此作业中前三个文件或管道打开。作业中打开的第一个文件的 fd 为 0，第二个文件的 fd 为 1，第三个文件的 fd 为 2。对于用 spawn API 启动的作业，可使用文件描述符映射预先指定这些描述符（参见有关 Spawn API 的文档）。如果环境变量 QIBM_USE_DESCRIPTOR_STDIO 未设置或设置为任何其它值，则不对 STDIN、STDOUT 或 STDERR 使用文件描述符。而是将 STDOUT 和 STDERR 路由选择至当前作业所拥有的一个假脱机文件，此时使用 STDIN 将导致 IO 异常。

有关使用 “调用” API 的示例，请参见示例：Java 调用 API。有关 IBM Developer Kit for Java 支持的 “调用” API 函数的详细信息，参见调用 API 函数。

“调用” API 函数: IBM Developer Kit for Java(TM) 支持下面这些 “调用” API 函数。

注意: 在使用此 API 之前，必须确保您处于具有多线程能力的作业中。有关具有多线程能力的作业的信息，参见多线程应用程序。

- **JNI_GetDefaultJavaVMInitArgs**

  注意: 只有 Java Development Kit（JDK）1.1.x 才支持此函数。

  返回一个 JDK 1.1 结构，该结构包含在创建 Java 虚拟机时需要传送给 JNI_CreateJavaVM 的自变量的缺省值。

  特征符:

  ```
  jint JNI_GetDefaultJavaVMInitArgs(void *args_);
  ```

- **JNI_GetCreatedJavaVMs**

  返回关于已创建的所有 Java 虚拟机的信息。

  特征符:

  ```
  jint JNI_GetCreatedJavaVMs(JavaVM **vmBuf,
                             jsize bufLen,
                             jsize *nVMs);
  ```

vmBuf 是一个输出区，其大小由 bufLen 确定，bufLen 是指针数。每个 Java 虚拟机都有相关联的 JavaVM 结构，该结构在 java.h 中定义。除非 vmBuf 已满，否则此 API 会将一个指向与每个已创建的 Java 虚拟机相关联的 JavaVM 结构的指针存储到 vmBuf 中。指向 JavaVM 结构的指针按照相应 Java 虚拟机的创建顺序存储。nVMs 返回当前已创建的虚拟机数。iSeries 服务器支持创建多个 Java 虚拟机，因此有可能是一个大于 1 的值。此信息以及 vmBuf 的大小共同确定是否返回指向每个已创建的 Java 虚拟机的 JavaVM 结构的指针。

- **JNI_CreateJavaVM**

  允许您创建 Java 虚拟机以及随后在应用程序中使用它。

  Java Development Kit 1.1.x 的特征符:

  ```
  jint JNI_CreateJavaVM(JavaVM **p_vm,
                        JNIEnv **p_env,
                              void *vm_args);
  ```

  Java 2 Software Development Kit（J2SDK）的特征符:

  ```
  jint JNI_CreateJavaVM(JavaVM **p_vm,
                        void **p_env,
                              void *vm_args);
  ```

  p_vm 是新创建的 Java 虚拟机的 JavaVM 指针的地址。其它几个 JNI “调用” API 使用 p_vm 来标识 Java 虚拟机。p_env 是新创建的 Java 虚拟机的 “JNI 环境” 指针的地址。它指向由启动那些函数的 JNI 函数组成的表。vm_args 是一个包含 Java 虚拟机初始化参数的结构。当使用 JDK 1.1.x 时，可通过调用 JNI_GetDefaultJavaVMInitArgs 来获得包含缺省值的结构。有关如何使用 J2SDK 来执行此操作的详细信息，

  参见 Java本机接口  。

  如果启动 “运行 Java”（RUNJVA）命令或 JAVA 命令，并指定带有等价命令参数的属性，则优先采用命令参数而忽略该属性。例如，将忽略此命令中的 os400.optimization 参数:

  ```
  JAVA CLASS(Hello) PROP((os400.optimization 0))
  ```

  有关 JNI_CreateJavaVM API 支持的 0S/400 独有属性的列表，参见 Java 系统属性。

  注意: 在一个进程中有多个 Java 虚拟机的情况下，所有 Java 虚拟机共享分配给任何本机方法的同一进程静态存储器。Java 虚拟机内部实现已经以每个 Java 虚拟机为基础将数据分区，但您必须考虑到，对于本机方法应用程序，Java 虚拟机将共享进程静态存储器。有关其它注意事项，参见支持多个 Java 虚拟机。

- **DestroyJavaVM**

  破坏 Java 虚拟机。

  特征符:

  ```
  jint DestroyJavaVM(JavaVM *vm)
  ```

  当创建 Java 虚拟机时，vm 是返回的 JavaVM 指针。

- **AttachCurrentThread**

  将一个线程与 Java 虚拟机相连，以便它可以使用 Java 虚拟机服务。

  Java Development Kit（JDK）1.1.x 的特征符:

  ```
  jint AttachCurrentThread(JavaVM *vm,
                           JNIEnv **p_env,
                           void *thr_args);
  ```

  Java 2 Software Development Kit（J2SDK）的特征符:

```
jint AttachCurrentThread(JavaVM *vm,
                         void **p_env,
                         void *thr_args);
```

JavaVM 指针 vm 标识线程将要连接的 Java 虚拟机。p_env 是指向放置当前线程的“JNI 接口”指针的位置的指针。thr_args 包含特定 VM 线程连接自变量。

- **DetachCurrentThread**

  特征符:

```
jint DetachCurrentThread(JavaVM *vm);
```

  vm 标识将要与线程断开连接的 Java 虚拟机。

有关“调用”API 函数的完整描述，请参考 Sun Microsystems 的 Java 本机接口规范，或者访问 The Source for Java Technology java.sun.com 。

支持多个 Java 虚拟机: iSeries 服务器上的 Java(TM) 与 Sun Microsystems 的参考实现不同，它支持在单个作业或进程中创建多个 Java 虚拟机。这意味着在一个作业中可以多次成功地调用 JNI_CreateJavaVM()，并且 JNI_GetCreatedJavaVMs() 可以在它的结果列表中返回多个 Java 虚拟机。

- 在 V5R2 之前，JNI_GetCreatedJavaVMs 函数有可能在它的 JVM 列表中返回多个 JVM。在 V5R2 中，JNI_GetCreatedJavaVMs 最多返回一个 JVM。
- 在 V5R2 之前，可在单一进程内重复地调用 JNI_CreateJavaVM 函数，对于每次成功的调用，都将创建单独的并且不相同的 JVM。在 V5R2 中，JNI_CreateJavaVM 函数返回一个错误代码。（jni.h 中定义的 JNI_ERR -1 错误）

如果想创建多个 Java 虚拟机以便在单个作业或进程中使用，您应该仔细地考虑下列各项:

本机方法静态存储器作用域限定

- 对于每个作业，无论您创建的 Java 虚拟机的数目是多少，包含本机方法实现的服务程序都只激活一次。这就意味着本机方法静态存储器仅作用于该作业，而不作用于任何特定的 Java 虚拟机。
- 本机方法放入静态存储器中的值独立于调用该本机方法的 Java 虚拟机。这些值对于该作业中的任何 Java 虚拟机都可见。
- 如果您打算在多 Java 虚拟机方案中使用本机方法静态存储器，则应该仔细地考虑对于同步化的可能需求，超越同步化方法和监控程序的使用，这是 Java 虚拟机特定的。将本机方法限制为同步的只会阻止单个 Java 虚拟机内的同时运行，而不会阻止多个 Java 虚拟机中的同时运行。

Java 虚拟机停止

- 如果 Java 虚拟机异常停止，则可能是由于用户调用了 `java.lang.System.exit()`，也可能是由于内部 Java 虚拟机发生故障，失败的 Java 虚拟机以及与它相连的所有线程都将停止。
- 如果进程的初始线程是与失败的 Java 虚拟机相连的线程之一，将会对该初始线程抛出一个异常。如果初始线程处理了此异常，则其它 Java 虚拟机可以继续运行。
- 如果进程的初始线程由于未处理的异常或其它任何原因而停止，则该进程中的所有 Java 虚拟机也都将停止。

从 C 异常停止

如果在多线程作业的任何线程中使用 ILE/C `exit()` or `abort()` 例程，则将使整个作业立即停止，包括所有 Java 虚拟机在内。

示例: Java 调用 API: 本示例遵循标准“调用”API 范例。例如，它执行下列各项:

- 使用 JNI_CreateJavaVM 来创建 Java(TM) 虚拟机。
- 使用 Java 虚拟机来查找要运行的类文件。
- 查找该类的 main 方法的 methodID。
- 调用该类的 main 方法。
- 在发生异常时报告错误。

要编译此程序，必须将它与一个导出函数的服务程序绑定，以启动新的 Java 虚拟机。您需要的入口点如下:

- JNI_GetDefaultJavaVMInitArgs，它对要创建的参数进行初始化。
- JNI_CreateJavaVM，它创建 Java 虚拟机。

编译程序时，无需用编译命令作任何显式处理。导出这些入口点的服务程序位于系统绑定目录中。服务程序的名称是 QJVAJNI。

要运行此程序，使用 `SBMJOB CMD(CALL PGM(YOURLIB/PGMNAME)) ALWMLTTHD(*YES)`。任何创建 Java 虚拟机的作业都必须具有多线程能力。iSeries 服务器上唯一具有多线程能力的作业是批处理立即（BCI）作业。主程序的输出以及该程序的任何输出都结束于 QPRINT 假脱机文件。如果使用“使用提交的作业”（WRKSBMJOB）命令并查看用“提交作业”（SBMJOB）启动的作业，则这些假脱机文件是可见的。

注意: 除非您知道程序是进程中唯一的线程，否则建议您不要使用下面所使用的 C 运行时 `exit()` 例程。当从能够支持多线程的进程中调用 `exit()` 时，它立即结束进程中的所有线程。

示例: 将 Java“调用”API 与 JDK 1.1.x 配合使用。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <jni.h>

int main (int argc, char *argv[])
{
    JDK1_1InitArgs initArgs; /* Virtual Machine (VM) initialization structure.
                              * This is the structure that is passed by reference to JNI_CreateJavaVM().
                              *See jni.h for details.
                              */

    JavaVM* myJVM;           /* The JavaVM and JNIEnv pointers that you get back. */

    JNIEnv* myEnv;           /* ...from the JNI_CreateJavaVM() call. */

    char*   myClasspath;     /* You need to change the classpath, so you have your own. */

    jclass  myClass;         /* The class you are going to find, called 'NativeHello'. */

    jmethodID mainID;        /* The method ID of the class' "main" routine. */

    jclass  stringClass;     /* Required to create a string array argument for 'main'. */

    jobjectArray args;       /* Because main expects an array of strings, you must pass one. */

    /*  Set the version field of the initialization arguments. */
    initArgs.version = 0x00010001;

    /*  Get the default initialization arguments. */
    JNI_GetDefaultJavaVMInitArgs(&initArgs);

    /* Now, you want to add the directory onto the end of the classpath,
     * so that the findClass finds it correctly.  To do this, you have two options:
```

```
     * You can append your classpath entries to the default classpath that is returned
     * by the call to JNI_GetDefaultJavaVMInitArgs, or
     * you can use OS/400 specific functions for the same result.  This is a
     *    three-step solution:
     *       1. Set the CLASSPATH environment variable to its requirements with 'putenv()'
     *       2. Clear the initialization arguments classpath to NULL, which forces
     *          JNI_CreateJavaVM to look at the CLASSPATH value
     *       3. Set the "os400.class.path.system=PRE" property, to force JNI_CreateJavaVM
     *          to prepend the system default classpath to the effective classpath.
     *
     * The first option is used in this example, because it is more platform independent
     *
     * Note: You must specify the directory name in UTF-8 format!  So, you wrap
     *       blocks of code in #pragma convert statements.
     */

#pragma convert(819)
    myClasspath = malloc( strlen(initArgs.classpath) + strlen(":/CrtJvmExample") + 1 );
    strcpy( myClasspath, initArgs.classpath );
    strcat( myClasspath, ":/CrtJvmExample" );
    initArgs.classpath = myClasspath;

#pragma convert(0)

    /*  Create the JVM. */
    if (JNI_CreateJavaVM(&myJVM, &myEnv, &initArgs)) {
        fprintf(stderr, "Failed to create the JVM\n");
        exit(1);
    }

    /*  Use the newly created JVM to find the example class.
     *  Note: Again, you are dealing with UTF-8 here, so you
     *  have to wrap the calls in #pragma convert.
     */

#pragma convert(819)
    if (! (myClass = (*myEnv)->FindClass(myEnv, "NativeHello"))) {

#pragma convert(0)
        /* Cannot find the class, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to find the class 'NativeHello'\n");
        exit(1);  /* Exit stops the entire process on an iSeries server. */
    }

    /* Now, get the method identifier for the 'main' entry point
     * of the class. Note: The signature of 'main' is always
     * the same for every class, "main" and "([Ljava/lang/String;)V"
     * Again, you are dealing with UTF-8.
     */

#pragma convert(819)
    if (! (mainID = (*myEnv)->GetStaticMethodID(myEnv, myClass,
                                                "main",
                                                "([Ljava/lang/String;)V"))) {
        /*  The 'main' methodID is not found for some reason. */
        if ( (*myEnv)->ExceptionOccurred(myEnv) ) {
            /* a java exception occurred, so print it out */
            (*myEnv)->ExceptionDescribe(myEnv);
            /*  The JVM ends. */
            (*myEnv)->FatalError(myEnv, "Failed to find jmethodID of 'main()'");
            }

#pragma convert(0)
        /* Cannot find the 'main' methodID, so write an error message
```

```
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to find the 'main()' method\n");
        exit(1);  /* Exit stops the entire process on an iSeries server. */
    }

#pragma convert(819)
    if (! (stringClass = (*myEnv)->FindClass(myEnv,"java/lang/String"))) {

#pragma convert(0)
        /* Did not find java/lang/String, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to find the java/lang/String");
        exit(1);  /* exit stops the entire process on an iSeries server.*/
    }

    /*  Now, you need to create an empty array of strings,
     *  because ([Ljava/lang/String) is a required part of the signature of
     *  every Java main routine.
     */

    if (! (args = (*myEnv)->NewObjectArray(myEnv,0,stringClass,0)))   {
        /* Empty array was not created, so write an error message
         * to C stderr and exit the program.
         */

        fprintf(stderr, "Failed to create empty array of strings");
        exit(1);  /* Exit stops the entire process on an iSeries server. */
    }

   /* 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) ) {
        fprintf(stderr,"An exception occurred while running 'main'");
        exit(1);
    }

    /* Finally, destroy the JavaVM that you created. */

    if ( (*myJVM)->DestroyJavaVM(myJVM) ) {
        fprintf(stderr, "Failed to destroy the JVM\n");
        exit(1);
    }

    /* All done. */

    return 0;
}
```

示例: 将 Java “调用” API 与 J2SDK 配合使用。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
#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. */
    initArgs.version = 0x00010002;

    /* 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";

    initArgs.options=options;  /* Pass in the classpath that has been set up. */
    initArgs.nOptions = 1;     /* Only passing the one classpath option */

    /*  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;
}
```

«

有关更多信息，参见 Java 调用 API。

Java 本机方法和线程注意事项

可以使用本机方法来访问在 Java(TM) 中不可用的函数。

要更好地将 Java 与本机方法配合使用，您需要理解下面这些概念:

- Java 线程（无论是由 Java 创建还是相连的本机线程）禁用所有浮点异常。如果该线程运行重新启用浮点异常的本机方法，则 Java 不会再次关闭浮点异常。如果用户应用程序在返回运行 Java 代码之前不禁用它们，则发生浮点异常时，Java 代码可能不能正确运行。当本机线程与 Java 虚拟机断开连接时，其浮点异常屏蔽恢复为连接 Java 虚拟机时有效的值。
- 当本机线程连接 Java 虚拟机时，在必要时，Java 虚拟机将更改线程优先级，以便与 Java 定义的一到十优先级方案一致。当线程断开连接时，将恢复优先级。在连接后，线程可以使用本机方法接口（例如，POSIX API）来更改线程优先级。返回 Java 虚拟机时，Java 不更改线程优先级。
- “Java 本机接口”（JNI）的“调用” API 组件允许用户在他们的应用程序中嵌入 Java 虚拟机。如果应用程序创建了 Java 虚拟机，而该 Java 虚拟机异常结束，则当 Java 虚拟机结束时，如果该进程的初始线程与该 Java 虚拟机相连，则向该线程发出 MCH74A5 “Java 虚拟机已终止” iSeries 异常。Java 虚拟机可能会因下列任何原因而异常终止:
  - 用户调用 java.lang.System.exit() 方法。
  - Java 虚拟机所必需的线程结束。
  - Java 虚拟机中发生内部错误。

  以下行为与大多数其他 Java 平台不同。在大多数其它平台上，当 Java 虚拟机结束时，自动创建该 Java 虚拟机的进程也会突然结束。如果应用程序监控并处理所发出的 MCH74A5 异常，则它可以继续运行。否则，如果对该异常不加以处理，则该进程将结束。如果添加处理特定于 iSeries 服务器的 MCH74A5 异常的代码，会使应用程序不易移植到其它平台上。

因为本机方法总是在多线程进程中运行，所以它们包含的代码必须具有线程安全性。这对用于本机方法的语言和函数实施了下面这些限制:

- 不应将 ILE CL 用于本机方法，原因是此语言不具有线程安全性。要运行具有线程安全性的 CL 命令，可以使用 C 语言 system() 函数或 java.lang.Runtime.exec() 方法。
  - 使用 C 语言 system() 函数来从 C 或 C++ 本机方法中运行具有线程安全性的 CL 命令。
  - 使用 java.lang.Runtime.exec() 方法来直接从 Java 中运行具有线程安全性的 CL 命令。
- 您可以使用 ILE C、ILE C++、ILE COBOL 和 ILE RPG 来编写本机方法，但从本机方法中调用的所有函数都必须具有线程安全性。

  注意: 当前仅向用 C 和 C++ 语言提供了编写本机方法的编译期支持。用其它语言编写本机方法虽然是有可能的，但可能会复杂得多。

  注意:
  并非所有标准 C、C++、COBOL 或 RPG 函数都具有线程安全性。
- C 和 C++ exit() 及 abort() 函数决不应在本机方法中使用。这些函数导致运行 Java 虚拟机的整个进程停止。这包括该进程中的所有线程，无论这些线程是否是由 Java 启动的。

  注意: 这里所指的 exit() 函数是 C 和 C++ 函数，与 java.lang.Runtime.exit() 方法不同。

有关 iSeries 服务器上的线程的更多信息，参见多线程应用程序。

本机方法和 Java 本机接口

本机方法是在除 Java(TM) 之外的语言中启动的 Java 方法。本机方法可以访问在 Java 中不能直接使用的特定于系统的函数和 API。

因为本机方法涉及特定于系统的代码，所以使用本机方法会限制应用程序的可移植性。本机方法可以是新的本机代码语句，也可以是调用现有本机代码的本机代码语句。

在决定需要本机方法之后，它就可能必须与它运行时所在的 Java 虚拟机交互作用。“Java 本机接口”（JNI）以平台中立的方式促进了这种交互性。

JNI 是一组接口，它允许本机方法与 Java 虚拟机之间以各种方式进行交互作用。例如，JNI 包括创建新对象和调用方法、获取字段和设置字段、处理异常以及操作字符串和数组的接口。

有关 JNI 的完整描述，参考 Sun Microsystems 的 Java 本机接口或 The Source for Java Technology java.sun.com  。

本机方法中的字符串

许多“Java(TM) 本机接口”（JNI）函数都接受 C 语言风格的字符串作为参数。例如，FindClass() JNI 函数接受指定全限定类文件名的字符串参数。如果找得到该类文件，则 FindClass 将装入它，并将对它的引用返回给 FindClass 的调用程序。

所有 JNI 函数都期望其字符串参数以 UTF-8 编码。有关 UTF-8 的详细信息，您可以参考“JNI 规范”，但在大多数情况下，将 7 位“美国标准信息交换代码”（ASCII）字符视为与它们的 UTF-8 表示法等价便足够了。7 位 ASCII 字符实际上是 8 位字符，但其首位总是为 0。所以，大多数 ASCII C 字符串实际上已经是以 UTF-8 编码的。

缺省情况下，iSeries 服务器上的 C 编译器以扩展二 — 十进制交换码（EBCDIC）操作，所以，可以向 JNI 函数提供以 UTF-8 编码的字符串。有两种方法做到这一点。可以使用文字字符串，也可以使用动态字符串。文字字符串是编译源代码时已知其值的字符串。动态字符串的值在编译期是未知的，它的值是在运行时实际计算得出的。

本机方法中的文字字符串: 如果文字串由 7 位“美国信息交换标准代码”（ASCII）表示的字符组成，则很容易以 UTF-8 格式对该字符串进行编码。如果字符串可用 ASCII 表示（大多数情况是这样），则可以使用更改编译器当前代码页的“pragma”语句将该字符串括起来。于是，编译器在内部以 JNI 所必需的 UTF-8 格式存储该字符串。如果字符串不能以 ASCII 表示，则将原始扩展二 — 十进制交换码（EBCDIC）字符串视为动态字符串，并在将它传送至 JNI 之前使用 iconv() 对它进行处理，这样比较容易编码。有关动态字符串的更多信息，参见动态字符串。

例如，要查找名为 `java/lang/String` 的类，则代码类似于:

```
#pragma convert(819)
myClass = (*env)->FindClass(env,"java/lang/String");
#pragma convert(0)
```

带有数字 819 的第一个 pragma 通知编译器以 ASCII 格式存储随后的所有双引号字符串（文字字符串）。带有数字 0 的第二个 pragma 让编译器还原为编译器的双引号字符串缺省代码页，这通常是 EBCDIC 代码页 37。因此，通过用这些 pragma 将此调用括起来，我们满足了字符串参数需以 UTF-8 编码这一 JNI 需求。

注意: 进行文本替代时请务必要仔细。例如，如果代码类似于:

```
#pragma convert(819)
#define MyString "java/lang/String"
#pragma convert(0)
myClass = (*env)->FindClass(env,MyString);
```

则生成的字符串是 EBCDIC 字符串，其原因是 MyString 的值在编译期间被替换到 FindClass 调用中。进行此替换时，带有数字 819 的 pragma 未生效。因此，文字串未以 ASCII 格式存储。

将动态字符串与 EBCDIC、Unicode 和 UTF-8 进行相互转换: 要操作在运行时计算的字符串变量，可能有必要将字符串与扩展二 — 十进制交换码（EBCDIC）、Unicode 和 UTF-8 进行相互转换。

提供代码页转换功能的系统 API 是 iconv()。要使用 iconv()，请遵循下面这些步骤:

1. 用 `QtqIconvOpen()` 创建转换描述符。
2. 调用 `iconv()` 以使用该描述符转换成一个字符串。
3. 使用 `iconv_close` 来关闭该描述符。

在将 Java(TM) 本机接口用于本机方法示例的示例 3 中，例程创建、使用、然后破坏该例程内的转换描述符 iconv。此方案避免了 iconv_t 描述符的多线程使用问题，但对于性能敏感代码，最好在静态存储器中创建转换描述符，然后使用互斥（mutex）或其它同步设施来减轻对它的多重访问。

示例: 将 Java 本机接口用于本机方法

本示例程序是一个简单的 "Java(TM) 本机接口"（JNI）示例，在此示例中，使用 C 本机方法来显示 "Hello, World"。将 `javah` 工具与 NativeHello 类文件配合使用来生成 NativeHello.h 文件。此示例假定 NativeHello C 实现是服务程序 NATHELLO 的一部分。

注意: NATHELLO 服务程序所在的库必须在库列表中，这样此示例才能运行。

示例 1: NativeHello 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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...");
    }
}
```

示例 2: 生成的 NativeHello.h 头文件

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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
```

此 NativeHello.c 示例显示 C 语言的本机方法实现。此示例显示了如何将 Java 链接至本机方法。然而，它指出由于以下事实而引起的复杂情况: iSeries 服务器在内部是扩展二 — 十进制交换码（EBCDIC）机器。它还显示了由于 JNI 中目前缺少真正的国际化元素而引起的复杂情况。

虽然这些原因对于 JNI 而言早已存在，但它们将导致编写的 C 代码中存在某些独特的特定于 iSeries 服务器的差异。您必须记住，如果写标准输出或标准错误，或读取标准输入，则数据可能是以 EBCDIC 格式编码的。

在 C 代码中，可以很容易地将大多数文字字符串（那些只包含 7 位字符的字符串）转换成 JNI 所需要的 UTF-8 格式。为此，用代码页转换 pragma 将文字字符串括起来。然而，因为可以将信息从 C 代码直接写至标准输出或标准错误，所以可以允许某些文字保持处于 EBCDIC 格式。

注意: #pragma convert(0) 语句将字符数据转换成 EBCDIC。#pragma convert(819) 语句将字符数据转换成“美国信息交换标准代码”（ASCII）。这些语句在编译时转换 C 程序中的字符数据。

示例 3: NativeHello Java 类的 NativeHello.c 本机方法实现

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
#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 covert:  */
#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)
```

有关背景信息，参见将 Java 本机接口用于本机方法。 

Java 的 IBM OS/400 PASE 本机方法

iSeries Java(TM) 虚拟机（JVM）现在支持使用在 OS/400(R) PASE 环境中运行的本机方法。在 V5R2 之前，本机 iSeries JVM 仅使用 ILE 本机方法。OS/400 PASE 本机方法的支持包括:

- 从 OS/400 PASE 本机方法完全使用本机 iSeries Java 本机接口（JNI）
- 能够从本机 iSeries JVM 调用 OS/400 PASE 本机方法

此新支持使您能够容易地将在 AIX(R) 中运行的 Java 应用程序移植到 iSeries 服务器中。可以将类文件和 AIX 本机方法库复制到 iSeries 上的集成文件系统中并从控制语言（CL）、Qshell 或 OS/400 PASE 终端会话命令提示符的其中任何一项运行它们。

有关使用 Java 的 IBM OS/400 PASE 本机方法的更多信息，参见以下主题:

Java OS/400 PASE 环境变量
了解在使用 OS/400 PASE 本机方法之前必须定义的环境变量。这些环境变量管理 OS/400 PASE 和 JVM 运行时环境。

Java OS/400 PASE 错误代码
要帮助对 OS/400 PASE 本机方法进行故障诊断，找出有关错误情况的信息，这些情况由 OS/400 作业记录消息和 Java 运行时异常所指明。

管理本机方法库
找出有关 Java 库命名约定和库搜索算法的信息。此信息对于在 iSeries 服务器上管理本机方法库的多个版本很重要。

示例: Java 的 IBM OS/400 PASE 本机方法
了解如何运行打印出 Java 字符串的内容的简单 Java 程序。与直接从 Java 代码访问字符串不同，示例调用一个本机方法，该本机方法然后通过 JNI 回调至 Java 中来获取字符串值。

此信息假定您已熟悉 OS/400 PASE。有关更多信息，参见以下主题:

OS/400 PASE

Java OS/400 PASE 环境变量

Java 虚拟机（JVM）使用以下变量来启动 OS/400 PASE 环境。要运行 Java 示例的 IBM OS/400 PASE 本机方法，需要设置 QIBM_JAVA_PASE_STARTUP 变量。

有关设置示例的环境变量的信息，参见以下主题:

IBM OS/400 PASE 示例的环境变量。

QIBM_JAVA_PASE_STARTUP

当以下两种情况同时发生时，需要设置此环境变量:

- 正在使用 OS/400 PASE 本机方法
- 正在从 iSeries 命令提示符或 Qshell 命令提示符下启动 Java

JVM 使用此环境变量来启动 PASE 环境。该变量的值标识 OS/400 PASE 启动程序。iSeries 服务器包括两个 OS/400 PASE 启动程序:

- /usr/lib/start32: 启动 32 位 OS/400 PASE 环境

- /usr/lib/start64: 启动 64 位 OS/400 PASE 环境

OS/400 PASE 环境使用的所有共享库对象的位格式必须与 OS/400 PASE 环境的位格式匹配。

当从 OS/400 PASE 终端会话启动 Java 时，不能使用此变量。OS/400 PASE 终端会话总是使用 32 位 OS/400 PASE 环境。任何从 OS/400 PASE 终端会话启动的 JVM 使用与终端会话相同类型的 PASE 环境。

QIBM_JAVA_PASE_CHILD_STARTUP

当次 JVM 的 OS/400 PASE 环境必须与主 JVM 的环境不同时，设置此可选环境变量。调用 Java 中的 Runtime.exec() 会启动次（或子）JVM。

有关更多信息，参见使用 QIBM_JAVA_PASE_CHILD_STARTUP。

示例: IBM OS/400 PASE 示例的环境变量: 要使用 Java 的 IBM OS/400 PASE 本机方法示例，必须设置以下环境变量。

PASE_LIBPATH

iSeries 服务器使用此 OS/400 PASE 环境变量来标识 OS/400 PASE 本机方法库的位置。可以将该路径设置为单个目录或多个目录。对于多个目录，使用冒号（:）来分隔各项。服务器也可以使用 LIBPATH 环境变量。

有关将 Java、本机方法库和 PASE_LIBPATH 与此示例配合使用的更多信息，参见以下主题:

使用 Java、OS/400 PASE 和本机方法库

PASE_THREAD_ATTACH

将此 OS/400 PASE 环境变量设置为 Y 会导致当不是由 OS/400 PASE 启动的 ILE 线程在调用 OS/400 PASE 过程时自动连接至 OS/400 PASE。

有关 OS/400 PASE 环境变量的更多信息，参见以下主题中的适当项:

使用 OS/400 PASE 环境变量

QIBM_JAVA_PASE_STARTUP

JVM 使用此环境变量来启动 OS/400 PASE 环境。该变量的值标识 OS/400 PASE 启动程序。

有关更多信息，参见以下主题:

Java OS/400 PASE 变量

使用 QIBM_JAVA_PASE_CHILD_STARTUP: QIBM_JAVA_PASE_CHILD_STARTUP 环境变量指示任何次 JVM 的 OS/400 PASE 启动程序。当满足以下所有条件时，使用 QIBM_JAVA_PASE_CHILD_STARTUP:

- 要运行的 Java 应用程序通过对 Runtime.exec() 的 Java 调用来创建 Java 虚拟机（JVM ）。
- 主和次 JVM 都使用 OS/400 PASE 本机方法
- 次 JVM 的 OS/400 PASE 环境必须与主 JVM 的 OS/400 PASE 环境不同

当满足所有先前列示的条件时，执行以下操作:

- 将 QIBM_JAVA_PASE_CHILD_STARTUP 环境变量设置为次 JVM 的 OS/400 PASE 启动程序。

- 当从 iSeries 命令提示符或 Qshell 命令提示符下启动主 JVM 时，将 QIBM_JAVA_PASE_STARTUP 环境变量设置为主 JVM 的 OS/400 PASE 启动程序。

  注意: 当从 OS/400 PASE 终端会话启动主 JVM 时，不要设置 QIBM_JAVA_PASE_STARTUP。

次 JVM 的进程继承 QIBM_JAVA_PASE_CHILD_STARTUP 环境变量。另外，OS/400 将次 JVM 进程的 QIBM_JAVA_PASE_STARTUP 环境变量设置为父进程的 QIBM_JAVA_PASE_CHILD_STARTUP 环境变量的值。

下表标识 QIBM_JAVA_PASE_STARTUP 和 QIBM_JAVA_PASE_CHILD_STARTUP 的命令环境和定义的各种组合的结果 OS/400 PASE 环境（如果有话）：

启动环境			结果行为	
命令环境	**QIBM_JAVA_PASE_STARTUP**	**QIBM_JAVA_PASE_CHILD_STARTUP**	**主 JVM OS/400 PASE 启动**	**次 JVM OS/400 PASE 启动**
CL 或 QSH	已定义的 startX	已定义的 startY	使用 startX	使用 startY
CL 或 QSH	已定义的 startX	未定义	使用 startX	使用 startX
CL 或 QSH	未定义	已定义的 startY	无 OS/400 PASE 环境	使用 startY
CL 或 QSH	未定义	未定义	无 OS/400 PASE 环境	无 OS/400 PASE 环境
OS/400 PASE 终端会话	已定义的 startX	已定义的 startY	不允许 *	不允许 *
OS/400 PASE 终端会话	已定义的 startX	未定义	不允许 *	不允许 *
OS/400 PASE 终端会话	未定义	已定义的 startY	使用 OS/400 PASE 终端会话环境	使用 startY
OS/400 PASE 终端会话	未定义	未定义	使用 OS/400 PASE 终端会话环境	无 OS/400 PASE 环境

* 标记为“不允许”的行指示 QIBM_JAVA_PASE_STARTUP 环境变量可能与 OS/400 PASE 终端会话有冲突的情况。由于可能有冲突，所以不允许从 OS/400 PASE 终端会话使用 QIBM_JAVA_PASE_STARTUP。 

管理本机方法库

要使用本机方法库，特别是当要在 iSeries 服务器上管理本机方法库的多个版本时，需要了解 Java 库命名约定和库搜索算法。

OS/400 使用与 Java 虚拟机（JVM）装入的库的名称相匹配的第一个本机方法库。要确保 OS/400 找到正确的本机方法，必须避免与 JVM 使用的本机方法库有关的库名冲突和混淆。

OS/400 PASE 和 AIX Java 库命名约定: 如果 Java 代码装入名为 Sample 的库，则相应的可执行文件必须命名为 libSample.a 或 libSample.so。

Java 库搜索次序: 当启用 JVM 的 OS/400 PASE 本机方法时，服务器使用三种不同的列表（按以下次序）来创建单个本机方法库搜索路径:

1. OS/400 库列表
2. LIBPATH 环境变量
3. PASE_LIBPATH 环境变量

为了执行搜索，OS/400 将库列表转换为集成文件系统格式。QSYS 文件系统对象在集成文件系统中具有等价的名称，但某些集成文件系统对象没有等价的 QSYS 文件系统名。因为库装入程序在 QSYS 文件系统和集成文件系统中寻找对象，所以 OS/400 使用集成文件系统格式来搜索本机方法库。

下表显示 OS/400 如何将库列表中的项转换为集成文件系统格式:

库列表项	集成文件系统格式
QSYS	/qsys.lib
QSYS2	/qsys.lib/qsys2.lib
QGPL	/qsys.lib/qgpl.lib
QTEMP	/qsys.lib/qtemp.lib

示例: 搜索 Sample2 库

在以下示例中，LIBPATH 设置为 /home/user1/lib32:/samples/lib32，而 PASE_LIBPATH 设置为 /QOpenSys/samples/lib。

下表（当从上至下阅读时）指示完全搜索路径:

源	集成文件系统目录
库列表	/qsys.lib /qsys.lib/qsys2.lib /qsys.lib/qgpl.lib /qsys.lib/qtemp.lib
LIBPATH	/home/user1/lib32 /samples/lib32
PASE_LIBPATH	/QOpenSys/samples/lib

注意: 大写和小写字符仅在 /QOpenSys 路径中有意义。

为了搜索库 Sample2，Java 库装入程序按以下次序搜索候选文件:

1. /qsys.lib/sample2.srvpgm
2. /qsys.lib/libSample2.a
3. /qsys.lib/libSample2.so
1. /qsys.lib/qsys2.lib/sample2.srvpgm
2. /qsys.lib/qsys2.lib/libSample2.a
3. /qsys.lib/qsys2.lib/libSample2.so
1. /qsys.lib/qgpl.lib/sample2.srvpgm
2. /qsys.lib/qgpl.lib/libSample2.a
3. /qsys.lib/qgpl.lib/libSample2.so
1. /qsys.lib/qtemp.lib/sample2.srvpgm
2. /qsys.lib/qtemp.lib/libSample2.a
3. /qsys.lib/qtemp.lib/libSample2.so
1. /home/user1/lib32/sample2.srvpgm
2. /home/user1/lib32/libSample2.a
3. /home/user1/lib32/libSample2.so
1. /samples/lib32/sample2.srvpgm
2. /samples/lib32/libSample2.a

3. /samples/lib32/libSample2.so

1. /QOpenSys/samples/lib/SAMPLE2.srvpgm
2. /QOpenSys/samples/lib/libSample2.a
3. /QOpenSys/samples/lib/libSample2.so

OS/400 将列表中实际上存在的第一个候选文件装入 JVM 作为本机方法库。即使类似于 “/qsys.lib/libSample2.a” 和 “/qsys.lib/libSample2.so” 的候选文件出现在搜索中，也不可能在 /qsys.lib 目录中创建集成文件系统文件或符号链接。因此，即使 OS/400 检查这些候选文件，它也不会在以 /qsys.lib 开始的集成文件系统目录中找到它们。

然而，可以创建从其它集成文件系统目录到 QSYS 文件系统中的 OS/400 对象的任意符号链接。因此，有效的候选文件包括诸如 

Java OS/400 PASE 错误代码

下面的列表描述当使用 Java 的 OS/400 PASE 本机方法时在启动或运行时可能遇到的错误。

启动错误: 对于 JVAB55C “无法创建 Java 虚拟机” 消息，有 3 个新的错误代码:

- 19 — 启动 OS/400 PASE 环境时出错。指示用户应用程序或操作系统问题。

  错误代码 19 也包括纯英文的文本。您可能会看到以下错误文本:

  – `Java OS/400 PASE error. OS/400 PASE is already active and the QIBM_JAVA_PASE_STARTUP environment variable is defined.`

    除去 QIBM_JAVA_PASE_STARTUP 环境变量定义或结束活动的 OS/400 PASE 终端会话。

  – `Java OS/400 PASE error. Unable to run OS/400 PASE startup program &programName.`

    由 QIBM_JAVA_PASE_STARTUP 环境变量标识的 OS/400 PASE 程序不存在，或不能在 OS/400 PASE 环境中运行该程序。

  – `Java OS/400 PASE internal error number &errorCode.`

    您可能会看到以下内部错误号中的任何一个:

    - 106 — 指示的 JDK 版本不支持 OS/400 PASE 或指定的 OS/400 PASE 格式。

      指定的 JDK 不支持 OS/400 PASE，或指定的 JDK 不支持 OS/400 PASE 启动程序位格式。对于 V5R2，受支持的组合是:

      - JDK 1.2 和 OS/400 PASE 32 位格式
      - JDK 1.3 和 OS/400 PASE 32 位格式
      - JDK 1.3 和 OS/400 PASE 64 位格式

    将以下错误代码报告给服务代表:

    - 101 — 未标识启动程序。
    - 102 — 无法检索 OS/400 PASE JavaVM 指针
    - 103 — 无法找到 Qp2CallPase
    - 104 — OS/400 PASE 指针大小错误。
    - 105 — 找不到 OS/400 PASE libjvm.a。

- 20 — OS/400 PASE 操作数无效。向服务代表报告。
- 21 — 无法将作业连接至 OS/400 PASE。向服务代表报告。

运行时错误: 除了启动错误外，在 JVM 的 Qshell 输出中还可能会出现 PaseInternalError 或 PaseExit Java 异常:

- PaseInternalError — 指示内部系统错误。检查许可内码作业记录项。

  有关更多信息，参见 Qp2CallPase。
- PaseExit — OS/400 PASE 应用程序调用了 exit() 功能，或 OS/400 PASE 环境已异常结束。检查作业记录和许可内码作业记录以获取附加信息。 «

»

示例: Java 的 IBM OS/400 PASE 本机方法

Java 的 IBM OS/400 PASE 本机方法示例调用本机 C 方法的实例，然后该实例使用“Java 本机接口”（JNI）回调入 Java 代码中。

要查看示例源文件的 HTML 版本，使用以下链接:

- PaseExample1.java
- PaseExample1.c

必须完成以下任务，然后才能运行 OS/400 PASE 本机方法示例:

1. 将示例源代码下载至 AIX 工作站
2. 准备示例源代码
3. 准备 iSeries 服务器

运行 Java 的 OS/400 PASE 本机方法示例: 在完成上述任务后，可以运行示例。使用以下命令之一来运行示例程序:

- 从 iSeries 服务器命令提示符下:

```
JAVA CLASS(PaseExample1) CLASSPATH('/home/example')
```

- 从 Qshell 命令提示符下或 OS/400 PASE 终端会话:

```
cd /home/example
java PaseExample1
```

集成语言环境和 Java 的比较

iSeries 服务器上的 Java(TM) 环境与集成语言环境（ILE）是分开的。Java 不是 ILE 语言，它不能与 ILE 对象模块绑定来在 iSeries 服务器上创建程序或服务程序。

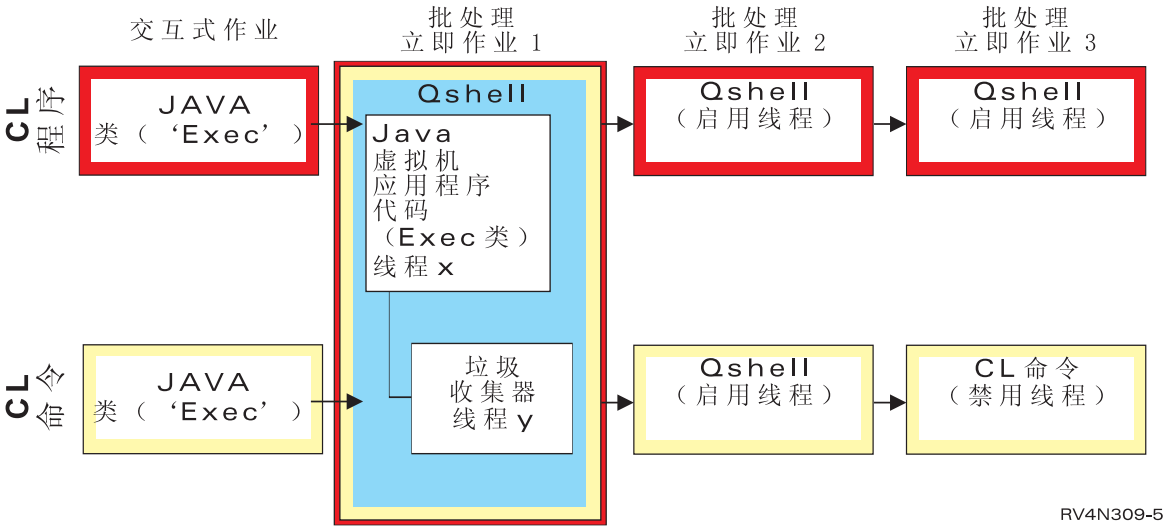
ILE	Java
作为 iSeries 服务器上的库或文件结构一部分的成员存储源代码。	集成文件系统中的流文件包含源代码。
源输入实用程序（SEU）编辑扩展二 — 十进制交换码（EBCDIC）源文件。	“美国信息交换标准代码”（ASCII）源文件通常使用工作站编辑器进行编辑。
源文件编译成目标代码模块，存储在 iSeries 服务器上的库中。	源代码编译成类文件，由集成文件系统负责存储。
目标模块在程序或服务程序中静态地绑定在一起。	类在运行时按需要动态装入。
可以直接调用使用其它 ILE 编程语言编写的函数。	必须使用“Java 本机接口”来从 Java 中调用其他语言。
ILE 语言总是被编译，并作为机器指令运行。	Java 程序可以被解释或编译。

使用 java.lang.Runtime.exec()

java.lang.Runtime.exec() 方法从 Java(TM) 程序中调用程序或命令。实际发生的处理完全取决于向 exec() 方法传送了什么信息。在所有情况下，Runtime.exec() 方法都创建另一个启用了线程的批处理立即（BCI）作业。BCI 作业处理在 Runtime.exec() 方法上传入的命令字符串。

如果使用的是 JDK 1.2 或更低版本，则必须在 iSeries 服务器上安装 Qshell Interpreter 才能使用 java.lang.Runtime.exec() 方法。有关 Qshell Interpreter 的更多信息，参见 Qshell Interpreter。

注意: java.lang.Runtime.exec() 方法在单独的进程中运行程序。这与 C 系统函数不同，C 系统函数在同一进程中运行程序。



如果处理的命令是 Qshell 实用程序，则它在第二个 BCI 作业中运行，并且不创建第三个 BCI 作业。如果处理的命令是 CL 命令，则启动第二个 BCI 作业来运行 Qshell，并且启动第三个 BCI 作业来运行 CL 命令。Qshell 实用程序是 QSH 可直接运行的内部实用程序。Qshell 实用程序的一个示例是 javac 命令，它编译 Java 程序。第二个（或第三个）BCI 作业中的处理与 Java 虚拟机并行运行。那些作业中的任何退出或关闭处理都不会影响原始的 Java 虚拟机。

当调用 iSeries 命令或程序时，您必须确保传送给被调用程序的任何参数都在该程序所期望的代码页中。

有关 java.lang.Runtime.exec() 的示例，参见调用另一个 Java 程序、调用 CL 程序或调用 CL 命令。

示例: 使用 java.lang.Runtime.exec() 来调用另一个 Java 程序

此示例显示如何使用 java.lang.Runtime.exec() 来调用另一个 Java(TM) 程序。这个类调用作为 IBM Developer Kit for Java 的一部分交付的 Hello 程序。当 Hello 类写 System.out 时，此程序获取一个指向该流的句柄，并可以从该流中读取。

注意: 使用 Qshell Interpreter 来调用程序

示例 1: CallHelloPgm 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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
```

有关背景信息，参见使用 java.lang.Runtime.exec()。

示例: 使用 java.lang.Runtime.exec() 来调用 CL 程序

此示例显示如何从 Java(TM) 程序中运行 CL 程序。请参见调用 CL 命令以获取如何在 Java 程序中调用 CL 命令的示例。在此示例中，Java 类 CallCLPgm 运行一个 CL 程序。CL 程序使用“显示 Java 程序”（DSPJVAPGM）命令来显示与 Hello 类文件相关联的程序。此示例假定该 CL 程序已经过编译，并且存在于名为 JAVSAMPLIB 的库中。CL 程序的输出位于 QSYSPRT 假脱机文件中。

注意: JAVSAMPLIB 并不是作为 IBM Developer Kit 许可程序（LP）（程序号 5769-JV1）安装过程的一部分创建的。必须显式地创建该库。

示例 1: CallCLPgm 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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
```

示例 2: 显示 Java CL 程序

```
PGM
DSPJVAPGM CLSF('/QIBM/ProdData/Java400/com/ibm/as400/system/Hello.class') +
               OUTPUT(*PRINT)
ENDPGM
```

有关背景信息，参见使用 java.lang.Runtime.exec()。

示例: 使用 java.lang.Runtime.exec() 来调用 CL 命令

此示例显示如何从 Java 程序中运行控制语言（CL）命令。在此示例中，Java 类运行一个 CL 命令。该 CL 命令使用“显示 Java 程序”（DSPJVAPGM）命令来显示与 Hello 类文件相关联的程序。CL 命令的输出位于 QSYSPRT 假脱机文件中。

» 如果正在使用 JDK 1.1.8 或 JDK 1.2, « 传送到 `Runtime.getRuntime().exec()` 函数中的每个命令都必须括在引号中，并且具有 Qshell 格式。此外，要从 Qshell 中运行 CL 命令，需要传入以下字符串:

```
"system \"CL COMMAND\""
```

其中，*CL COMMAND* 是要运行的 CL 命令。因此，调用命令 MYCLCOM 的程序行应该是:

```
Runtime.getRuntime()Exec("system \"MYCLCOM\"");
```

» **注意**: 当使用 JDK 1.3 或 JDK 1.4 时，应省略斜杠和引号定界符（\"）。例如，当使用 JDK V1.3 或更高版本时，对命令 MYCLCOM 的调用如下:

```
Runtime.getRuntime()Exec("system MYCLCOM");
```

有关更多信息，参见 Java 2 Software Development Kit Standard Edition 的 Java 系统属性中的 os400.runtime.exec 系统属性。 «

示例 1: CallCLCom class

» 当使用 JDK 1.1.8 或 JDK 1.2 时，以下示例使用必要的 Qshell 定界符。如果正在使用 JDK V1.3 或更高版本，省略这些定界符。 «

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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
```

进程间通信

当与在另一进程中运行的程序通信时，可以有许多选择。

其中一个选择是使用套接字来进行进程间通信。一个程序可作为服务器程序，侦听套接字连接上来自客户机程序的输入。客户机程序通过套接字与该服务器相连。在建立套接字连接之后，这两个程序便可以发送或接收信息。

另一个选项是使用流文件在程序间进行通信。为此，需使用 System.in、System.out 和 System.err 类。

第三个选项是使用 IBM Toolbox for Java(TM)，它提供了数据队列和 iSeries 消息对象。

» 还可以从其它语言中调用 Java。有关更多信息，参见示例：从 C 调用 Java 和示例：从 RPG 调用 Java。
«

使用套接字来进行进程间通信

套接字流在运行于不同进程中的程序之间进行通信。这些程序可以单独启动，也可以通过使用 java.lang.Runtime.exec() 方法来从 Java(TM) 主程序启动。如果程序是使用除 Java 之外的语言编写的，则必须确保发生任何“美国信息交换标准代码”（ASCII）或扩展二 — 十进制交换码（EBCDIC）转换。有关更多详细信息，参见 Java 字符编码。

有关使用套接字的示例，参见示例：使用套接字来进行进程间通信。

示例：使用套接字来进行进程间通信： 此示例使用套接字来在 Java(TM) 程序与 C 程序之间进行通信。首先应启动 C 程序，它在套接字上进行侦听。在 Java 程序与该套接字相连接之后，C 程序便使用该套接字连接来向它发送一个字符串。从 C 程序发送的字符串是代码页 819 中的“美国信息交换标准代码”（ASCII）字符串。

应在 Qshell Interpreter 命令行或另一 Java 平台上使用命令 `java TalkToC xxxxx nnnn` 来启动 Java 程序。或者，在 iSeries 命令行上输入 `JAVA TALKTOC PARM(xxxxx nnnn)` 来启动 Java 程序。`xxxxx` 是运行 C 程序所在的系统的域名或“因特网协议”（IP）地址。`nnnn` 是 C 程序使用的套接字的端口号。还应将此端口号用作对 C 程序的调用上的第一个参数。

示例 1：TalkToC 客户机类

注意：请阅读代码示例不保证声明以了解重要的法律信息。

```
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
```

通过对端口号传送一个参数来启动 SockServ.C。例如，CALL SockServ '2001'。

示例 2: SockServ.C 服务器程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
#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);

}
```

有关更多信息，请参见使用套接字来进行进程间通信。

使用输入和输出流来进行进程间通信

输入和输出流在运行于不同进程之中的程序间进行通信。java.lang.Runtime.exec() 方法运行程序。父程序可获取指向子进程输入和输出流的句柄，并且可以读写那些流。如果子程序不是使用 Java(TM) 语言编写的，则必须确保进行任何“美国信息交换标准代码”（ASCII）或扩展二 — 十进制交换码（EBCDIC）转换。有关更多详细信息，参见 Java 字符编码。

有关使用输入和输出流的示例，参见示例：使用输入和输出流来进行进程间通信。

示例：使用输入和输出流来进行进程间通信： 此示例显示如何从 Java(TM) 中调用 C 程序并使用输入和输出流来进行进程间通信。在此示例中，C 程序将一个字符串写至其标准输出流，Java 程序读取此字符串并显示它。此示例假定已创建了一个名为 JAVSAMPLIB 的库，并且已在该库中创建了 CSAMP1 程序。

注意：JAVSAMPLIB 并不是作为 IBM Developer Kit 许可程序（LP）（程序号 5769-JV1）安装过程的一部分创建的。您必须显式地创建它。

示例 1：CallPgm 类

注意：请阅读代码示例不保证声明以了解重要的法律信息。

```
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
```

示例 2: CSAMP1 C 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
#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);

}
```

有关更多信息，参见使用输入和输出流来进行进程间通信。

示例: 从 C 中调用 Java

以下是一个使用 system() 函数来调用 Java Hello 程序的 C 程序示例。

示例: 从 C 中调用 Java

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
#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')");
 }
```

示例: 从 RPG 中调用 Java

这是一个使用 QCMDEXC API 来调用 Java(TM) Hello 程序的 RPG 程序示例。

示例 1: 从 RPG 中调用 Java

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
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
```

Java 平台

» **Java(TM) 平台**是用于开发和管理 Java applet 和应用程序的环境。它由三个主要组件组成: Java 语言、Java 包和 Java 虚拟机。Java 语言和包类似于 C++ 及其类库。Java 包包含类，类可从任何相符的 Java 实现中得到。应用程序编程接口（API）在任何支持 Java 的系统上都应相同。

Java 在编译和运行方式方面与传统语言（如 C++）不同。在传统编程环境中，您编写程序源代码并将其编译成针对特定硬件和操作系统的目标代码。此目标代码与其它目标代码模块绑定，以创建运行程序。代码是特定于特定一组计算机硬件的，如果不加以更改，则不能在其它系统上运行。下图说明了传统语言部署环境。

要高效地使用 Java 平台，请查看下列各项:

Java applet 和应用程序

您可以编写 Java applet 并将其包括在 HTML 页面中，这与包括图像的方式很相像。当使用启用 Java 的浏览器来查看包含 applet 的 HTML 页面时，便将 applet 的代码传送至系统并由浏览器的 Java 虚拟机运行。还可以编写不要求使用 Web 浏览器的 Java 应用程序。

Java 虚拟机

可以将 Java 虚拟机嵌入在 Web 浏览器中，也可以嵌入在诸如 IBM(R) Operating System/400(R)（OS/400(R)）之类的操作系统中。Java 虚拟机由 Java 解释器和 Java 运行时环境组成。解释器执行解释类文件并在特定硬件平台上运行 Java 指令的这一任务。Java 虚拟机允许一次性编写和编译 Java 代码，并在任何平台上运行。

Java JAR 类文件

Java 环境与其它编程环境的不同之处在于 Java 编译器不生成特定于硬件的指令集的机器码。而是，Java 编译器将 Java 源代码转换成 Java 虚拟机指令，并存储在 Java 类文件中。可使用 JAR 文件来存储类文件。类文件不面向特定硬件平台，而是面向 Java 虚拟机体系结构。

Java 线程

Java 是多线程编程语言；因此同时可以在 Java 虚拟机中运行多个线程。Java 线程为 Java 程序提供了同时执行多个任务的方法。

Java Development Kit

Java Development Kit（JDK）是由 Sun Microsystems 发布的供 Java 开发者使用的软件。它包括 Java 解释器、Java 类和 Java 开发工具。请查找下列关于 JDK 的信息:

- Java 包
- Java 工具 «

Java applet 和应用程序

applet 是设计成要包括在 HTML Web 文档中的 Java(TM) 程序。HTML 文档包含标记，标记指定了 Java applet 的名称以及它的“统一资源定位器”（URL）。URL 是 applet 字节码在因特网上的驻留位置。当显示包含 Java applet 标记的 HTML 文档时，启用 Java 的 Web 浏览器将从因特网下载 Java 字节码并使用 Java 虚拟机来处理 Web 文档内的代码。这些 Java applet 使 Web 页面能够包含动画图形或交互式内容。

有关更多信息，参见编写 Applet  ，这是 Sun Microsystems 的 Java applet 教程。此教程包含 applet 的概述、有关编写 applet 的指导以及一些常见的 applet 问题。

应用程序是独立的程序，它们不要求使用浏览器。Java 应用程序的运行方式是：从命令行启动 Java 解释器，并指定包含经编译的应用程序的文件。应用程序通常驻留在部署它们的系统上。应用程序访问系统上的资源，并且受 Java 安全性模型限制。

Java 虚拟机

Java(TM) 虚拟机是一种运行时环境，可以将它添加到 Web 浏览器或任何操作系统（例如，IBM Operating System/400（OS/400））中。Java 虚拟机运行 Java 编译器所生成的指令。它由字节码解释器和运行时组成，无论 Java 类文件最初是在什么平台上开发的，都允许它们在任何平台上运行。

类装入程序和安全性管理器是 Java 运行时的一部分，它们隔离来自另一平台的代码。它们也可以限制装入的每个类所能够访问的系统资源。

注意：Java 应用程序不受限制；只有 applet 才受限制。应用程序可以自由地访问系统资源和使用本机方法。大多数 IBM Developer Kit for Java 程序都是应用程序。

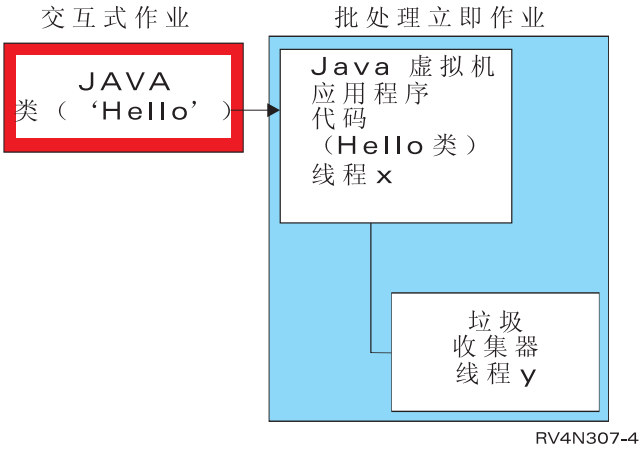
可以使用“创建 Java 程序”（CRTJVAPGM）命令来确保代码满足 Java 运行时为验证字节码而实施的安全性需求。这包括强制类型限制、检查数据转换、确保不会发生参数栈上溢或下溢以及检查访问违例。然而，无需显式地验证字节码。如果不预先使用 CRTJVAPGM 命令，则在首次使用某个类时进行检查。在验证字节码之后，解释器就对字节码进行解码，并运行执行期望的操作所需的机器指令。

注意：仅当指定了 OPTIMIZE(*INTERPRET) 或 INTERPRET(*YES) 时，才能使用 Java 解释器。

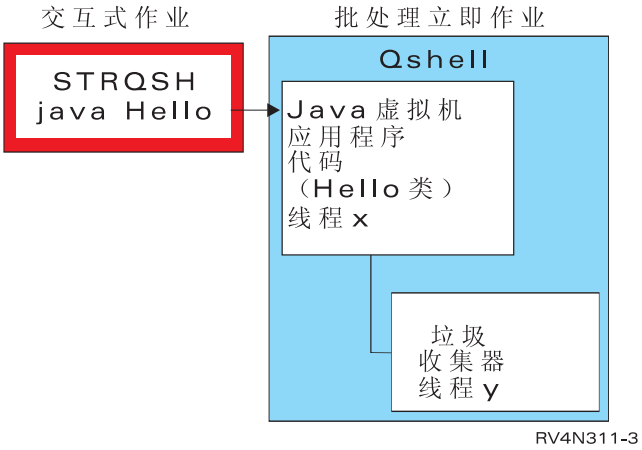
除了装入和运行字节码之外，Java 虚拟机还包括一个用于管理内存的垃圾收集器。垃圾收集与字节码的装入和解释是同时运行的。

Java 运行时环境

每当在 iSeries 命令行上输入“运行 Java”（RUNJVA）命令或 JAVA 命令时，Java 运行时环境便启动。因为 Java 环境是多线程的，所以有必要在一个支持线程的作业（如批处理立即（BCI）作业）中运行 Java 虚拟机。Java 虚拟机启动之后，便可以启动其它线程，垃圾收集器在这些线程中运行。典型的 Java 环境类似于:



通过使用 Qshell 中的 java 命令来从 Qshell Interpreter 启动 Java 运行时环境也是有可能的。在此环境中，Qshell Interpreter 在与交互式作业相关联的 BCI 作业中运行。Java 运行时环境在运行 Qshell Interpreter 的作业中启动。



当从交互式作业启动 Java 运行时环境时，将显示 “Java Shell 屏幕”。此屏幕提供了一个输入行，供输入数据到 System.in 流中，以及显示写入到 System.out 流和 System.err 流的数据。

Java 解释器

Java 解释器是 Java 虚拟机的一部分，它为特定硬件平台解释 Java 类文件。Java 解释器对每个字节码进行运行，并对该字节码运行一系列机器指令。

Java JAR 和类文件

“Java(TM) 压缩文档”（JAR）文件是一种将许多个文件组合成一个文件的文件格式。可使用 JAR 来作为一般归档工具，并且还可用来分发所有类型的 Java 程序（包括 applet）。Java applet 通过单个 “超文本传送协议”（HTTP）事务下载到浏览器中，而不是通过为每个 applet 打开新连接的方式进行下载。这种下载方法提高了 applet 在 Web 页面上装入并开始工作的速度。

JAR 是唯一的跨平台压缩文档格式。JAR 也是唯一能够处理音频文件、图像文件和类文件的格式。JAR 是用 Java 编写的开放标准且可完全扩展的格式。

JAR 格式还支持压缩，从而减小了文件大小和缩短了下载时间。而且，applet 作者可对 JAR 文件中的个别项进行数字签名以认证其来源。

» 要更新 JAR 文件中的类，参见 Java jar 工具。 «

Java 类文件是 Java 编译器编译源文件时生成的流文件。类文件包含描述类的每个字段和方法的表。该文件还包含用来表示 Java 对象的每个方法、静态数据和描述的字节码。

Java 线程

线程是在程序中运行的单个独立流。Java(TM) 是多线程编程语言，所以同一时间可以有多个线程在 Java 虚拟机中运行。Java 线程为 Java 程序提供了同时执行多个任务的方法。线程从本质上讲是程序中的控制流。

线程是一种时髦的编程结构，它用于支持并行程序和改进应用程序的性能和可伸缩性。大多数编程语言都通过使用加载件编程库来支持线程。Java 以内部应用程序接口（API）形式支持线程。

注意：线程使用提供了对增强交互性的支持，这意味着因为更多任务是并行运行的，所以键盘等待时间更短。但是，程序不一定仅仅是因为拥有线程就会增强其交互性。

线程是这样一种机制：等待长时间运行的交互作用，同时又允许程序处理其它工作。线程有能力通过同一代码流支持多个工作流。它们有时被称作**轻量级进程**。Java 语言包括对线程的直接支持。但是，它设计成不支持带有中断或多次等待的异步非分块输入和输出。

线程允许开发在机器带有多个处理器的环境中能够表现良好的并行程序。如果构造正确的话，它们还可提供处理多个事务和用户的模型。

您在许多情况下都可以在 Java 程序中使用线程。某些程序必须能够在参与多个活动的同时，仍能够对用户的附加输入作出响应。例如，Web 浏览器应能够在播放声音的同时对用户输入作出响应。

线程还可以使用异步方法。当调用第二个方法时，第二个方法不必等待第一个方法完成即可继续执行它自己的活动。

也有许多不使用线程的理由。如果程序使用固有的顺序逻辑，则一个线程可以完成整个序列。在这种情况下，使用多个线程将生成一个复杂的程序，这没有什么好处。创建和启动线程要做相当多的工作。如果一个操作仅涉及少量语句，则在单个线程中处理它速度较快。即使该操作从概念上讲是异步的，情况也如此。当多个线程共享对象时，这些对象必须同步才能协调线程访问并维护一致性。同步化将增加程序的复杂度，难以为获得最佳性能而进行调整，同时也是编程错误的根源。

有关线程的更多信息，参见开发多线程应用程序。

Sun Microsystems 的 Java Development Kit

Java(TM) Development Kit（JDK）是由 Sun Microsystems 发布的供 Java 开发者使用的软件。它包括 Java 解释器、Java 类和 Java 开发工具：编译器、调试器、反汇编程序、appletviewer、存根文件生成器和文档生成器。

JDK 使您能够编写一次开发便可以在任何 Java 虚拟机上的任何位置运行的应用程序。在一个系统上使用 JDK 开发的 Java 应用程序可以在另一个系统上使用，而无需更改或重新编译代码。Java 类文件可以移植至任何标准 Java 虚拟机。

要了解有关当前 JDK 的更多信息，请检查您的 iSeries 服务器上的 IBM Developer Kit for Java 版本。

可以输入下列命令之一来检查 iSeries 服务器上的缺省 IBM Developer Kit for Java 的 Java 虚拟机版本：

- `java -version`（在 Qshell 命令提示上）。
- `RUNJVA CLASS(*VERSION)`（在 CL 命令行上）。

然后，在 The Source for Java Technology java.sun.com  处查找 Sun Microsystems 的同一版本的 JDK 以获取特定文档。IBM Developer Kit for Java 是与 Sun Microsystems 的 Java 技术相兼容的实现，所以您应熟悉它们的 JDK 文档。

有关更多信息，参见下列主题:

- 支持多个 Java Development Kit（JDK）提供了有关使用不同 Java 虚拟机的信息。
- 本机方法和 Java 本机接口定义了本机方法的概念及其用途。此主题还简要说明了 “Java 本机接口”。

Java 包

Java 包是对 Java 中的相关类和接口进行分组的一种方法。Java 包类似于其它语言中的类库。

Java 包提供 Java API，它可以作为 Sun Microsystems 的 Java Development Kit（JDK）的一部分获得。

包	内容
java.applet	applet 类
java.awt	图形、窗口和图形用户界面（GUI）类
java.awt.datatransfer	数据传送类
java.awt.event	事件处理类和接口
java.awt.image	图象处理类
java.awt.peer	实现平台无关性的 GUI 接口
java.beans	JavaBean 组件模型 API
java.io	输入和输出类
java.lang	核心语言类
java.lang.reflect	反射 API 类
java.math	任意精度算术
java.net	联网类
java.rmi	“远程方法调用”类
java.rmi.dgc	与 RMI 相关的类
java.rmi.registry	与 RMI 相关的类
java.rmi.server	与 RMI 相关的类
java.security	安全性类
java.security.acl	与安全性相关的类
java.security.interfaces	与安全性相关的类
java.sql	数据库类的 JDBC SQL API
java.text	国际化类
java.util	数据类型
java.util.zip	压缩和解压缩类

有关 Sun Microsystems 的 Java API 的更多信息，参见 Sun Microsystems 的 API 用户指南。

Java 工具

有关 Sun Microsystems 的 Java Development Kit 所提供的工具的完整列表，参见 Sun Microsystems 的工具参考。有关 IBM Developer Kit for Java 支持的每个工具的更多信息，参见 IBM Developer Kit for Java 支持的 Java 工具。

高级主题

以下是 IBM Developer Kit for Java(TM) 的高级主题:

类、包和目录
每个 Java 类都是某个包的一部分。包名与该类所在的目录结构相关。

集成文件系统中的文件
集成文件系统以分层文件结构来存储与 Java 相关的类文件、源文件、ZIP 文件和 JAR 文件。

文件权限
要运行或调试 Java 程序，类文件、ZIP 文件或 JAR 文件需要读权限。请查找有关若干 CL 命令所需的文件权限的更多信息。

批处理作业
Java 程序可以通过使用“提交作业”（SBMJOB）命令来在批处理作业中运行。请查找更多关于 SBMJOB 命令以及如何验证批处理作业是否能够运行多个作业的信息。

Java 类、包和目录

每个 Java(TM) 类都是某个包的一部分。Java 源文件中的第一条语句指示这是哪个类，以及在哪个包中。如果源文件不包含 package 语句，则该类是未命名缺省包的一部分。

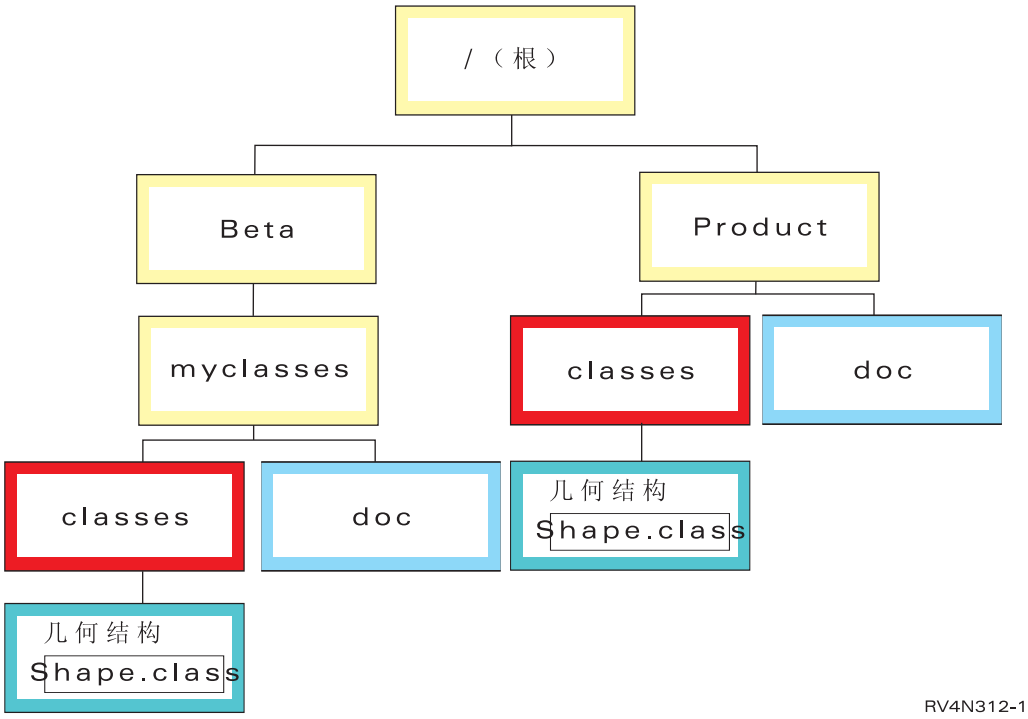
包名与该类驻留所在的目录结构相关。集成文件系统支持与大多数 PC 和 UNIX 系统上的分层文件结构相类似的分层文件结构中的 Java 类。必须将 Java 类存储在这样的目录中：此目录具有与该类的包名相匹配的相对目录路径。例如，考虑以下 Java 类:

```
package classes.geometry;
import java.awt.Dimension;

public class Shape {

Dimension metrics;

   // The implementation for the Shape class would be coded here ...

}
```

以上代码中的 package 语句指示 Shape 类是 classes.geometry 包的一部分。为了让 Java 运行时找得到 Shape 类，请将 Shape 类存储在相对目录结构 classes/geometry 中。

注意：包名与存储类的相对目录名相对应。Java 虚拟机类装入程序通过将相对路径名追加到类路径中指定的每个目录来查找类。Java 虚拟机类装入程序也可以通过搜索类路径中指定的 ZIP 文件或 JAR 文件来查找类。

例如，如果 Shape 类存储在“根”（/）文件系统中的 /Product/classes/geometry 目录中，则将需要在类路径中指定 /Product。



注意: 目录结构中可以存在 Shape 类的多个版本。要使用 Shape 类的“测试版”版本，请将 /Beta/myclasses 放在 CLASSPATH 中任何其它包含 Shape 类的目录或 ZIP 文件之前。

在编译 Java 源代码时，Java 编译器使用 Java 类路径、包名和目录结构来查找包和类。

集成文件系统中的文件

集成文件系统以分层文件结构来存储与 Java 相关的类文件、源文件、ZIP 文件和 JAR 文件。也可以将源文件存储在集成文件系统中。可以将文件存储在这些集成文件系统中:

- “根”（/）文件系统
- 开放式系统文件系统（QOpenSys）
- 用户定义文件系统
- 库文件系统（QSYS.LIB）
- OS/2 Warp Server for iSeries 文件系统（QLANSrv）
- 光盘文件系统（QOPT）

注意: 由于其它集成文件系统不具有线程安全性，所以不支持它们。

集成文件系统中的 Java 文件权限

要运行或调试 Java(TM) 程序，类文件、JAR 文件或 ZIP 文件需要具有读权限（*R）。任何目录都需要读和执行权限（*RX）。

要使用“创建 Java 程序”（CRTJVAPGM）命令来优化程序，类文件、JAR 文件或 ZIP 文件必须具有读权限（*R），目录必须具有执行权限（*X）。如果在类文件名中使用模式，则目录必须具有读和执行权限（*RX）。

要使用“删除 Java 程序”（DLTJVAPGM）命令来删除 Java 程序，必须对类文件具有读和写权限（*RW），目录必须具有执行权限（*X）。如果在类文件名中使用模式，则目录必须具有读和执行权限（*RX）。

要使用“显示 Java 程序”（DSPJVAPGM）命令来显示 Java 程序，您必须对类文件具有读权限（*R），目录必须具有执行权限（*X）。

注意: 不具有执行权限（*X）的文件和目录对于具有 QSECOFR 权限的用户而言似乎总是具有执行权限（*X）。即使两个不同的用户看起来对相同的文件具有同样的访问权，在某些情况下，他们也可能会得到不同的结果。当使用 Qshell Interpreter 或 java.Runtime.exec() 运行外壳程序脚本时，知道这一点很重要。

例如，一个用户编写了一个使用 java.Runtime.exec() 来调用外壳程序脚本的 Java 程序，然后使用具有 QSECOFR 权限的用户标识测试该程序。如果该外壳程序脚本的文件方式具有读和写权限（*RW），则集成文件系统允许具有 QSECOFR 权限的用户标识运行它。然而，某个不具有 QSECOFR 权限的用户可能会尝试运行同一个 Java 程序，因而集成文件系统将发出 java.Runtime.exec() 代码: 由于 *X 丢失，所以外壳程序脚本不能运行。在这种情况下，java.Runtime.exec() 将抛出输入和输出异常。

» 还可以将权限指定给 Java 程序在集成文件系统中创建的新文件。通过对文件使用 os400.file.create.auth 系统属性和对目录使用 os400.dir.create.auth，可使用读、写和执行权限的任意组合。 «

有关更多信息，参见程序和 CL 命令 API 或集成文件系统。

在批处理作业中运行 Java

可使用“提交作业”（SBMJOB）命令来在批处理作业中运行 Java(TM) 程序。在此方式下，“Java Qshell 命令输入”屏幕不可用于处理 System.in、System.out 和 System.err 流。

可以将这些流重定向到其它文件。缺省处理是将 System.out 和 System.err 流发送到假脱机文件。导致 System.in 读请求发生输入和输出异常的批处理作业拥有该假脱机文件。可以在 Java 程序中重定向 System.in、System.out 和 System.err。也可以使用 os400.stdin、os400.stdout 和 os400.stderr 系统属性来重定向 System.in、System.out 和 System.err。

注意: SBMJOB 将“当前工作目录”（CWD）设置为用户概要文件中指定的 HOME 目录。

示例: 在批处理作业中运行 Java

```
SBMJOB CMD(JAVA Hello OPTION(*VERBOSE)) CPYENVVAR(*YES)
```

运行前一示例中的 JAVA 命令将产生第二个作业。因此，运行批处理作业的子系统必须能够运行多个作业。

可以通过遵循这些步骤来验证批处理作业是否能够运行多个作业:

1. 在 CL 命令行上，输入 DSPSBSD(MYSBSD)，其中 MYSBSD 是批处理作业的子系统描述。
2. 选择选项 6，即“作业队列项”。
3. 查看作业队列的“最大活动”字段。

   如果“最大活动”字段小于或等于 1，并且不是 *NOMAX，则在 CL 命令行上输入以下命令:

   ```
   CHGJOBQE SBSD(MYSBSD) JOBQ(MYJOBQ) MAXACT(*NOMAX)
   ```

   其中:

   - MYSBSD 是子系统描述，而
   - MYJOBQ 是作业队列。

在不带图形用户界面的主机上运行 Java 应用程序

如果要在不带图形用户界面（GUI）的主机（如 iSeries 服务器）上运行 Java(TM) 应用程序，则可以使用 Remote Abstract Window Toolkit（AWT）、Class Broker for Java（CBJ）或 Native Abstract Windowing Toolkit（NAWT）。

您将 Remote AWT 与服务器应用程序的安装和管理界面配合使用。这些界面通常带有最少量的复杂图形和高度交互式的内容。Remote AWT 将在 iSeries 服务器与工作站之间分布 AWT 处理。因此，图形加强和高度交互式操作的响应速度不象在带有以本地方式连接的图形终端的平台上的 AWT 实现那样快。要使用 Remote AWT，参见设置 Remote AWT。

可以使用 CBJ 以获得高性能 GUI 服务。由于不建议将 Remote AWT 用于复杂图形或高度交互式操作，所以可以使用 CBJ，它是专门为这些环境设计的。要使用 CBJ，参见设置 CBJ。

» 在使用 X Window System 的 iSeries 服务器上，可以完全地使用 NAWT 来进行 Java 图形计算。**X Window System** 是一个图形系统，它提供了用于显示图形的客户机／服务器基本部件。X Window 图形服务器具有高度的可移植性，从而允许支持各种语言和操作系统。NAWT 为 Java 应用程序和 servlet 提供了使用 Java Development Kit（JDK）的 AWT 图形功能的能力。 «

IBM Developer Kit for Java Remote Abstract Window Toolkit

Remote Abstract Window Toolkit 是 Abstract Window Toolkit（AWT）的一个实现。它允许 Java (TM) 应用程序在不带图形用户界面（GUI）的主机上运行，而无需作任何更改。iSeries 服务器不支持以本地方式连接的图形终端；因此，要允许图形 Java 应用程序在 iSeries 服务器上运行，Remote AWT 是必需的。

您将 Remote AWT 与服务器应用程序的安装和管理界面配合使用。这些界面通常带有最少量的复杂图形和高度交互式的内容。Remote AWT 将在 iSeries 服务器与工作站之间分布 AWT 处理。因此，图形加强和高度交互式操作的响应速度可能不象在带有以本地方式连接的图形终端的平台上的 AWT 实现那样快。

因为不建议将 IBM Developer Kit for Java Remote AWT 用于复杂图形或高度交互式操作，所以您可以使用 Class Broker for Java，它是专门为这些环境设计的。

有关如何设置 Remote AWT 的信息，参见在远程屏幕上设置 Remote Abstract Window Toolkit for Java。

在远程屏幕上设置 Remote Abstract Window Toolkit for Java

借助 Remote Abstract Window Toolkit（AWT），您可以运行 Java(TM) AWT 图形程序（不必对程序源作任何更改）和以远程方式显示图形。要使用 Remote AWT，必须在 iSeries 服务器和远程屏幕上设置“传输控制协议／网际协议”（TCP/IP）并安装 Sun Microsystems 的 Java Development Kit（JDK）1.1.8 或 Java 2 SDK（J2SDK）Standard Edition。

您可以使用任何支持图形的硬件（包括“IBM 网络工作站”）来作为 Remote AWT 的远程屏幕，条件是该硬件符合下列需求:

- 运行 Windows 95(R)、Windows NT 4.0、IBM Operating System/2(R)（OS/2(R)）、Sun Solaris 或 AIX(R) 的支持图形的硬件
- 可以通过 TCP/IP 来访问 iSeries 服务器的已配置硬件
- Java Development Kit 1.1.8 或 J2SDK

要设置 Remote AWT，请完成下列任务:

1. 通过将 Remote AWT 类文件复制到远程屏幕，或将路径映射至远程屏幕上的网络驱动器，来使远程屏幕可访问 Remote AWT 类文件。

2. 将 RAWTGui.zip 或 RAWTGui.jar 添加至远程屏幕的 CLASSPATH。对于 JDK 1.1.8，通过设置 CLASSPATH 环境变量或通过使用 `java` 命令的 `-classpath` 参数，将 `RAWTGui.zip` 文件添加至远程屏幕的 CLASSPATH。对于 J2SDK，当使用 java 命令的 `-jar` 参数时，将把 `RAWTGui.jar` 文件自动添加至 CLASSPATH。
3. 在远程屏幕上启动 Remote AWT。

有关使用 Remote AWT 的详细信息和提示，请查看下列主题:

- 使用 Remote Abstract Window Toolkit 来运行 Java 程序提供了有关如何使用多个 JDK 和 Netscape 来对 iSeries 服务器运行 Java 程序的指示信息。
- 使用 Remote Abstract Window Toolkit 来进行打印说明了如何进行打印，此打印与标准 Java AWT 打印相同。它还显示了如何打印至 iSeries 服务器。
- Remote Abstract Window Toolkit 属性显示了如何使用 os400.class.path.rawt 属性来运行 Remote AWT 应用程序。
- Remote Abstract Window Toolkit SecurityManager 限制提供了有关在 Remote Abstract Window Toolkit SecurityManager 控制下使用 Remote AWT 来运行 Java 应用程序时适用的限制的信息。

有关设置 TCP/IP 的更多信息，参见 TCP/IP Configuration and Reference, SC41-5420 一书中的 How do I set up TCP/IP。 

有关如何设置 Remote AWT 的示例，参见示例: 在 Windows 远程屏幕上设置 Remote Abstract Window Toolkit for Java。

有关 AWT 的更多信息，参见 Sun Microsystems 的 Abstract Window Toolkit。

使远程屏幕可访问 Remote Abstract Window Toolkit for Java 类文件

要使远程屏幕能访问 Remote Abstract Window Toolkit（AWT）类文件，对于 Java(TM) Development Kit（JDK）1.1.*x* 或 Java 2 SDK（J2SDK）Standard Edition V1.2，应遵循下面这些步骤。然而，重要的是要注意，为了让 Remote AWT 能够正确工作，在支持图形的远程屏幕上使用的 RAWTGui.jar 文件的版本必须与主机上使用的 JDK 或 J2SDK 版本相匹配。

如果您使用的是 JDK 1.1.8，则可以:

- 将 Remote AWT 类文件复制到远程屏幕。

  Remote AWT 文件与 IBM Developer Kit for Java 一起安装在两个 ZIP 文件中:

  – /QIBM/ProdData/Java400/jdk118/RAWTApplHost.zip

  – /QIBM/ProdData/Java400/jdk118/RAWTGui.zip

  RAWTApplHost.zip 文件包含用于 iSeries 服务器的 Remote AWT 类。RAWTGui.zip 文件包含用于远程屏幕的 Remote AWT 类。

  将 RAWTGui.zip 从 /QIBM/ProdData/Java400/jdk118 复制到远程屏幕。
- 将路径 /QIBM/ProdData/Java400/jdk118/RAWTGui.zip 映射至远程屏幕上的网络驱动器。

如果您使用的是 J2SDK V1.2 或更高版本，则可以:

- 将 Remote AWT 类文件复制到远程屏幕。

  Remote AWT 文件与 IBM Developer Kit for Java 一起安装在两个 JAR 文件中:

  – /QIBM/ProdData/Java400/jdk12/RAWTAHost.jar

  – /QIBM/ProdData/Java400/jdk12/RAWTGui.jar

如果您使用的是除 1.2 之外的 J2SDK 版本，则将该版本号替代到本节中的所有路径实例中。

RAWTAHost.jar 文件包含用于 iSeries 服务器的 Remote AWT 类。RAWTGui.jar 文件包含用于远程屏幕的 Remote AWT 类。

将 RAWTGui.jar 从 /QIBM/ProdData/Java400/jdk12 复制到远程屏幕上的网络驱动器。

- 将路径 /QIBM/ProdData/Java400/jdk12/RAWTGui.jar 映射到远程屏幕上的网络驱动器。

将 RAWTGui.zip 或 RAWTGui.jar 添加至远程屏幕的 CLASSPATH

设置 CLASSPATH 将使远程屏幕上的 Java(TM) 虚拟机能够找到 Remote Abstract Window Toolkit（AWT）类。此步骤仅对 JDK 1.1.x 而言才是必需的。此步骤对 Java 2 SDK（J2SDK）而言不是必需的。要将 RAWTGui.zip 文件添加至远程屏幕的 CLASSPATH，请执行下列步骤之一:

- 设置 CLASSPATH 环境变量。有关更多信息，参见远程屏幕的 Java Development Kit（JDK）信息。

  将 RAWTGui.zip 文件所在的路径添加至 CLASSPATH 环境变量。
- 使用 `java` 命令的 -classpath 参数。

  当使用 `java` 命令来启动 Remote AWT 时，您可以使用 -classpath 参数来指定 CLASSPATH。CLASSPATH 包括 RAWTGui.zip 文件所在的路径。

  例如，在 Windows(R) 中，CLASSPATH 参数看起来可能象下面这样:

  ```
  -classpath c:\jdk1.1.7\lib\classes.zip;c:\rawt\RAWTGui.zip
  ```

  J2SDK V1.2 和更高版本的 JAR 支持设置 CLASSPATH，因此您无需显式地设置 CLASSPATH 参数。要在远程屏幕上设置类路径并启动 Remote AWT，请输入此命令:

  ```
  java -jar <PATH>RAWTGui.jar
  ```

  其中，<PATH> 是 RAWTGui.jar 文件所在的全限定驱动器和目录。例如，`java -jar c:\rawt2\RAWTGui.jar`。

在远程屏幕上启动 Remote Abstract Window Toolkit for Java

您需要在远程屏幕上启动一次服务器守护程序，它将保持活动状态，直到您结束它为止。在 iSeries 服务器上退出的 Java(TM) 程序不会结束服务器守护程序。

注意: 当启动服务器守护程序时，“欢迎”对话框会保持活动状态。当“欢迎”对话框屏幕关闭时，服务器守护程序便结束了。当服务器守护程序处于活动状态时，您可以将“欢迎”对话框屏幕最小化，并使用该屏幕来结束服务器守护程序。

要启动 JDK 1.1.*x* 的 Remote Abstract Window Toolkit（AWT）服务器守护程序，请在命令行上输入以下命令:

```
java -classpath <PATH>RAWTGui.zip;C:\jdk1.1.8\lib\classes.zip
com.ibm.rawt.server.RAWTPCServer
```

其中，<*PATH*> 是 RAWTGui.jar 文件所在的全限定驱动器和目录。例如，`java -jar c:\rawt2\RAWTGui.jar`。

要为 J2SDK V1.3 启动 Remote AWT 服务器守护程序，请在命令行上输入以下命令:

```
java -jar <PATH>RAWTGui.jar
```

其中，<*PATH*> 是 RAWTGui.jar 文件所在的全限定驱动器和目录。例如，`java -jar c:\rawt2\RAWTGui.jar`。

当 Java 应用程序使用 Remote AWT 进行连接时，服务器守护程序将选择 2000 之上的首个空闲端口。Java 应用程序在结束之前将一直使用此端口。附加的 Java 应用程序将与 2000 之上的后续空闲端口相连。可用端口的范围可达 9999。

有关设置 TCP/IP 的更多信息，参见 TCP/IP Configuration and Reference（SC41-5420）一书中的 How do I set up TCP/IP。 

使用 Remote Abstract Window Toolkit 来运行 Java 程序

要使用 Remote Abstract Window Toolkit（AWT）来运行 Java(TM)，请执行下列步骤:

1. 在远程屏幕上启动 Remote AWT。
2. 在 iSeries 服务器上启动 Java 程序。
   a. 在命令行上输入“运行 Java”（RUNJVA）命令。
      注意: 必须对 Java 程序定义 Java 类路径。
   b. 按 F4 键（提示）。
   c. 在类参数行上输入 Java 程序类名。
   d. 按 F10 键（其他参数）。
   e. 按 Page Down 键。
   f. 在下一个属性名参数行上输入 `RmtAwtServer`。
   g. 在下一个属性值参数行上输入远程屏幕的“传输控制协议／网际协议”（TCP/IP）地址（例如 1.1.11.11）。
   h. 在属性名参数行上输入 `os400.class.path.rawt`。
   i. 在属性值参数行上输入 `1`。
   j. 要输入更多属性，输入 `+`。
   k. 在属性名参数行上输入 `java.version`。
   l. 在属性值参数行上输入 `1.3`。此版本必须与远程屏幕上运行的 RAWTGui.jar 守护程序代码相匹配。
      命令行应具有以下模式，全部内容都在一行上:

      ```
      java class (classname) prop(('RmtAwtServer' '1.1.11.11')
           ('os400.class.path.rawt' '1')('java.version' '1.3'))
      ```

   m. 按“执行”键。

还可以使用 Remote AWT 及 Netscape 来运行 Java 程序。

使用 Remote Abstract Window Toolkit 及 Netscape 来运行 Java 程序: 当使用 Netscape 来运行 Java(TM) 应用程序时，可以通过两种方式之一运行它们。

一种方法是通过打开包含 com.ibm.rawt.server.StartRAWT.class 的 HTML 文件，来在 Netscape Java 虚拟机中启动 Remote Abstract Window Toolkit（AWT）服务器。例如，请参见下面的 RAWT.html 文件。在该服务器启动之后，即可以在 iSeries 服务器上启动 Java 应用程序。

或者，您也可以通过打开包含 com.ibm.rawt.server.StartRAWT400.class 和 IBM Toolbox for Java 类的 HTML 文件，来在 Netscape Java 虚拟机中启动 Remote AWT 服务器。例如，请参见下面显示的 RAWT400.html 文件。在该服务器启动之后，您便可以注册到 Java 应用程序所在的 iSeries 服务器，并启动该应用程序。

在 Netscape Java 虚拟机中的 Remote AWT 服务器中运行

要在 Netscape Java 虚拟机中运行 Remote AWT 服务器，请执行下列步骤:

1. 针对您的 RAWTGui.zip 的特定安装信息，编辑这个示例 .html 文件。此文件（RAWT.html）在 Netscape Java 虚拟机中启动 Remote AWT。

   示例: 在 Netscape Java 虚拟机中启动 Remote AWT

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
<HTML>
<BODY TEXT="#000000" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<APPLET CODE="com.ibm.rawt.server.StartRAWT.class"
codebase="file://C|remote_awt\jdk1.1.7\lib\RAWTGui.zip"
WIDTH=600 HEIGHT=50>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

2. 使用 Netscape 4.05 或更高版本来浏览 RAWT.html 页面。在授予了所有请求的特权之后，Netscape 将启动 Remote AWT 服务器，并在其 Java 虚拟机中运行该服务器。
3. 在 iSeries 服务器上使用 Remote AWT 来启动 Java 应用程序。
4. 在退出应用程序之后，请在按住 Shift 键的同时单击“重新装入”按钮来再次启动 Remote AWT 服务器。

在 Netscape Java 虚拟机中运行 Remote AWT，并注册到 iSeries 服务器

要在 Netscape Java 虚拟机中运行 Remote AWT 服务器并注册到 iSeries 服务器，请执行下列步骤:

1. 针对您的 jt400.zip 和 RAWTGui.zip 的特定安装信息，编辑这个示例 .html 文件。此文件（RAWT400.html）启动 Remote AWT，并使用 IBM Toolbox for Java 来注册到 iSeries 服务器。

   示例: 启动 Remote AWT 并使用 Toolbox for Java 来注册到 iSeries 服务器

   注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
<HTML>
<BODY TEXT="#000000" LINK="#0000EE" VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<APPLET ARCHIVE="file://C\jt400\lib\jt400.zip"
code="com.ibm.rawt.server.StartRAWT400.class"
codebase="file://C|remote_awt\jdk1.1.1\lib\RAWTGui.zip"
WIDTH=600 HEIGHT=50>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

2. 使用 Netscape 4.05 浏览此 RAWT400.html 页面。在授予了所有请求的特权之后，Netscape 将启动 Remote AWT applet，该 applet 显示一个面板，您可在其中执行下列任何操作选项:
   - 使用 IBM Toolbox for Java 注册到带有 Remote AWT 的 iSeries 服务器，以访问 iSeries 服务器。
   - 使用 Remote AWT 属性来输入 Java 应用程序名和自变量。
   - 按“启动应用程序”按钮来使用 Remote AWT 启动指定的 Java 应用程序。

使用 Remote Abstract Window Toolkit 进行打印

使用 Remote Abstract Window Toolkit（AWT）进行打印与标准 Java(TM) AWT 打印相同。Remote AWT 远程屏幕处理打印输出并将输出定向至远程屏幕操作系统所知的任何打印机。这可以是直接连接到远程屏幕的打印机，也可以是远程屏幕操作系统所知的网络打印机。

可以选择在远程屏幕上打印，也可以选择打印至 iSeries 服务器。当应用程序发出打印请求时，将显示新的打印对话框。打印请求允许您选择“远程屏幕”打印机或 OS/400 打印机。如果选择 OS/400 打印机，则会出现一个注册对话框屏幕。注册后，将出现“打印对话框”屏幕。您可以指定 OS/400 打印队列、打印文件以及文件和打印扉页标题。还可以选择纸张大小、方向和份数。

要使用远程打印，必须安装 IBM Toolbox for Java（5763-JC1）并将以下内容添加到 iSeries 服务器上的类路径:

```
QIBM/ProdData/HTTP/Public/jt400/lib/jt400.zip
```

您可以通过添加类路径环境变量或使用类路径参数来更新类路径。

注意: 如果打印到 iSeries 服务器时出现以下消息，则表示尚未安装 IBM Toolbox for Java，或 IBM Toolbox for Java 类不在类路径中。

```
未能装入类文件: com/ibm/as400/access/PrintObjectList.class
在事件调度期间发生异常: java.lang.NoClassDefFoundError: com/ibm/as400/access/PrintObjectList
```

Remote Abstract Window Toolkit 属性

当在 iSeries 服务器上运行 Java(TM) Remote AWT 应用程序时，os400.class.path.rawt 属性值必须设置为 1。使用 Remote AWT 时，有许多缺省属性是必需的。这些缺省属性以及 Remote AWT 的正确版本和 CLASSPATH 是在您使用 os400.class.path.rawt 属性时设置的。Remote AWT 的版本是根据 JDK 版本设置的，该版本要么是缺省版本（如果未指定版本的话），要么是您使用 java.version 属性所指定的版本。

这里是 Remote AWT 所必需的缺省属性。

对于 JDK 1.1.*x*:

- awt.toolkit=com.ibm.rawt.CToolkit

对于 J2SDK:

- awt.toolkit=com.ibm.rawt2.ahost.java.awt.AHToolkit
- java.awt.graphicsenv=com.ibm.rawt2.ahost.java.awt.AHGraphicsEnvironment
- java.awt.printerjob=com.ibm.rawt2.ahost.java.awt.print.AHPrinterjob

远程屏幕的 Remote AWT 属性

如果服务器守护程序或 Java 应用程序异常结束，并发出此消息，则请检查远程屏幕上的 Java 版本。

```
应用程序 — 主机 / 用户 — 工作站中的 JDK 版本与 Remote AWT 版本不兼容...
```

要检查版本级别，请在命令行上输入 `java -version`。如果存在 JDK 版本问题，则可以在远程屏幕上使用这个新属性。此属性不适用于 iSeries 服务器。如果版本级别不是 1.1.x，则必须安装正确的级别。如果它的级别是 1.1.x，则可以通过使用此属性指示 Java 版本来运行 Remote AWT 服务器或 Java 应用程序:

```
-DJdkVersion=1.1.x
```

Remote Abstract Window Toolkit SecurityManager 限制

Java(TM) SecurityManager 并不常用。然而，如果安装了 SecurityManager，则下面列示的这些调用必须成功才能使 RAWT 工作。

- SecurityManager.checkAccess(..)
- SecurityManager.checkMemberAccess(..)
- SecurityManager.checkExit(..)
- SecurityManager.checkRead(String file)
- SecurityManager.checkConnect(...)
- SecurityManager.checkListen(...)
- SecurityManager.checkAccept(...)
- SecurityManager.checkPropertiesAccess(..)

示例: 在 Windows 远程屏幕上设置 Remote Abstract Window Toolkit for Java(TM)

此示例显示在 Windows(R) 远程屏幕上设置 Remote AWT 的一种方法。根据您的喜好，还有许多其它实现方法。可以在其它远程屏幕操作系统上使用类似的过程。通过远程屏幕操作系统提供的 Windows .bat 文件或其它编程设施，设置和启动过程是自动的。

要在 Windows 远程屏幕上设置 Remote Abstract Window Toolkit(AWT) for Java(TM)，请执行以下任务:

- 使远程屏幕可访问 Remote AWT 类文件。

  将 Remote AWT 类文件复制到远程屏幕。执行以下任务之一:

  – 将 /QIBM/ProdData/Java400/jdk118/RAWTGui.zip 复制到 c:\rawt\RAWTGui.zip.

  – 将 /QIBM/ProdData/Java400/jdk1*x*/RAWTGui.jar（其中，*x* 是 Java 2 Software Development Kit（J2SDK）Standard Edition 版本（2、3 或 4））复制到 c:\rawt2\RAWTGui.jar。

- 通过在命令行上输入此命令来在远程屏幕上启动 Remote AWT:

```
java -classpath c:\jdk1.1.8\lib\classes.zip;c:\rawt\RAWTGui.zip
     java com.ibm.rawt.server.RAWTPCServer
```

  或

```
java -jar c:\rawt2\RAWTGui.jar
```

有关更多信息，参见在远程屏幕上设置 Remote Abstract Window Toolkit for Java。

Class Broker for Java

Class Broker for Java(TM)（CBJ）是一个通用框架，用于使用 Java 语言编写客户机／服务器应用程序。通常，客户机／服务器应用程序由客户机对象和服务器对象组成。服务器和客户机负责这两种对象之间的所有通信。这种通信通常由“远程方法调用”（RMI）或通过套接字连接完成。RMI 使用起来既不容易也不灵活，并且，在您能够得心应手地使用套接字之前，要学习相当多的知识。

CBJ 易用、灵活并隐藏了套接字连接的复杂性。除了少数几个用来初始化应用程序的 CBJ 类调用之外，所代理的客户机／服务器应用程序作为本地应用程序出现。CBJ 处理客户机与服务器之间的所有通信和资源装入。程序的某些对象在客户机上运行而另一些在服务器上运行的这一事实几乎是透明的。CBJ 使用 CBJ 运行时来创建客户机和服务器代理。代理程序创建代理对象之后，客户机便通过在服务器的代理上调用方法，来与远程服务器通信。同样，服务器对象通过在客户机的代理上调用方法来与客户机通信。因此，应用程序的客户机方和服务器方似乎都是在调用本地对象的方法。

有关使用 Class Broker for Java 的信息，参见设置 Class Broker for Java。

在远程屏幕上设置 Class Broker for Java

借助 Class Broker for Java(TM)（CBJ），您可以在 iSeries 服务器上运行启用 Class Broker for Java 的 Java 图形程序并以远程方式显示图形。

您可以自行安装 CBJ，也可以请系统管理员为您安装。如果由系统管理员安装该产品，则所有程序员可以共享同一 Java 代码。

可以在 Windows(R) 95/98/NT、UNIX(R) 或 iSeries 服务器上安装 CBJ。在大多数环境中，必须在客户机和服务器上都安装 CBJ。

注意: 如果客户机正在运行通过服务器上的 Web 服务器访问的客户机 applet，则无需在客户机上安装 CBJ。

要在 iSeries 服务器上安装 CBJ，参见在 iSeries 服务器上安装 Class Broker for Java。

要在工作站上安装 CBJ，参见在 Windows 或 UNIX 上安装 Class Broker for Java。

在 iSeries 服务器上安装 Class Broker for Java

要在 iSeries 服务器上安装 Class Broker for Java(TM)（CBJ），请执行下列步骤:

1. 确保已正确安装 IBM Developer Kit for Java。参见安装 IBM Developer Kit for Java 以测试安装。
2. 在集成文件系统中选择一个目录来存储名为 cbj_1.1.jar 的 CBJ 包。例如，/usr/local。
3. 通过输入 QSH 命令来启动 Qshell Interpreter，并转至您在集成文件系统中选择的目录。例如，/usr/local。在安装 IBM Developer Kit for Java 时，将把 cbj_1.1.jar 包安装在 QIBM/ProdData/Java400/ext 中。
4. 在 Qshell 中输入此命令:

   ```
   jar xvf "PATH"cbj_1.1.jar
   ```

   “PATH” 是 cbj_1.1.jar 包所在的目录路径。例如，QIBM/ProdData/Java400/ext。

   将把 CBJ 文件抽取到名为 /usr/local/JCBroker 的子目录中。有关更多信息，参见 cbj_.1.1.jar 的包内容。
5. 通过输入此命令，在 iSeries 服务器上为 jcb.jar 创建 Java 程序:

   ```
   CRTJVAPGM CLSF('/usr/local/JCBroker/lib/jcb.jar')
   ```
6. 如果想使用 CBJ 类（并非以调试方式），则必须将 JCBroker\lib 和 JCBroker\lib\jcb.jar 添加到 Java 命令行的类路径选项中。不建议将 JCBroker\lib 和 JCBroker\lib\jcb.jar 添加到 CLASSPATH 环境变量中，因为当您运行 ARCHIVE 标记中设置了 applet_jcb.jar 的 applet 时，这可能会与类装入相冲突。
7. 如果要以调试方式运行 CBJ，则在类路径中用 jcbd.jar 来替换 jcb.jar，或者在 Java 命令行上覆盖类路径值。也可以使用此命令:

   ```
   CRTJVAPGM CLSF('/usr/local/JCBroker/lib/jcbd.jar')
   ```
8. 有关 CBJ API、运行演示程序、编辑属性、设计和编写 CBJ 应用程序的更多信息以及其它主题，请参考 cpj_1.1.jar 包中的 JCBroker/index.html 文件。

在 Windows 或 UNIX 上安装 Class Broker for Java

要在 Windows(R) 上安装 Class Broker for Java(TM)（CBJ），请执行下列步骤:

1. 确保正确安装了 JDK/JRE1.1 或 JDK/JRE1.2。
2. 选择一个目录来存储名为 cbj_1.1.jar 的 CBJ 包。例如，C:\。
3. 转至该目录（C:\），并输入以下命令:

   ```
   C:\ > jar xvf cbj_1.1.jar
   ```

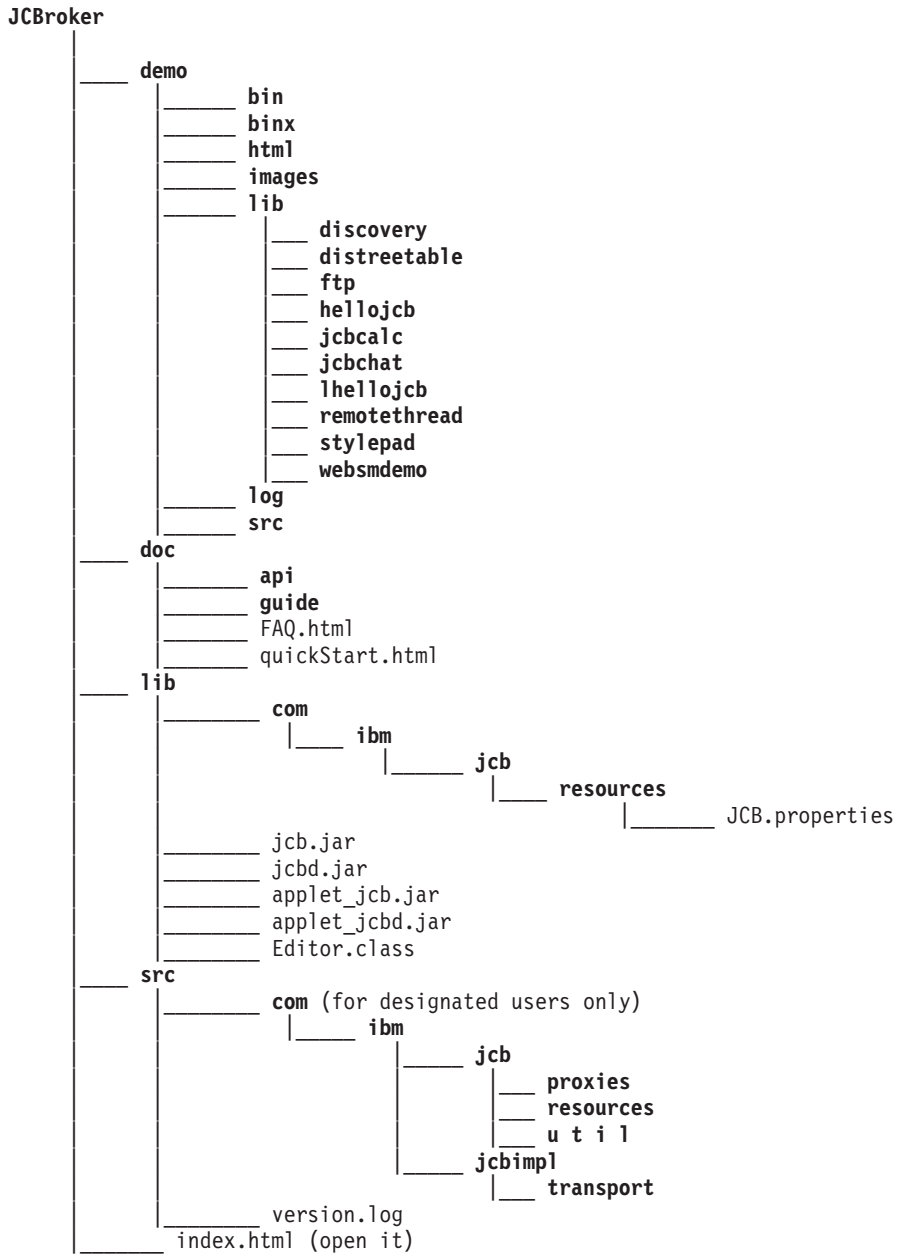
   CBJ 文件便被抽取和复制到该目录中。有关更多信息，参见 cbj_.1.1.jar 的包内容。
4. 如果要使用 CBJ 类（并非以调试方式），请将 C:\JCBroker\lib 和 C:\JCBroker\lib\jcb.jar 添加到 Java 命令行上的类路径选项中。

   注意: 不建议将它添加到系统 CLASSPATH 环境变量中，因为当您运行 ARCHIVE 标记中设置了 applet_jcb.jar 的 applet 时，这可能会与类装入相冲突。
5. 要以调试方式运行 CBJ，请在类路径中用 jcbd.jar 来替换 jcb.jar 。

   要在 UNIX(R) 上安装 CBJ，遵循在 Windows 上安装时相同的步骤，只有下列依赖于系统的更改除外:
   - 用 UNIX 文件分隔符 “/” 替换 Windows 文件分隔符 “\”。
   - 用 UNIX 类路径分隔符 “:” 替换 Windows 类路径分隔符 “;”。
   - 用 UNIX 系统环境变量 “$XXX” 替换 Windows 系统环境变量 “%XXX%”。
6. 有关 CBJ API、运行演示程序、编辑属性、设计和编写 CBJ 应用程序的更多信息以及其它主题，请参考 cpj_1.1.jar 包中的 JCBroker/index.html 文件。

cbj_1.1.jar 的包内容

以下是 Class Broker for Java(TM)（CBJ）包（cbj_1.1.jar）。粗体项指示子目录的主目录。非粗体项指示文件。



JCBroker 目录下面的下一级目录由下列数据组成:

demo

此目录包含这些子目录: bin、binx、html、images、lib、log 和 src。这些目录包含数个演示应用程序的 Windows(R) .exe 文件、UNIX(R) .exe、HTML 页面、GIF 文件、类文件和源代码（对于某些示例而言）。您可以通过遵循演示指示信息来运行这些演示示例。log 目录是调试作业记录文件的位置。

doc

此目录包含这两个子目录: api 和 guide。CBJ API 指南和关于 CBJ 的用户指南存储在这里。API 文档是使用 J2SDK 的 javadoc 工具生成的。要浏览 API 文档或用户指南，请在相应的目录中打开 index.html 文件。用户指南是介绍性层次的文档，而 QuickStart 指南则是快速入门文档。

此目录还包含 FAQ.html 文件和 quickStart.html 文件，其中，quickStart.html 文件说明如何安装和使用 CBJ 并显示如何运行演示示例。

lib

此目录包含开发、运行和部署新的基于 CBJ 的应用程序所需的 CBJ 类和资源。类封装在 jcb.jar 文件中，对于调试方式，类封装在 jcbd.jar 文件中。Applet 的 ARCHIVE 标记中包括的类封装在 applet_jcb.jar 文件中。对于调试方式，类封装在 applet_jcbd.jar 文件中。名为 JCB.properties 的 Java Class Broker（JCB）属性文件位于 com/ibm/jcb/resources 子目录中。当 CBJ 运行时启动时，会读取它。您可以更改此文件以反映您的喜好设置。安装和设置部分说明了这些属性。只适用于 applet 的那些属性包含在 applet_JCB.properties 文件中，该文件是 applet_jcb.jar 和 applet_jcbd.jar 的一部分。当以 applet 方式运行时，您也可以更改此文件以反映您的喜好设置。要更改此文件，请运行驻留在此目录中的 Editor Java 应用程序。

src

此目录包含 CBJ 源代码和名为 version.log 的内部版本作业记录文件。除了选择的包之外，此目录没有其它内容。

index.html

这是您参考本文档其余部分的起始页。

Native Abstract Windowing Toolkit

Native Abstract Windowing Toolkit（NAWT）为 Java(TM) 应用程序和 servlet 提供了使用 Java Development Kit（JDK）的 Abstract Windowing Toolkit（AWT）图形功能的能力。

可以在不需要直接用户交互作用的 Java 图形应用程序中使用 NAWT（发音为 “naw-TEE” ）。生成图像（如 JPEG 或 GIF 编码文件）或输出流的代码便是这样的应用程序的一个示例。对于此发行版，NAWT 支持 JDK V1.2 和 V1.3。

对于非交互式应用程序，还可使用 NAWT 来作为 Remote Abstract Windowing Toolkit（RAWT）的备用项。在 RAWT 中，图形计算由相连接的 PC 服务器执行。RAWT 可用于更一般的基于图形的应用程序类型，但需要远程 PC，并且通常会因为 iSeries 服务器与 PC GUI 服务器之间的网络等待时间而造成一定程度的性能开销。

NAWT 使用 X Window System 作为下层图形引擎，完全在 iSeries 服务器上执行图形计算。X Window System 是一个图形系统，它提供了用于显示图形的客户机／服务器基本部件。

有关在应用程序中安装和使用 NAWT 的更多信息，参见以下主题:

安装 NAWT
阅读如何在 iSeries 服务器上安装 NAWT 的信息，包括关于建议的 X Window 图形服务器、OS/400 PASE、需要的软件修订（PTF）和 iSeries Tools for Developers PRPQ 等信息。

有关使用 VNC 的提示
查明如何从 CL 程序中启动 VNC 显示服务器和如何结束 VNC 显示服务器。

有关 NAWT 使用的 VNC 服务器的更多信息，请参考 AT&T Research Virtual Network Computing Web 页面。

安装 NAWT

NAWT 依靠 X Window System 来作为它的图形引擎。本发行版的建议 X Window 图形服务器是“虚拟网络计算”（VNC），它在 OS/400 的“可移植应用程序解决方案环境”（PASE）之下运行。

PASE 是一个与 UNIX 相似的环境，它能够运行大多数为 IBM 的 AIX 操作系统编译的二进制可执行文件。VNC 是 AT&T Research Labs 的产品，它提供了*虚拟* X Window 服务器；即，不需要直接连接支持图形的显示设备的 X 服务器。

要安装和运行 Native Abstract Windowing Toolkit（NAWT），请执行下列任务:

1. 安装 OS/400 PASE
2. 安装 NAWT PTF
3. 安装 iSeries Tools for Developers PRPQ
4. 创建 VNC 密码文件
5. 配置 Java 系统属性
6. 启动 VNC 服务器
7. 设置环境变量
8. 验证安装过程

安装 OS/400 PASE: 订购并安装 OS/400 的“可移植应用程序解决方案环境”（PASE），5722SS1，选项 33。有关更多信息，参见 OS/400 PASE。

安装 NAWT PTF: 在安装任何软件修订（PTF）之前，通过执行以下步骤确保您具有与要使用的 JDK 版本相对应的许可程序 5722JV1 选项:

1. 在命令行上输入“转至许可程序”（GO LICPGM）命令。
2. 选择选项 10（显示已安装的许可程序）并验证是否已安装与要使用的 JDK 版本相对应的许可程序 5722JV1 选项。

   这些选项是:

JDK 版本	选项
JDK 1.2	57SSJV1 选项 3
JDK 1.3	57SSJV1 选项 5

应用最新的 Java 分组软件修订来获得任何新的 NAWT 修订。有关软件修订的更多信息，参见使用软件修订。

安装 iSeries Tools for Developers PRPQ: 安装 iSeries Tools for Developers PRPQ（5799PTL）。如果您没有 PRPQ，您必须订购它。

PRPQ 的更新版本包括“虚拟网络计算”（VNC）的预编译的启用 OS/400 PASE 的版本。旧版本不包括 VNC。如何安装 PRPQ 取决于您所拥有的版本:

- 对于在 2002 年 6 月 14 日或之后订购的 PRPQ 的版本，通过使用安装指示信息完成此任务，安装指示信息可在 Application Factory - iSeries Tools for Development Web 站点上获得。

注意: 要安装 PRPQ 中可用的 VNC 支持，仅遵循 Application Factory Web 站点上的安装指示信息。您无需遵循设置指示信息。

- 对于在 2002 年 6 月 14 日前订购的 PRPQ 版本，参考安装 iSeries Tools for Developers PRPQ 的旧版本来完成此任务。

创建 VNC 密码文件: 缺省情况下，VNC 需要一个密码文件，此文件保护 VNC 屏幕免遭未经授权的用户访问。如何创建加密密码取决于正在使用的 PRPQ 的版本:

- 对于在 2002 年 6 月 14 日或之后订购的 PRPQ 的版本，在 iSeries 命令提示符处使用以下命令:

```
MKDIR DIR('/home/your_profile_name/.vnc')
QAPTL/VNCPASSWD USEHOME(*NO) PWDFILE('/home/your_profile_name/.vnc/passwd')
```

- 对于在 2002 年 6 月 14 日前订购的 PRPQ 的版本，在 iSeries 命令提示符处使用以下命令:

```
MKDIR DIR('/home/your_profile_name/.vnc')VNCSAVF/VNCPASSWD USEHOME(*NO)
PWDFILE('/home/your_profile_name/.vnc/passwd')
```

仅启动 VNC 服务器的用户才需要具有密码文件。

配置 Java 系统属性

设置 Java 系统属性。第一行为期望的 JDK 版本（1.2 或 1.3）配置 Java，第二行启用 NAWT:

```
java.version=version
os400.class.path.rawt=2
```

其中 *version* 是 **1.2** 或 **1.3**，取决于您要使用的 JDK 的哪个版本。

要获取关于如何设置 Java 系统属性的帮助，参见为 IBM Developer Kit for Java 定制 iSeries 服务器。

启动 VNC 服务器

要启动 VNC 服务器，请执行下列步骤:

1. 启动 PASE 外壳程序:

   ```
   CALL QP2TERM
   ```

2. 从 PASE 外壳程序中，启动配置了 NAWT 的 VNC 服务器版本:

   ```
   /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
   ```

   当服务器启动时，您将看到一条消息，它类似于 “新的 ‘X’ 桌面是 *systemname:1*”。请记住这个桌面名，下一个步骤要用到此名称。

   注意: 如果启动另一个 VNC 服务器，则显示号（即冒号右边的编号）可能不同。每个并行运行的 VNC 服务器都需要唯一的显示号。当您未在对 vncserver_java 的调用上指定显示号时，vncserver_java 程序将查找可用的显示。通过使用以下命令启动 vncserver_java 程序来请求特定的显示:

   ```
   /QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java :n
   ```

   其中，*n* 是要使用的显示号。

设置环境变量

在任何将 Java 与 NAWT 配合运行的会话中，必须告诉 Java 在哪里查找 X 服务器以及在哪里查找 X 权限文件。通过设置环境变量来为 Java 提供此信息。

在要运行 Java 程序的会话中，将 DISPLAY 环境变量设置为系统名和显示号（即运行 vncserver_java 时打印的值）。

并将 XAUTHORITY 环境变量设置为 /home/*your_profile_name*/.Xauthority。

例如，在 iSeries 命令提示符处，输入以下命令:

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:1')
ADDENVVAR ENVVAR(XAUTHORITY) VALUE('/home/your_profile_name/.Xauthority')
```

注意:

- 当启动 VNC 服务器时，将创建或修改 .Xauthority 文件。X 服务器授权对 X 服务器连接强制安全性协议，以阻止其它用户的应用程序拦截 X 服务器请求。
- 仅实际启动 JVM 的用户才需要设置这些环境变量。例如，在 servlet 环境中，仅启动 servlet 引擎的用户才需要设置它们。

验证安装过程

使用以下命令运行 NAWT 测试 Java 应用程序来验证以上步骤是否成功完成:

```
JAVA CLASS(NAWTtest) CLASSPATH('/QIBM/ProdData/Java400/')
```

测试应用程序将在集成文件系统中创建名为 /tmp/NAWTtest.jpg

安装 iSeries Tools for Developers 的旧版本

在 2002 年 6 月 14 日之前订购的 iSeries Tools for Developers PRPQ（5799PTL）的版本不包括虚拟网络计算（VNC）的预编译 OS/400 启用 PASE 的版本。

使用以下指示信息来确定您是否具有增强型 PRPQ 以及在具有 PRPQ 的旧版本的情况下安装 VNC。

确定是否具有增强型 PRPQ: 如果您拥有 PRPQ 5799-PTL，但不能确定是否具有包含 VNC 的增强型版本，则检查以下文件是否存在:

```
/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java
```

PRPQ 的增强型版本包括 vncserver_java 文件，但旧版本不包括。如果在 iSeries 服务器上没有 vncserver_java，您可以订购并安装 PRPQ 的最新版本或使用以下指示信息来完成 VNC 安装。

安装 VNC: 要在 iSeries Tools for Developers PRPQ 的旧版本上安装 VNC，完成以下步骤。

1. 通过运行下列命令，在 iSeries 服务器上创建保存文件:

   ```
   crtlib vncsavf
   crtsavf vncsavf/vncpasswd
   crtsavf vncsavf/vnc
   crtsavf vncsavf/fonts
   crtsavf vncsavf/icewm
   ```

2. 单击下列列表中的链接来从 Application Factory - iSeries Tools for Development  Web 站点将保存文件下载到您的工作站:
   - vnc.savf
   - vncpasswd.savf
   - fonts.savf
   - icewm.savf
3. 通过在工作站运行以下命令，使用 FTP 将保存文件从工作站传送至 iSeries 服务器:

   ```
   ftp youriseriesserver
   bin
   cd /qsys.lib/vncsavf.lib
   put vnc.savf
   ```

```
put vncpasswd.savf
put fonts.savf
put icewm.savf
quit
```

4. 通过在 iSeries 服务器上运行下列命令，恢复保存文件:

```
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. 通过创建 VNC 密码文件继续安装 NAWT。«

有关使用 VNC 的提示

» 本页面讨论与“虚拟网络计算”（VNC）的使用相关的附加提示。

从 CL 程序启动 VNC 显示服务器

以下示例是一种设置 DISPLAY 环境变量和从 CL 程序自动启动 VNC 的方法。假定屏幕: *n* 没有在运行。并假定您已通过运行 VNCPASSWD 命令创建了 VNC 所需的密码文件。

```
ADDENVVAR ENVVAR(DISPLAY) VALUE('systemname:n')
call qp2shell parm('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' ':n')
```

其中，*n* 是表示要启动的显示号的数值。

结束 VNC 显示服务器

要结束已启动的 VNC 服务器:

```
call qp2shell parm('/QOpenSys/QIBM/ProdData/DeveloperTools/vnc/vncserver_java' '-kill' ':n')
```

其中，*n* 是表示要结束的显示号的数值。 «

第 2 章 Java 安全性

在 iSeries 服务器上运行的大多数 Java(TM) 程序是应用程序，而不是 applet，所以“砂箱”安全性模型对它们不起限制作用。从安全性的观点看，Java 应用程序所受的安全性限制与 iSeries 服务器上的任何其它程序相同。要在 iSeries 服务器上运行 Java 程序，您必须对集成文件系统中的类文件具有权限。程序一旦启动，它就在该用户权限控制下运行。

您可以使用沿用权限来访问具有运行程序的用户的权限和程序拥有者权限的对象。沿用权限临时地将用户原先无权访问的对象的权限授予用户。参见创建 Java 程序（CRTJVAPGM） 命令信息以了解有关两个新的沿用权限参数（USRPRF 和 USEADPAUT）的详细信息。

» IBM Developer Kit for Java 为 Java 应用程序提供了下列安全性功能:

Java 安全性模型
Java 虚拟机中的字节码装入程序和验证程序还提供了使用 Java 安全性模型来评测 Java 安全性的方法。就像 applet 一样，字节码装入程序和验证程序检查字节码是否有效，以及数据类型是否使用正确。它们还检查对寄存器和内存的访问是否正确，以及堆栈是否未溢出或下溢。这些检查确保 Java 虚拟机可以安全地运行类，而不会危及系统的完整性。

Java 密码术扩展
iSeries 服务器上的“Java 密码术扩展”（JCE）实现与 Sun Microsystems 的实现相兼容。本文档阐述了 iSeries 实现的独特方面。我们假定您熟悉与 JCE 相关的一般文档。

Java 安全套接字扩展
Java 安全套接字扩展（JSSE）是安全套接字层（SSL）协议的 Java 实现。JSSE 使用 SSL 和“传输层安全性”（TLS）协议使客户机和服务器能够基于 TCP/IP 进行安全通信。此文档涵盖了 JSSE 的 iSeries 实现的几个独特方面。假定您熟悉 JSSE 的一般文档。

Java 认证和授权服务
“Java 认证和授权服务”（JAAS）是 IBM Developer Kit for Java 支持的另一项安全性。当前，Java 2 Software Development Kit（J2SDK）Standard Edition 提供了基于代码起源位置与代码签名人的访问控制（基于代码源的访问控制）。然而，它不包含强制实施基于代码运行者的附加访问控制的能力。JAAS 提供了一个框架，用于将此项支持添加至 Java 2 安全性模型。

Java 一般安全性服务
“Java 一般安全性服务”（JGSS）是 IBM Developer Kit for Java 支持的另一项安全性。JGSS 提供一个在应用程序之间进行安全消息传递的一般接口。JGSS 支持各种基于保密密钥、公用密钥或其它安全性技术的安全性机制。

注意: 对于 J2SDK V1.4，JAAS、JCE、JGSS 和 JSSE 是基本 JDK 的一部分，而不认为是其扩展。对于先前的 JDK 版本，这些安全性项是扩展。 «

Java 安全性模型

您可以从任何系统下载 Java(TM) applet; 因而 Java 虚拟机中存在安全性机制，可保护您免遭恶意 applet 的攻击。当 Java 虚拟机装入字节码时，Java 运行时系统便对它们执行验证。这将确保它们是有效的字节码，且代码不违反 Java 虚拟机对 Java applet 所加的任何限制。Java applet 在它们可以执行什么操作、如何访问内存以及如何使用 Java 虚拟机方面受到限制。限制的目的在于防止 Java applet 获取对系统上的下层操作系统或数据的访问权。因为 Java applet 只能在它自己的砂箱中“运行”，所以这是“砂箱”安全性模型。

“砂箱”安全性模型是类装入程序、类文件验证程序和 java.lang.SecurityManager 类的组合。

有关安全性的更多信息，参见 Sun Microsystems 的安全性文档和使用 SSL 来保证应用程序的安全。

Java 密码术扩展

“Java(TM) 密码术扩展”（JCE）1.2 是对 Java 2 Software Development Kit（J2SDK）Standard Edition 的标准扩展。iSeries 服务器上的 JCE 实现与 Sun Microsystems 的实现相兼容。本文档阐述了 iSeries 实现的独特方面。我们假定您熟悉与 JCE 扩展相关的一般文档。为了便于您使用该信息以及我们的 iSeries 信息，我们提供了至 Sun JCE 文档  的链接。

在 iSeries 服务器上，加密级别由“Cryptographic Access Provider 产品”控制。这个产品有两个版本可用，即 5722-AC2 和 5722-AC3。5722-AC3 产品支持所有加密算法。5722-AC2 产品不允许 Triple-DES，并将对称算法限制为 56 位，将非对称算法限制为 1024 位。

除了已提到的有关 5722-AC2 的限制之外，“IBM JCE 提供程序”支持下列算法:

- DES
- Triple-DES
- RC2
- RC4
- Blowfish
- RSA
- Diffie-Hellman
- DSA
- Mars
- MD2
- MD5
- SHA-1
- Seal

另外，它还提供了随机数生成器。

» 如果要将 IBM JCE 与 Java 1.2 配合使用，请编辑 /QIBM/ProdData/Java400/jdk12/lib/security/java.security 文件。 « 该文件需要更改的部分如下。

```
#
# To use the IBMJCE security provider, you need to:
# 1) Install an IBM Cryptographic Access Provider Product
# 2) uncomment the second provider entry that follows.
#
# List of providers and their preference orders:
#
security.provider.1=sun.security.provider.Sun
#security.provider.2=com.ibm.crypto.provider.IBMJCE
```

» 如果要将 IBM JCE 与 Java 1.3 配合使用，请编辑 /QIBM/ProdData/OS400/Java400/jdk/lib/security/java.security 文件。 « 该文件需要更改的部分如下。

```
#
# 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
```

在这两种情况下都只需要删除一个字符。

Java 安全套接字扩展

Java(TM) 安全套接字扩展（JSSE）是安全套接字层（SSL）协议的 Java 实现。JSSE 使用 SSL 和“传输层安全性”（TLS）协议使客户机和服务器能够基于 TCP/IP 进行安全通信。

JSSE 提供以下功能:

- 对数据进行加密
- 认证远程用户标识
- 认证远程系统名
- 执行客户机／服务器认证
- 确保消息完整性

由于已集成到 Java 2 Software Development Kit Standard Edition（J2SDK）V1.4 中，JSSE 提供的功能比 SSL 单独提供的功能更多。有关更多信息，参见下列主题:

使用 SSL（JSSE V1.0.8）
SSL 通过提供一种认证服务器和客户机的方法来提供保密性和数据完整性。所有 SSL 通信都从服务器和客户机之间的“握手”开始。在握手期间，SSL 协商客户机和服务器用来相互通信的密码套件。此密码套件是可以通过 SSL 获得的各种安全性属性的组合。可以将 SSL 与 J2SDK V1.4 之前的 JDK 的任何受支持的版本配合使用。

使用 JSSE V1.4
JSSE 类似于一个框架，它使 SSL 和 TLS 的下层机制抽象化。通过使下层协议的复杂性和特质抽象化，JSSE 使程序员能够使用安全的加密通信，而同时使可能的安全性弱点最小化。此信息仅适用于在运行 J2SDK V1.4 的 iSeries 服务器上使用 JSSE。

注意: 此信息与 JSSE 的版本有关，现在已将 JSSE 封装到 J2SDK V1.4 中。对于 JSSE 的先前版本，参见 Sun Java Web 站点上的 Java 安全套接字扩展 。  

使用 SSL（JSSE V1.0.8）

可以使用 Java 安全套接字扩展（JSSE V1.0.8）（它是安全套接字层（SSL）的 Java 实现）来使 Java(TM) 应用程序更安全。SSL 执行下列各项来改进应用程序的安全性:

- 通过加密来保护通信数据。
- 认证远程用户标识。
- 认证远程系统名。

注意: SSL 使用数字证书来对 Java 应用程序的套接字通信进行加密。数字证书是用来标识安全系统、用户和应用程序的因特网标准。可以使用 “IBM 数字证书管理程序” 来控制数字证书。有关更多信息，参见 IBM 数字证书管理器。

要通过使用 SSL 来使 Java 应用程序更安全:

- 准备 iSeries 服务器以支持 SSL。
- 通过执行下列各项，将 Java 应用程序设计成使用 SSL:
  - 如果尚未使用套接字生成器，则将 Java 套接字代码更改为使用套接字生成器。
  - 将 Java 代码更改为使用 SSL。
- 通过执行下列各项，使用数字证书来使 Java 应用程序更安全:
  1. 选择要使用的数字证书类型。
  2. 在运行应用程序时使用数字证书。

还可以使用 QsyRegisterAppForCertUse API 来将 Java 应用程序注册为安全应用程序。有关更多信息，参见 QsyRegisterAppForCertUse。

» 有关 SSL 的 Java 版本的更多信息，参见 Sun Microsystems 的 Java 安全套接字扩展  。 «

为安全套接字层支持准备 iSeries 服务器

要为了使用安全套接字层（SSL）而准备系统，需要安装 “数字证书管理器” LP:

- » 5722-SS1 OS/400 — 数字证书管理器 «

还需要安装下列 Cryptographic Access Provider LP 之一:

- » 5722-AC1 Cryptographic Access Provider 40 位
- 5722-AC2 Cryptographic Access Provider 56 位
- 5722-AC3 Cryptographic Access Provider 128 位 «

您还需确保可以在系统上访问或创建数字证书。有关 iSeries 数字证书管理和因特网的更多信息，参阅 IBM 数字证书管理器入门。

Cryptographic Access Provider

Cryptographic Access Provider 提供了许多密码套件供系统使用。密码套件是各种安全性功能部件的组合。以下列表显示了每个 Cryptographic Access Provider 提供的密码套件:

» 5722-AC1 Cryptographic Access Provider 40 位 «

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

» 5722-AC2 Cryptographic Access Provider 56 位 «

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_DES_CBC_MD5
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5

» 5722-AC3 Cryptographic Access Provider 128 位 «

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC2_CBC_128_MD5
SSL_RSA_WITH_DES_CBC_MD5
SSL_RSA_WITH_3DES_EDE_CBC_MD5

根据您所在的国家或地区的不同，您可以选择的 Cryptographic Access Provider 可能会有所限制。在装入 Cryptographic Access Provider 之后，便可以使用该 Access Provider 提供的任何密码套件。

将 Java 代码更改为使用套接字生成器

要将安全套接字层（SSL）与现有代码配合使用，首先必须将代码更改为使用套接字生成器。

要将代码更改为使用套接字生成器，请执行下列步骤:

1. 将下面这一行添加至程序以导入 SocketFactory 类:

   ```
   import javax.net.*;
   ```

2. 添加声明 SocketFactory 对象实例的代码行。例如:

   ```
   SocketFactory socketFactory
   ```

3. 通过将 SocketFactory 实例设置为等于方法 SocketFactory.getDefault() 来对该实例进行初始化。例如:

   ```
   socketFactory = SocketFactory.getDefault();
   ```

   SocketFactory 的完整声明应类似于:

   ```
   SocketFactory socketFactory = SocketFactory.getDefault();
   ```

4. 初始化现有套接字。对所声明的每个套接字，对套接字生成器调用 SocketFactory 方法 createSocket(host,port)。

   套接字声明现在应类似于:

   ```
   Socket s = socketFactory.createSocket(host,port);
   ```

   其中:

   - *s* 是正在创建的套接字。
   - *socketFactory* 是步骤 2 中创建的 SocketFactory。
   - *host* 是表示主机服务器名的字符串变量。
   - *port* 是表示套接字连接的端口号的整型变量。

完成所有这些步骤之后，代码便开始使用套接字生成器了。无需对代码作任何其它更改。您调用的所有方法及所有套接字语法仍起作用。

有关转换为使用套接字生成器的客户机程序的示例，参见示例: 将 Java(TM) 代码更改为使用服务器套接字生成器。

有关转换为使用套接字生成器的客户机程序的示例，参见示例: 将 Java 代码更改为使用服务器套接字生成器。

示例: 将 Java 代码更改为使用服务器套接字生成器

这些示例显示了如何更改名为 simpleSocketServer 的简单套接字类，以使它使用套接字生成器来创建所有套接字。第一个示例显示不带套接字生成器的 simpleSocketServer 类。第二个示例显示带有套接字生成器的 simpleSocketServer 类。在第二个示例中，将 simpleSocketServer 重命名为 factorySocketServer。

示例 1: 不带套接字生成器的套接字服务器程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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
```

示例 2: 带有套接字生成器的简单套接字服务器程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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();
  }

}
```

有关背景信息，参见将 Java(TM)代码更改为使用套接字生成器。

示例: 将 Java 代码更改为使用客户机套接字生成器

这些示例显示了如何更改名为 simpleSocketClient 的简单套接字类，以使它使用套接字生成器来创建所有套接字。第一个示例显示不带套接字生成器的 simpleSocketClient 类。第二个示例显示带有套接字生成器的 simpleSocketClient 类。在第二个示例中，将 simpleSocketClient 重命名为 factorySocketClient。

示例 1: 不带套接字生成器的套接字客户机程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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.
```

示例 2: 带有套接字生成器的简单套接字客户机程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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.
```

有关背景信息，参见将 Java(TM)代码更改为使用套接字生成器。

将 Java 代码更改为使用安全套接字层

如果代码已使用套接字生成器来创建其套接字，则可以对程序添加安全套接字层（SSL）支持。如果代码尚未使用套接字生成器，则参见将 Java(TM)代码更改为使用套接字生成器。

要将代码更改为使用 SSL，请执行下列步骤:

1. 导入 javax.net.ssl.* 以添加 SSL 支持:

   ```
   import javax.net.ssl.*;
   ```

2. 通过使用 SSLSocketFactory 初始化 SocketFactory 来对其进行声明:

   ```
   SocketFactory newSF = SSLSocketFactory.getDefault();
   ```

3. 使用新的 SocketFactory 来初始化套接字，方法与使用旧 SocketFactory 初始化套接字的方法相同:

   ```
   Socket s = newSF.createSocket(args[0], serverPort);
   ```

代码现在便开始使用 SSL 支持了。无需对代码作任何其它更改。

参见示例: 将 Java 客户机更改为使用安全套接字层和示例: 将 Java 服务器更改为使用安全套接字层以获取示例代码。

示例: 将 Java 服务器更改为使用安全套接字层

这些示例显示如何将一个名为 factorySocketServer 的类更改为使用安全套接字层（SSL）。

第一个示例显示不使用 SSL 的 factorySocketServer 类。第二个示例显示同一个类，将其重命名为 factorySSLSocketServer，并使用 SSL。

示例 1: 不带 SSL 支持的简单 factorySocketServer 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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();
  }
}
```

示例 2: 带有 SSL 支持的简单 factorySocketServer 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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();
  }
}
```

示例: 将 Java 客户机更改为使用安全套接字层

这些示例显示如何将一个名为 factorySocketClient 的类更改为使用安全套接字层（SSL）。

第一个示例显示不使用 SSL 的 factorySocketClient 类。第二个示例显示同一个类，将其重命名为 factorySSLSocketClient 并使用 SSL。

示例 1: 不带 SSL 支持的简单 factorySocketClient 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/* 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.
```

示例 2: 带有 SSL 支持的简单 factorySocketClient 类

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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.
```

有关背景信息，参见将 Java(TM) 代码更改为使用安全套接字层。

选择要使用的数字证书

当确定要使用哪个数字证书时，您应考虑几个因素。您可以使用系统的缺省证书，也可以指定使用另一个证书。

在下列情况下，您将希望使用系统的缺省证书:

- Java(TM) 应用程序没有任何特定的安全性需求。
- 您不知道 Java 应用程序需要哪种安全性。
- 系统的缺省证书已符合 Java 应用程序的安全性需求。

注意: 如果您确定要使用系统的缺省证书，则请与系统管理员一起检查，确保已创建缺省系统证书。有关数字证书管理的更多信息，参见 IBM 数字证书管理器入门。

如果不想使用系统的缺省证书，则您需要选择另一个要使用的证书。可从两类证书中进行选择:

- **用户证书**，它标识应用程序的用户。
- **系统证书**，它标识应用程序运行时所在的系统。

» 在下列情况下，应使用用户证书:

- 应用程序作为客户机应用程序运行。
- 您想让证书标识正在使用应用程序的用户。

在下列情况下，应使用系统证书:

- 应用程序作为服务器应用程序运行。
- 您想让证书标识运行应用程序所在的系统。 «

在您知道需要哪种证书之后，就可从您能够访问的任何证书容器中的任何数字证书中进行选择。

在运行 Java 应用程序时使用数字证书

要使用安全套接字层（SSL），必须使用数字证书来运行 Java 应用程序。

要指定所要使用的数字证书，请使用下列属性:

- os400.certificateContainer
- os400.certificateLabel

例如，如果想要使用数字证书 MYCERTIFICATE 来运行 Java 应用程序 MyClass.class，且 MYCERTIFICATE 在数字证书容器 YOURDCC 中，则 java 命令将类似于:

```
java -Dos400.certificateContainer=YOURDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

如果尚未确定要使用哪一个数字证书，则请参见选择要使用的数字证书。您也可以决定使用系统的缺省证书，它存储在系统的缺省证书容器中。

要使用系统的缺省数字证书，您无需在任何地方指定证书或证书容器。Java 应用程序将自动使用系统的缺省数字证书。

有关 iSeries 数字证书管理和因特网的更多信息，参阅 IBM 数字证书管理器入门。

数字证书和 -os400.certificateLabel 属性: 数字证书是用来标识安全系统、用户和应用程序的因特网标准。数字证书存储在数字证书容器中。如果想要使用数字证书容器的缺省证书，则无需指定证书标号。如果想要使用特定数字证书，则必须在 java 命令中使用以下属性来指定证书的标号:

```
os400.certificateLabel=
```

例如，如果想要使用的证书的名称是 MYCERTIFICATE，则输入的 java 命令将类似于:

```
java -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

在此示例中，Java 应用程序 MyClass 将使用证书 MYCERTIFICATE。MYCERTIFICATE 将需要存在于系统的缺省证书容器中才能由 MyClass 使用。

数字证书容器和 -os400.certificateContainer 属性: 数字证书容器存储数字证书。如果要使用 iSeries 系统缺省证书容器，则无需指定证书容器。要使用特定数字证书容器，您需要在 java 命令中使用以下属性来指定数字证书容器:

```
os400.certificateContainer=
```

例如，如果包含要使用的数字证书的证书容器名为 MYDCC，则您输入的 java 命令将类似于:

```
java -Dos400.certificateContainer=MYDCC MyClass
```

在本示例中，名为 MyClass.class 的 Java 应用程序将使用名为 MYDCC 的数字证书容器中的缺省数字证书来在系统上运行。您在该应用程序中创建的任何套接字都使用 MYDCC 中的缺省证书来标识它们本身，并保证它们的所有通信的安全。

如果要使用数字证书容器中的数字证书 MYCERTIFICATE，则输入的 java 命令将类似于:

```
java -Dos400.certificateContainer=MYDCC
     -Dos400.certificateLabel=MYCERTIFICATE MyClass
```

使用 Java 安全套接字扩展 V1.4

Java 安全套接字扩展（JSSE）使用安全套接字层（SSL）协议和传输层安全性（TLS）协议来在客户机与服务器之间提供安全的加密通信。

JSSE 的 IBM 实现称为 IBM JSSE。IBM JSSE 包括一个本机 iSeries JSSE 提供程序和一个纯 Java JSSE 提供程序。

有关配置 iSeries 服务器以支持 JSSE 的更多信息，使用以下链接:

配置服务器以支持 JSSE
查明如何配置 iSeries 服务器以使用 IBM JSSE。信息包括软件需求、如何更改 JSSE 提供程序、必要的安全性属性和系统属性。

使用本机 iSeries JSSE 提供程序
了解如何使用 JSSE KeyStore 类和 SSLConfiguration 类的本机 iSeries 实现。

JSSE 示例
使用示例程序来发现如何在应用程序中使用 JSSE。示例 Java 源代码显示客户机和服务器如何使用其上的 SSLContext 对象来创建安全通信环境。 

将 iSeries 服务器配置为支持 JSSE

当在 iSeries 服务器上使用 Java 2 Software Development Kit（J2SDK）V1.4 时，则已配置 JSSE。缺省配置使用本机 iSeries JSSE 提供程序。

软件需求: 要将 JSSE 与 J2SDK V1.4 配合使用，必须已在 iSeries 服务器上安装 IBM Cryptographic Access Provider 128 位（5722-AC3）。有关更多信息，参见 Cryptographic Access Provider。

更改 JSSE 提供程序: 可以将 JSSE 配置为使用纯 Java JSSE 提供程序来代替本机 iSeries JSSE 提供程序。通过更改某些特定 JSSE 安全性属性和 Java 系统属性，可以在两个提供程序之间进行切换。有关更多信息，参见下列主题:

- JSSE 提供程序
- JSSE 安全性属性
- Java 系统属性

安全性管理器: 如果在启用 Java 安全性管理器的情况下运行 JSSE 应用程序，可能需要设置可用的网络许可权。有关更多信息，参见 Java 2 SDK中的许可权  中的 SSLPermission。

JSSE 提供程序

IBM JSSE 包括一个本机 iSeries JSSE 提供程序和一个纯 Java JSSE 提供程序。您选择使用的提供程序取决于应用程序的需要。

两个 JSSE 提供程序都遵守 JSSE 接口规范。两个提供程序可以互相通信并与任何其它 SSL 或 TLS 实现（甚至非 Java 实现）通信。

纯 Java JSSE 提供程序: 纯 Java JSSE 提供程序提供以下功能:

- 使用任何类型的 KeyStore 对象来控制和配置数字证书（例如，JKS、PKCS12 等）
- 允许您将来自多个实现的 JSSE 组件的任何组合一起使用（例如，可以使用来自本机 iSeries JSSE 提供程序的 X509TrustManager 来初始化来自此纯 Java JSSE 提供程序的 SSLContext）。

IBMJSSE 是纯 Java 实现的提供程序名。您需要使用正确的大小写将此提供程序名传送至 java.security.Security.getProvider() 方法或几个 JSSE 类的各种 getInstance() 方法。

本机 iSeries JSSE 提供程序: 本机 iSeries JSSE 提供程序提供以下功能:

- 使用本机 iSeries SSL 支持
- 使用“数字证书管理器”来配置和控制数字证书
- 提供最佳性能

注意: 本机 iSeries JSSE 提供程序需要 KeyStore 的唯一 iSeries 类型。JSSE 本机 iSeries JSSE 提供程序也不允许将来自任何其它实现的组件插入它的实现。

IbmISeriesSslProvider 是本机 iSeries 实现的名称。您需要使用正确的大小写将此提供程序名传送至 java.security.Security.getProvider() 方法或几个 JSSE 类的各种 getInstance() 方法。

更改缺省 JSSE 提供程序: 通过对安全性属性进行适当的更改，可以更改缺省 JSSE 提供程序。有关更多信息，参见以下主题:

- JSSE 安全性属性

在更改 JSSE 提供程序之后，确保系统属性为新的提供程序所需要的数字证书信息（keystore）指定正确的配置。有关更多信息，参见以下主题:

- Java 系统属性

JSSE 安全性属性

Java 虚拟机（JVM）使用许多重要的安全性属性，您可以通过编辑 Java 主安全性属性文件设置它们。此文件（名为 java.security）通常驻留在 iSeries 服务器上的 /QIBM/ProdData/Java400/jdk14/lib/security 目录中。

下面的列表描述几个与使用 JSSE 有关的安全性属性。使用描述作为编辑 java.security 文件的指南。

security.provider.<integer>

要使用的 JSSE 提供程序。还静态地注册加密提供程序类。严格按照以下示例来指定不同的 JSSE 提供程序:

```
security.provider.5=com.ibm.as400.ibmonly.net.ssl.Provider
security.provider.6=com.ibm.jsse.IBMJSSEProvider
```

ssl.KeyManagerFactory.algorithm

指定缺省 KeyManagerFactory 算法。对于本机 iSeries JSSE 提供程序，使用:

```
ssl.KeyManagerFactory.algorithm=IbmISeriesX509
```

对于纯 Java JSSE 提供程序，使用:

```
ssl.KeyManagerFactory.algorithm=IbmX509
```

有关更多信息，参见 javax.net.ssl.KeyManagerFactory 的 javadoc。

ssl.TrustManagerFactory.algorithm

指定缺省 TrustManagerFactory 算法。对于本机 iSeries JSSE 提供程序，使用:

```
ssl.TrustManagerFactory.algorithm=IbmISeriesX509
```

对于纯 Java JSSE 提供程序，使用:

```
ssl.TrustManagerFactory.algorithm=IbmX509
```

有关更多信息，参见 javax.net.ssl.TrustManagerFactory 的 javadoc。

ssl.SocketFactory.provider

指定缺省 SSL 套接字生成器。对于本机 iSeries JSSE 提供程序，使用:

```
ssl.SocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLSocketFactoryImpl
```

对于纯 Java JSSE 提供程序，使用:

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
```

有关更多信息，参见 javax.net.ssl.SSLSocketFactory 的 javadoc。

ssl.ServerSocketFactory.provider

指定缺省 SSL 服务器套接字生成器。对于本机 iSeries JSSE 提供程序，使用:

```
ssl.ServerSocketFactory.provider=com.ibm.as400.ibmonly.net.ssl.SSLServerSocketFactoryImpl
```

对于纯 Java JSSE 提供程序，使用:

```
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

有关更多信息，参见 javax.net.ssl.SSLServerSocketFactory 的 javadoc。

JSSE Java 系统属性

要在应用程序中使用 JSSE，需要指定几个系统属性，缺省 SSLContext 对象需要这些属性以便提供对配置的确认。某些属性适用于两个提供程序，而其它属性仅适用于本机 iSeries 提供程序。

当使用本机 iSeries JSSE 提供程序时，如果不指定这些属性，os400.certificateContainer 缺省为 *SYSTEM，这意味着 JSSE 使用系统证书存储库中的缺省项。

适用于两个提供程序的属性: 以下属性适用于两个提供程序。每个描述包括缺省属性（如果适用）。

javax.net.ssl.trustStore

包含您想要缺省 TrustManager 使用的 KeyStore 对象的文件的名称。缺省值是 jssecacerts 或 cacerts（如果 jssecacerets 不存在）。

javax.net.ssl.trustStoreType

您想要缺省 TrustManager 使用的 KeyStore 对象的类型。缺省值是 KeyStore.getDefaultType 方法返回的值。

javax.net.ssl.trustStorePassword

您想要缺省 TrustManager 使用的 KeyStore 对象的密码。

javax.net.ssl.keyStore

包含您想要缺省 KeyManager 使用的 KeyStore 对象的文件的名称。

javax.net.ssl.keyStoreType

包含您想要缺省 KeyManager 使用的 KeyStore 对象的类型。缺省值是 KeyStore.getDefaultType 方法返回的值。

javax.net.ssl.keyStorePassword

您想要缺省 KeyManager 使用的 KeyStore 对象的密码。

仅适用于 iSeries 本机 JSSE 提供程序的属性: 以下属性仅适用于本地 iSeries JSSE 提供程序。

os400.secureApplication

应用程序标识符。仅当未指定以下任何属性时，JSSE 才使用此属性:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStoreType
- and javax.net.ssl.keyStorePassword

os400.certificateContainer

要使用的密钥环的名称。仅当未指定以下任何属性时，JSSE 才使用此属性:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStoreType
- javax.net.ssl.keyStorePassword
- os400.secureApplication

os400.certificateLabel

要使用的密钥环标号。仅当设置并使用 os400.certificateContainer 属性时，JSSE 才使用此属性。

附加信息: 有关系统属性的更多信息，参见下列主题:

- iSeries 服务器上的 J2SDK V1.4 的 Java 系统属性
- Sun Java Web 站点上的 System Properties 

使用本机 iSeries JSSE 提供程序

本机 iSeries JSSE 提供程序提供一套完整的 JSSE 类和接口。要有效地使用本机 iSeries 提供程序，请参考以下信息:

- SSLContext.getInstance 方法的协议值
- 本机 KeyStore 实现
- 使用本机 iSeries 提供程序时的限制
- SSLConfiguration 的 Javadoc 信息

SSLContext.getInstance 方法的协议值: 下表标识并描述本机 iSeries JSSE 提供程序的 SSLContext.getInstance 方法的协议值。

协议值	受支持的 SSL 协议
SSL	SSL V2、SSL V3 和 TLS V1

协议值	受支持的 SSL 协议
SSLv2	SSL V2
SSLv3	SSL V3
TLS	SSL V2、SSL V3 和 TLS V1
TLSv1	TLS V1
SSL_TLS	SSL V2、SSL V3 和 TLS V1

本机 iSeries KeyStore 实现: 本机 iSeries 提供程序提供类型 IbmISeriesKeyStore 的 KeyStore 类的实现。此 keystore 实现提供“数字证书管理器”支持的包装器。keystore 的内容基于特定应用程序标识符或密钥环文件、密码和标号。JSSE 从“数字证书管理器”装入密钥环项。为了装入这些项，当应用程序首次尝试访问 keystore 项或 keystore 信息时，JSSE 使用适当的应用程序标识符或密钥环信息。不能修改 keystore，必须通过使用“数字证书管理器”来进行所有配置更改。

有关使用“数字证书管理器”的更多信息，参见下列主题:

数字证书管理器

当使用本机 iSeries 提供程序时的限制: 要本机 iSeries JSSE 提供程序运行，JSSE 应用程序必须仅使用来自本机实现的组件。例如，启用 JSSE 的本机 iSeries 应用程序不能使用 X509KeyManager 对象（该对象是使用纯 Java JSSE 提供程序创建的）来成功地初始化通过使用本机 iSeries JSSE 提供程序创建的 SSLContext 对象。

另外，必须通过使用 IbmISeriesKeyStore 对象或 com.ibm.as400.SSLConfiguration 对象来初始化本机 iSeries 提供程序中的 X509KeyManager 和 X509TrustManager 的实现。

注意: 先前提及的限制在以后的发行版中可能会更改，以便本机 iSeries JSSE 提供程序可以允许您插入非本机组件（例如，JKS KeyStore 或 IbmX509 TrustManagerFactory）。

示例: IBM Java 安全套接字扩展

JSSE 示例显示客户机和服务器如何使用本机 iSeries JSSE 提供程序来创建启用安全通信的上下文。

注意: 两个示例都使用本机 iSeries JSSE 提供程序，而与 java.security 文件指定的属性无关。

示例: 使用 SSLContext 对象的 SSL 客户机

此示例客户机程序利用 SSLContext 对象，该程序将此对象初始化为使用 “MY_CLIENT_APP” 应用程序标识。此程序将使用本机 iSeries 实现，而与在 java.security 文件中指定的内容无关。

示例: 使用 SSLContext 对象的 SSL 服务器

以下服务器程序利用它用先前创建的 keystore 文件初始化的 SSLContext 对象。keystore 文件的名称为 `/home/keystore.file`，keystore 密码为 `password`。

示例程序需要 keystore 文件才能创建 IbmISeriesKeyStore 对象。KeyStore 对象必须指定 MY_SERVER_APP 作为应用程序标识符。

要创建 keystore 文件，可以使用以下任何一个命令:

- 从 Qshell 命令提示符:

```
java com.ibm.as400.SSLConfiguration -create -keystore /home/keystore.file
  -storepass password -appid MY_SERVER_APP
```

有关通过 Qshell 使用 Java 命令的更多信息，参见下列主题:

Qshell

- 从 iSeries 命令提示符:

```
RUNJVA CLASS(com.ibm.as400.SSLConfiguration) PARM('-create' '-keystore'
     '/home/keystore.file' '-storepass' 'password' '-appid' 'MY_SERVER_APP')
```

以下不保证声明适用于所有 IBM JSSE 示例:

代码示例不保证声明

IBM 授予您使用所有编程代码示例的非专有版权许可证，您可以由此生成相似的定制功能以满足您特定的需要。

IBM 提供所有样本代码只是出于解释的目的。并未在所有环境下完全测试这些示例。因此，IBM 不保证或默示这些程序的可靠性、可服务性和功能。

本文档中包含的所有程序是以“按现状”的基础提供的，不附有任何形式的保证。明示的不保证声明包括非侵权性、适销性和适用于某特定用途的默示保证。

示例: 使用 SSLContext 对象的 SSL 客户机

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
////////////////////////////////////////////////////////////////////////////
//
// 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();
        }
    }

}
```

示例: 使用 SSLContext 对象的 SSL 服务器

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
////////////////////////////////////////////////////////////////////////////
//
// 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();
        }
    }

}
```

第 3 章 Java 认证和授权服务

“Java(TM) 认证和授权服务”（JAAS）是对 Java 2 Software Development Kit（J2SDK）Standard Edition 的标准扩展。当前，J2SDK 提供了基于代码起源位置与代码签名人的访问控制（基于代码源的访问控制）。然而，它不包含强制实施基于代码运行者的附加访问控制的能力。JAAS 提供了一个框架，用于将此项支持添加至 Java 2 安全性模型。

IBM 和 Sun Microsystems 使用 JAAS API 来作为对 J2SDK » V1.2 和 1.3 « 的扩展。IBM 和 Sun 引入此项扩展的目的是允许将特定用户或身份与当前 Java 线程相关联。这是通过使用 javax.security.auth.Subject 方法并对下层操作系统线程选择使用 com.ibm.security.auth.ThreadSubject 方法完成的。

» **注意**: 对于 J2SDK V1.4，JAAS 不再是扩展，而是基本 SDK 的一部分。 «

iSeries 服务器上的 JAAS 实现与 Sun Microsystems 的实现相兼容。本文档阐述了 iSeries 实现的独特方面。我们假定您熟悉与 JAAS 扩展相关的一般文档。为了便于您使用该信息以及我们的 iSeries 信息，我们提供了下列链接。

- API 开发者指南提供有关在软件开发中使用 JAAS API 的信息。
- 登录 / 认证模块开发者指南关注 JAAS 的认证方面。
- JAAS API 规范包含有关 JAAS 的 Javadoc 信息。

请选择下列任何主题来了解更多关于如何使用 JAAS 的详细信息:

- 为 JAAS 准备和配置 iSeries 服务器
- JAAS 样本
- 特定于 iSeries 服务器的 JAAS Javadoc

为 Java 认证和授权服务准备和配置 iSeries 服务器

必须符合软件需求并配置 iSeries 服务器才能使用 “Java(TM) 认证和授权服务”（JAAS）。

在 iSeries 服务器上运行 JAAS 1.0 的软件需求

安装下列许可程序:

- Java 2 SDK V1.4（J2SDK）
- 需要安装 IBM Toolbox for Java（修订版 4）许可程序（5722-JC1）才能更改 OS 线程身份。此许可程序包含支持更改 iSeries OS 线程身份所需的 ProfileTokenCredential 类和本机实现类。

配置系统

要将系统配置为使用 JAAS，请执行下列步骤:

1. 对于 JDK 1.2 和 1.3，添加指向 jaas13.jar 文件的扩展目录的符号链接。扩展类装入程序应装入 JAR 文件。在 iSeries 命令行上运行以下命令（将一行上输入整个命令）来添加该链接:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/jaas13.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jaas13.jar')
```

   » **注意**: 对于 JDK 1.4，不需要添加指向扩展目录的符号链接。对于此版本，JAAS 是基本 SDK 的一部分。 «

2. 在 ${java.home}/lib/security 中提供了缺省的 login.config 文件，此文件调用 com.ibm.as400.security.auth.login.BasicAuthenticationLoginModule。这个 login.config 文件将单一使用 ProfileTokenCredential 连接至认证的主题。如果要使用您自己的带有不同选项的 login.config 文件，则可在调用应用程序时包括以下系统属性:

   ```
   -Djava.security.auth.login.config=your login.config file
   ```

3. 添加指向 jt400Native.jar 文件的扩展目录的符号链接。这将允许扩展类装入程序装入这个文件。对于作为 IBM Toolbox for Java 一部分的凭证实现类，jaas13.jar 文件需要这个 JAR 文件。应用程序类装入程序还可通过将此文件包括在 CLASSPATH 中来装入它。如果从类路径目录装入此文件，则不要添加指向扩展目录的符号链接。

   » 以符号方式将 jt400Native.jar 文件链接至 /QIBM/ProdData/Java400/jdk14/lib/ext 目录将强制服务器上的所有 JDK 1.4 用户使用此版本的 jt400Native.jar 运行。« 如果各种用户需要不同版本的 IBM Toolbox for Java 类，则这可能不是所期望的。其它选项包括象前面描述的那样将 jt400Native.jar 放在应用程序的 CLASSPATH 中。另一个选项是添加指向您自己的目录的符号链接并接着通过在调用应用程序时指定 java.ext.dirs 系统属性来将该目录包括在扩展目录类路径中。

   要将 jt400Native.jar 文件链接至 /QIBM/ProdData/Java400/jdk13/lib/ext 目录，请在 iSeries 命令行上运行以下命令来添加链接:

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
      NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/jt400Native.jar')
   ```

   » 要将 jt400Native.jar 文件链接至 /QIBM/ProdData/Java400/jdk14/lib/ext 目录，请在 iSeries 命令行上运行以下命令来添加链接:

   ```
   ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
      NEWLNK('/QIBM/ProdData/Java400/jdk14/lib/ext/jt400Native.jar')
   ```

   «

   要将 jt400Native.jar 文件链接至您自己的目录，请执行下列操作:

   a. 在 iSeries 命令行上运行以下命令以添加链接:

      ```
      ADDLNK OBJ('/QIBM/ProdData/OS400/jt400/lib/jt400Native.jar')
         NEWLNK('your extension directory/jt400Native.jar')
      ```

   b. 在调用 Java 程序时，使用以下模式:

      ```
      java -Djava.ext.dirs=your extension directory:default
      extension directories
      ```

      注意: 有关 iSeries 凭证类的信息，参见 IBM Toolbox for Java。单击**安全性类**。单击**认证服务**。单击 **ProfileTokenCredential** 类。单击**包**。

4. 更新 Java 2 策略文件以将适当的许可权授予 IBM Toolbox for Java JAR 文件的实际位置。即使这些文件可能以符号方式链接至扩展目录并且在 ${java.home}/lib/security/java.policy 文件中对那些目录授予了 java.security.AllPermission，授权也基于 JAR 文件的实际位置。

   为了成功地使用 IBM Toolbox for Java 中的凭证类，请将以下内容添加至应用程序的 Java 2 策略文件:

   ```
   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";
    }
   ```

由于 IBM Toolbox for Java JAR 文件执行的操作不以特权方式运行，所以还需要为应用程序的 codeBase 添加这些许可权。

有关 Java 2 策略文件的信息，参见 API 开发者指南。

5. 确保 “iSeries 主机服务器” 已启动并且正在运行。将驻留在 Toolbox 中的 ProfileTokenCredential 类（例如 jt400Native.jar）用作与认证的主题相连的凭证。凭证类需要访问 “主机服务器”。可通过在 iSeries 命令提示上输入以下命令来验证服务器是否已启动并且正在运行:
   - StrHostSVR *all
   - StrTcpSvr *DDM

   如果服务器已启动，则这些步骤不执行任何操作。如果服务器尚未启动，则将启动服务器。

Java 认证和授权服务样本

在本信息中，我们提供了指向 iSeries 服务器上的一些 “Java(TM) 认证和授权服务” （JAAS）样本的链接。文档附带包括两个 JAAS 样本，即 HelloWorld 和 SampleThreadSubjectLogin。请单击下面这些链接以获取指示信息和源代码:

- HelloWorld
- SampleThreadSubjectLogin

第 4 章 IBM Java 一般安全性服务（JGSS）

“Java 一般安全性服务”（JGSS）提供认证和安全消息传递的一般接口。在此接口下，可以加入各种基于保密密钥、公用密钥或其它安全性技术的安全性机制。

通过将下层安全性机制的复杂性和特质抽象为标准化接口，JGSS 提供以下益处以便于开发安全联网应用程序:

- 可以对单一抽象接口开发应用程序
- 可以使用具有不同安全性机制的应用程序而不用作任何更改

JGSS 为“一般安全性服务应用程序编程接口”（ GSS-API）定义了 Java 绑定，该接口是一个加密 API，已由“因特网工程任务组”（IETF）标准化并被 X/Open Group 采用。

JGSS 的 IBM 实现称为 IBM JGSS。IBM JGSS 是 GSS-API 框架的一种实现，它使用 Kerberos V5 作为缺省下层安全性系统。它还提供了一个 Java(TM) 认证和授权服务（JAAS）登录模块用于创建和使用 Kerberos 凭证。另外，当使用这些凭证时，您也可以让 JGSS 执行 JAAS 授权检查。

IBM JGSS 包括一个本机 iSeries JGSS 提供程序、一个 Java JGSS 提供程序和 Kerberos 凭证管理工具（kinit、ktab 和 klist）的 Java 版本。

注意: 本机 iSeries JGSS 提供程序使用本机 iSeries 网络认证服务（NAS）库。当使用本机提供程序时，必须使用本机 iSeries Kerberos 实用程序。有关更多信息，参见 JGSS 提供程序。

有关使用 JGSS 的更多信息，参见下列主题:

JGSS 概念
介绍 JGSS 概念，包括 GSS-API 操作的高级描述和安全性机制的简短论述。

配置服务器以使用 JGSS
查明如何配置 iSeries 服务器以将 IBM JGSS 与 Java(TM) 2 Software Development Kit Standard Edition（J2SDK）配合使用。信息包括标识和设置使用带有安全性管理器的 JGSS 所必需的许可权。

运行 JGSS 应用程序
了解如何在 iSeries 服务器上运行 JGSS 应用程序。文档包括操作概念的解释和使用 JAAS 的指示信息。

开发 JGSS 应用程序
了解如何使用 JGSS 来开发安全应用程序。了解如何生成传送记号、创建 JGSS 对象、建立上下文等。

JGSS javadoc 参考信息
复查 org.ietf.jgss api 包中的类和方法的 javadoc 信息以及 Kerberos 凭证管理工具（kinit、ktab 和 klis）的 Java 版本的 javadoc 信息。

JGSS 样本
使用样本程序来发现如何在应用程序中使用 JGSS。样本文档包括 Java 源代码、运行样本的指示信息、配置和策略文件等。

要了解关于 Java 安全性和一般安全性服务的更多信息，参见以下文档:

-
- Sun Microsystems 的 J2SDK Security enhancement  ，它包含至更多 Java GSS-API 信息的链接。

- “因特网工程任务组”（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

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

JGSS 概念

JGSS 操作由四个不同的阶段组成，这些阶段已通过“一般安全性服务应用程序编程接口”（GSS-API）实行了标准化:

1. 收集主体的凭证
2. 在通信对等主体之间创建和建立安全上下文
3. 在对等系统之间交换安全消息
4. 清除和释放资源

另外，JGSS 使 Java 加密体系结构能够提供不同安全性机制的无缝可插性。

使用以下链接来阅读这些重要的 JGSS 概念的高层次描述。

- 主体和凭证
- 上下文建立
- 消息保护和交换
- 资源清除和释放
- 安全性机制

主体和凭证

应用程序参加与对等系统的 JGSS 安全通信所使用的身份称为主体。主体可以是实际的用户或无人照管服务。主体获取特定于安全性机制的凭证作为该机制下的身份证明。例如，当使用 Kerberos 机制时，主体的凭证的格式为 Kerberos 密钥分发中心（KDC）所发出的票据授予票据（TGT）的格式。在多机制环境中，GSS-API 凭证可以包含多凭证元素，每个元素代表一个下层机制凭证。

GSS-API 标准没有规定主体如何获取凭证，并且 GSS-API 实现一般不提供获取凭证的方法。主体在使用 GSS-API 之前获取凭证；GSS-API 仅代表主体查询凭证的安全性机制。

IBM JGSS 包括 Kerberos 凭证管理工具 kinit、ktab 和 klist 的 Java 版本。另外，IBM JGSS 通过提供使用 JAAS 的可选 Kerberos 登录界面来增强标准 GSS-API。纯 Java JGSS 提供程序支持可选的登录界面；而本机 iSeries 提供程序不支持。有关更多信息，参见下列主题:

- 获取 Kerberos 凭证
- JGSS 提供程序

上下文建立

在获取安全性凭证后，两个通信的对等系统使用它们的凭证来建立安全性上下文。尽管这些对等系统建立了单一连接上下文，但每个对等系统都将维护上下文的自己的本地副本。上下文建立涉及对接受对等系统启动对自己的对等认证。可选地，启动方可以请求相互认证，在这种认证中，接受方向启动方认证自己。

当完成上下文建立时，建立的上下文会体现状态信息（如共享加密密钥），以允许在两个对等系统之间进行后续安全消息交换。 « »

消息保护和交换

在建立上下文后，两个对等系统就可以进行安全消息交换了。消息的始发者在其本地 GSS-API 实现上进行调用来编码消息，这可以确保消息的完整性和消息机密性（可选）。然后应用程序将结果记号传送给对等系统。

对等系统的本地 GSS-API 实现以下列方式使用已建立的上下文中的信息:

- 验证消息的完整性
- 解密消息（如果消息是加密的） «

»

资源清除和释放

为了释放资源，JGSS 应用程序删除不再使用的上下文。尽管 JGSS 应用程序可以访问已删除的上下文，但使用它来进行消息交换的任何尝试都会导致异常。 « »

安全性机制

GSS-API 由基于一个或多个下层安全性机制的抽象框架组成。该框架与下层安全性机制交互的方式是特定于实现的。这样的实现以两个一般类别的形式存在:

- 在一个极端，整体式实现将框架紧密地绑定至单个机制上。这种实现可以防止使用其它机制，甚至同一机制的不同实现。
- 在此领域的另一端，高度模块化的实现具有容易使用和灵活的特点。这种实现提供了无缝且容易地在框架中加入不同的安全性机制和它们的实现的能力。

IBM JGSS 属于后面这个类别。作为一种模块化实现，IBM JGSS 扩展了由 Java 加密体系结构（JCA）定义的提供程序框架并将任何下层机制当作（JCA）提供程序。JGSS 提供程序提供 JGSS 安全性机制的具体实现。应用程序可以使多个机制实例化并使用它们。

提供程序支持多个机制是可能的，并且 JGSS 使得容易使用不同安全性机制。然而，GSS-API 未提供当多个机制可用时两个通信的对等系统如何选择机制的方法。选择机制的一种方法是从“简单和受保护的 GSS-API 协商机制”（SPNEGO）开始，这是一种在两个对等系统之间协商实际机制的伪机制。IBM JGSS does not include a SPNEGO mechanism.

有关 SPNEGO 的更多信息，参见“因特网工程任务组”（IETF）RFC 2478 The Simple and Protected GSS-API Negotiation Mechanism 。 « »

配置 iSeries 服务器以使用 IBM JGSS

如何配置 iSeries 服务器以使用 JGSS 取决于在服务器上运行的 Java 2 Software Development Kit（J2SDK）的版本。有关配置 iSeries 服务器以使用 JGSS 的更多信息，使用以下链接:

- 使用具有 J2SDK V1.3 的 JGSS
- 使用具有 J2SDK V1.4 的 JGSS
- 配置 JGSS 以使用本机 iSeries JGSS 提供程序

配置 iSeries 服务器以使用具有 J2SDK 的 JGSS V1.3。

当在 iSeries 服务器上使用 Java 2 Software Development Kit（J2SDK）V1.3 时，您需要准备并配置服务器以使用 JGSS。缺省配置使用纯 Java JGSS 提供程序。

软件需求

要使用具有 J2SDK 的 JGSS V1.3，服务器必须已安装 Java 认证和授权服务（JAAS）1.3。

配置服务器以使用 JGSS

要配置服务器以使用具有 J2SDK 的 JGSS V1.3，将一个符号链接添加至 ibmjgssprovider.jar 文件的扩展目录中。ibmjgssprovider.jar 文件包含 JGSS 类和纯 Java JGSS 提供程序。添加符号链接使扩展类装入器能够装入 ibmjgssprovider.jar 文件。

添加符号链接

要添加符号链接，在 iSeries 命令行上，输入以下命令（在单独一行上）并按**执行**键:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssprovider.jar')
NEWLNK('/QIBM/ProdData/Java400/jdk13/lib/ext/ibmjgssprovider.jar')
```

注意: iSeries 服务器上的缺省 Java 1.3 策略将适当的许可权授予 JGSS。如果您打算创建自己的 java.policy 文件，参见 JVM 许可权以获取要授予 ibmjgssprovider.jar 的许可权列表。

更改 JGSS 提供程序

在配置服务器以使用 JGSS 后（JGSS 使用纯 Java 提供程序作为缺省提供程序），可以配置 JGSS 以使用本机 iSeries JGSS 提供程序。那么，在配置 JGSS 以使用本机提供程序后，您可以容易地在两个提供程序之间进行切换。有关更多信息，参见下列主题:

- JGSS 提供程序
- 配置 JGSS 以使用本机 iSeries JGSS 提供程序

安全性管理器

如果要在启用 Java 安全性管理器的情况下运行 IBM JGSS 应用程序，参见使用安全性管理器。

配置 JGSS 以使用本机 iSeries JGSS 提供程序

IBM JGSS 在缺省情况下使用纯 Java 提供程序。您可以选择使用本机 iSeries JGSS 提供程序。有关不同提供程序的更多信息，参见 JGSS 提供程序。

软件需求

本机 iSeries JGSS 提供程序必须能够访问 IBM Toolbox for Java 中的类。有关如何访问 Toolbox for Java 的指示信息，参见允许本机 iSeries JGSS 提供程序访问 IBM Toolbox for Java。

确保您已配置网络认证服务。有关更多信息，参见网络认证服务。

指定本机 iSeries JGSS 提供程序

在使用具有 J2SDK 的本机 iSeries JGSS 提供程序 V1.3 之前，确保已配置服务器来使用 JGSS。有关更多信息，参见配置 iSeries 服务器以使用具有 J2SDK 的 JGSS V1.3。如果正在使用 J2SDK V1.4，则已配置了 JGSS。

注意: 在以下指示信息中，${java.home} 表示您正在服务器上使用的 Java 版本的位置的路径。例如，如果正在使用 J2SDK V1.4，则 ${java.home} 为 /QIBM/ProdData/Java400/jdk14。记住用 Java 主目录的实际路径替换命令中的 ${java.home}。

要配置 JGSS 以使用本机 iSeries JGSS 提供程序，完成以下任务:

- 添加指向本机 iSeries 提供程序 JAR 文件的扩展目录的符号链接
- 将本机 iSeries JGSS 提供程序添加到 java.security 文件的安全性提供程序列表中

添加符号链接

要添加指向 ibmjgssiseriesprovider.jar 文件的扩展目录的符号链接，在 iSeries 命令行上，输入以下命令（在单独一行上）并按**执行键**:

```
ADDLNK OBJ('/QIBM/ProdData/OS400/Java400/ext/ibmjgssiseriesprovider.jar')
NEWLNK('${java.home}/lib/ext/ibmjgssiseriesprovider.jar')
```

在添加指向 ibmjgssiseriesprovider.jar 文件的扩展目录的符号链接后，扩展类装入器将装入该 JAR 文件。

将提供程序添加到安全性提供程序列表中

将本机提供程序添加到 java.security 文件的安全性提供程序列表中

1. 打开 ${java.home}/lib/security/java.security 以进行编辑。
2. 找到安全性提供程序列表。它应 java.security 文件的顶部附近并类似于以下内容:

   ```
   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. 将本机 iSeries JGSS 提供程序添加到安全性提供程序列表中的原始 Java 提供程序的前面。换句话说，将 com.ibm.iseries.security.jgss.IBMJGSSiSeriesProvider 添加到列表中其编号小于 com.ibm.jgss.IBMJGSSProvider 的编号的行，然后更新 IBMJGSSProvider 的位置。例如:

   ```
   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
   ```

   注意，IBMJGSSiSeriesProvider 变成列表中的第四项，而 IBMJGSSProvider 变成列表中的第五项。并且，检查安全性提供程序列表中的项号是否连续以及每个项的项号增量是否为 1。

4. 保存并关闭 java.security 文件。 «

»

配置 iSeries 服务器以使用具有 J2SDK V1.4 的 JGSS

当在 iSeries 服务器上使用 Java 2 Software Development Kit（J2SDK）V1.4 时，则已配置 JGSS。缺省配置使用纯 Java JGSS 提供程序。

更改 JGSS 提供程序

可以配置 JGSS 以使用本机 iSeries JGSS 提供程序代替纯 Java JGSS 提供程序。那么，在配置 JGSS 以使用本机提供程序后，您可以容易地在两个提供程序之间进行切换。有关更多信息，参见下列主题:

- JGSS 提供程序
- 配置 JGSS 以使用本机 iSeries JGSS 提供程序

安全性管理器

如果要在启用 Java 安全性管理器的情况下运行 JGSS 应用程序，参见使用安全性管理器。 « »

JGSS 提供程序

IBM JGSS 包括一个本机 iSeries JGSS 提供程序和一个纯 Java JGSS 提供程序。您选择使用的提供程序取决于应用程序的需要。

纯 Java JGSS 提供程序提供以下功能:

- 确保应用程序的可移植性的级别最高
- 使用可选的 JAAS Kerberos 登录界面
- 与 Java Kerberos 凭证管理工具兼容

本机 iSeries JGSS 提供程序提供以下功能:

- 使用本机 iSeries Kerberos 库
- 与 Qshell Kerberos 凭证管理工具兼容
- JGSS 应用程序运行更快

注意: 两个 JGSS 提供程序都遵守 GSS-API 规范，因此它们彼此兼容。换句话说，使用纯 Java JGSS 提供程序的应用程序和使用本机 iSeries JGSS 提供程序的应用程序可以互操作。

更改 JGSS 提供程序

注意: 如果服务器正在运行 J2SDK V1.3，在更改为本机 iSeries JGSS 提供程序前，确保您已配置服务器来使用 JGSS。有关更多信息，参见下列主题:

- 配置 iSeries 服务器以使用具有 J2SDK V1.3 的 JGSS
- 配置 JGSS 以使用本机 JGSS 提供程序

通过使用以下选项之一，可以方便地更改 JGSS 提供程序:

- 在 `${java.home}/lib/security/java.security` 中编辑安全性提供程序列表

  注意: ${java.home} 表示您正在服务器上使用的 Java 版本的位置的路径。例如，如果正在使用 J2SDK V1.3，则 ${java.home} 为 /QIBM/ProdData/Java400/jdk13。
- 使用 GSSManager.addProviderAtFront() 或 GSSManager.addProviderAtEnd() 来在 JGSS 应用程序中指定提供程序的名称。有关更多信息，参见 GSSManager javadoc。 «

»

使用安全性管理器

如果要在启用 Java 安全性管理器的情况下运行 JGSS 应用程序，需要确保应用程序和 JGSS 具有必要的许可权。有关使用 JGSS 所需要的许可权的更多信息，参见以下主题:

- JVM 许可权

- JAAS 许可权检查

JVM 许可权

除了由 JGSS 执行的访问控制检查以外，Java 虚拟机（JVM）还在访问各种资源（包括文件、Java 属性、包和套接字）时执行授权检查。

有关使用 JVM 许可权的更多信息，参见 Permissions in the Java 2 SDK  。

以下列表标识当使用 JGSS 的 JAAS 功能部件或使用带有安全性管理器的 JGSS 时所需要的许可权:

- 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 许可权检查

当启用 JAAS 的程序使用凭证和访问服务时，IBM JGSS 执行运行时许可权检查。通过将 Java 属性 javax.security.auth.useSubjectCredsOnly 设置为 false，可以禁用此可选 JAAS 功能部件。而且，仅当应用程序与安全性管理器一起运行时，JGSS 才执行许可权检查。

JGSS 对 Java 策略执行许可权检查，该 Java 策略对当前访问控制上下文有效。JGSS 执行以下特定许可权检查:

- javax.security.auth.kerberos.DelegationPermission
- javax.security.auth.kerberos.ServicePermission

DelegationPermission check

DelegationPermission 允许安全性策略控制凭单转发和 Kerberos 的代理功能部件的使用。通过使用这些功能部件，客户机可以允许服务代表客户机进行操作。

DelegationPermission 采用两个自变量，顺序如下:

1. 辅助主体，它是代表客户机并在客户机的权限之下进行操作的服务主体的名称
2. 客户机要允许辅助主体使用的服务的名称

示例: 使用 DelegationPermission 检查

在以下示例中，superSecureServer 是辅助主体，krbtgt/REALM.IBM.COM@REALM.IBM.COM 是要允许 superSecureServer 代表客户机使用的服务。在这种情况中，服务是客户机的票据授予票据，这意味着 superSecureServer 可以代表客户机获取任何服务的凭单。

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"krbtgt/REALM.IBM.COM@REALM.IBM.COM\"";
```

在先前示例中，DelegationPermission 对客户机授予许可权，以从“密钥分发中心”（KDC）获取只有 superSecureServer 可以使用的新的票据授予票据。在客户机将新的票据授予票据发送至 superSecureServer 之后，superSecureServer 就具有代表客户机执行操作的能力。

以下示例允许客户机获取一个新的凭单，该凭单允许 superSecureServer 代表客户机仅访问 ftp 服务:

```
permission javax.security.auth.kerberos.DelegationPermission
     "\"superSecureServer/host.ibm.com@REALM.IBM.COM\"
     \"ftp/ftp.ibm.com@REALM.IBM.COM\"";
```

有关更多信息，参见 Sun Web 站点上的 J2SDK文档  中的 javax.security.auth.kerberos.DelegationPermission 类。

ServicePermission 检查

ServicePermission 检查会限制将凭证用于上下文启动和接收。上下文启动方必须具有启动上下文的许可权。同样，上下文接受方必须具有接受上下文的许可权。

示例: 使用 ServicePermission 检查

以下示例通过对客户机授予许可权来允许客户机端用 ftp 服务启动上下文:

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "initiate";
```

以下示例通过对服务器授予许可权来允许服务器端访问并使用 ftp 服务的保密密钥:

```
permission javax.security.auth.kerberos.ServicePermission
     "ftp/host.ibm.com@REALM.IBM.COM", "accept";
```

有关更多信息，参见 Sun Web 站点上的 J2SDK文档  中的 javax.security.auth.kerberos.ServicePermission 类。

运行 IBM JGSS 应用程序

“IBM Java 一般安全性服务”（JGSS）API 1.0 使安全应用程序与不同下层安全性机制的复杂性和特质隔离开来。JGSS 使用“Java 认证和授权服务”（JAAS）和“IBM Java 密码术扩展”（JCE）提供的功能部件。

JGSS 功能部件包括:

- 身份认证
- 消息完整性和机密性

- 可选的 JAAS Kerberos 登录界面和授权检查

有关运行 JGSS 应用程序的更多信息，参见下列主题:

获取 Kerberos 凭证
查明如何获取 Kerberos 凭证和创建保密密钥。了解如何使用 JAAS 来执行 Kerberos 登录和授权检查并复查 “Java 虚拟机”（JVM）所需要的 JAAS 许可权列表。

配置和策略文件
了解运行 JGSS 所需要的各种支持文件，包括配置文件、策略文件、Java 主安全性属性文件和凭证高速缓存。

调试
了解如何使用 JGSS 调试来分类并显示有帮助的调试消息。

JGSS 样本
使用样本程序来测试和验证 JGSS 设置。样本文档包括 Java 源代码、关于运行样本的指示信息、配置和策略文件等。 «

»

获取 Kerberos 凭证并创建保密密钥

GSS-API 未定义获取凭证的方法。因此，IBM JGSS Kerberos 机制要求用户通过以下其中一个方法来获取 Kerberos 凭证:

- Kinit 和 Ktab 工具
- 可选的 JAAS Kerberos 登录界面«

»

Kinit 和 Ktab 工具

您选择的 JGSS 提供程序确定您使用哪些工具来获取 Kerberos 凭证和保密密钥。

使用纯 Java JGSS 提供程序

如果正在使用纯 Java JGSS 提供程序，则使用 IBM JGSS Kinit 和 Ktab 工具来获取凭证和保密密钥。Kinit 和 Ktab 工具使用命令行界面并提供类似于其它版本所提供的选项。

- 通过使用 Kinit 工具，可以获取 Kerberos 凭证。此工具联系 “Kerberos 分发中心”（KDC）并获取票据授予票据（TGT）。TGT 允许您访问其它启用 Kerberos 的服务，包括使用 GSS-API 的服务。
- 通过使用 Ktab 工具，服务器可以获取保密密钥。JGSS 将保密密钥存储在服务器上的密钥表文件中。有关更多信息，参见 Ktab Java 文档。

或者，应用程序可以使用 JAAS 登录界面来获取 TGT 和保密密钥。有关更多信息，参见以下内容:

- Kinit javadoc
- Ktab javadoc
- JAAS 登录界面。

使用本机 iSeries JGSS 提供程序

如果正在使用本机 iSeries JGSS 提供程序，则使用 Qshell kinit 和 klist 实用程序。有关更多信息，参见 Kerberos 凭证和密钥表的实用程序。 « »

JAAS Kerberos 登录界面

IBM JGSS 提供了一个“Java 认证和授权服务”（JAAS）Kerberos 登录界面。通过将 Java 属性 javax.security.auth.useSubjectCredsOnly 设置为 false，可以禁用此功能部件。

注意: 尽管纯 Java JGSS 提供程序可以使用登录界面，但本机 iSeries JGSS 提供程序却不能使用。

有关 JAAS 的更多信息，参见 Java 认证和授权服务。

JAAS 和 JVM 许可权

如果正在使用安全性管理器，需要确保应用程序和 JGSS 具有必要的 JVM 和 JAAS 许可权。有关更多信息，参见使用安全性管理器。

JAAS 配置文件选项

登录界面需要一个 JAAS 配置文件，该文件指定 com.ibm.security.auth.module.Krb5LoginModule 作为要使用的登录模块。下表列示了 Krb5LoginModule 支持的选项。注意，选项是不区分大小写的。

选项名	值	缺省值	解释
principal	<string>	无；提示输入。	Kerberos 主体名称
credsType	initiator|acceptor|both	initiator	JGSS 凭证类型
forwardable	true|false	false	是否要获取可转发的票据授予票据（TGT）
proxiable	true|false	false	是否要获取可代理的 TGT
useCcache	<URL>	不使用 ccache	从指定的凭证高速缓存中检索 TGT
useKeytab	<URL>	不使用密钥表	从指定的密钥表中检索保密密钥
useDefaultCcache	true|false	不使用缺省 ccache	从缺省凭证高速缓存中检索 TGT
useDefaultKeytab	true|false	不使用缺省密钥表	从指定的密钥表中检索保密密钥

有关使用 Krb5LoginModule 的简单示例，参见样本 JAAS 登录配置文件。

选项不兼容性

某些 Krb5LoginModule 选项（主体名除外）互相不兼容，这意味着您不能将它们一起指定。下表表示兼容的和不兼容的登录模块选项。

表中的指示符描述两个相关联的选项之间的关系:

- X = 不兼容
- N/A = 不适用的组合
- 空白 = 兼容

Krb5LoginModule 选项	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
proxiable		X				X	X	X	X
useCcache		X		X	X		X	X	X

Krb5LoginModule 选项	credsType initiator	credsType acceptor	credsType both	forward	proxy	u s e Ccache	u s e Keytab	useDefault Ccache	useDefault Keytab
useKeytab	X			X	X	X		X	X
useDefaultCcache		X		X	X	X	X		X
useDefaultKeytab	X			X	X	X	X	X	

主体名选项

可以将主体名与任何其它选项一起指定。当您未指定主体名时，Krb5LoginModule 可能会提示用户输入主体名。Krb5LoginModule 是否提示用户取决于您指定的其它选项。有关更多信息，参见提示输入主体名和密码。

服务主体名格式

必须使用以下其中一个格式来指定服务主体名:

- <service_name>（例如，superSecureServer）
- <service_name>@<host>（例如，superSecureServer@myhost）

在后面这个格式中，<host> 是服务所驻留的机器的主机名。可以（但并不是一定要）使用全限定主机名。

注意: JAAS 将某些字符识别为定界符。当在 JAAS 字符串（如主体名）中使用以下任何字符时，用引号括起该字符:

```
_（下划线）
:（冒号）
/ （斜杠）
\（反斜杠）
```

提示输入主体名和密码

您在 JAAS 配置文件中指定的选项确定 Krb5LoginModule 登录是非交互式还是交互式。

- 非交互式登录不提示输入任何信息
- 交互式登录提示输入主体名和 / 或密码

非交互式登录

当将凭证类型指定为启动方（`credsType=initiator`）且执行以下其中一项操作时，登录以非交互式的方式进行:

- 指定 useCcache 选项
- 将 useDefaultCcache 选项设置为 true

当将凭证类型指定为接受方或两者（`credsType=acceptor` 或 `credsType=both`）且执行以下其中一项操作时，登录也以非交互式的方式进行:

- 指定 useKeytab 选项
- 将 useDefaultKeytab 选项指定为 true

交互式登录

其它配置导致登录模块提示输入主体名和密码，以便它可以从 Kerberos KDC 获取 TGT。当指定主体选项时，登录模块仅提示输入密码。

交互式登录要求应用程序指定 com.ibm.security.auth.callback.Krb5CallbackHandler 作为创建登录上下文时的回调处理程序。回调处理程序负责提示进行输入。

凭证类型选项

当要求凭证类型为启动方和接受方（`credsType=both`）时，Krb5LoginModule 获取 TGT 和保密密钥。登录模块使用 TGT 来启动上下文和保密密钥以接受上下文。JAAS 配置文件必须包含足够的信息才能使登录模块能够获得两种类型的凭证。

对于凭证类型接受方和两者，登录模块假定一个服务主体。 

配置和策略文件

JGSS 和 JAAS 依赖于几个配置和策略文件。需要编辑这些文件以符合您的环境和应用程序。如果不将 JAAS 与 JGSS 配合使用，则可以安全地忽略 JAAS 配置和策略文件。

- Kerberos 配置文件
- JAAS 配置文件
- JAAS 授权策略文件
- Java 主安全性属性文件
- 凭证高速缓存和服务器密钥表

注意: 在以下指示信息中，${java.home} 表示您正在服务器上使用的 Java 版本的位置的路径。例如，如果正在使用 J2SDK V1.4，则 ${java.home} 为 /QIBM/ProdData/Java400/jdk14。记住用 Java 主目录的实际路径替换属性设置中的 ${java.home}。

Kerberos 配置文件

IBM JGSS 需要一个 Kerberos 配置文件。Kerberos 配置文件的缺省名称和位置取决于正在使用的操作系统。JGSS 使用以下顺序来搜索缺省配置文件:

1. 由 Java 属性 java.security.krb5.conf 引用的文件
2. ${java.home}/lib/security/krb5.conf
3. Microsoft Windows(R) 平台上的 c:\winnt\krb5.ini
4. Solaris(TM) 平台上的 /etc/krb5/krb5.conf
5. 其它 Unix(R) 平台上的 /etc/krb5.conf

JAAS 配置文件

使用 JAAS 登录功能部件需要 JAAS 配置文件。通过设置以下其中一个属性，可以指定 JAAS 配置文件:

- Java 系统属性 java.security.auth.login.config
- ${java.home}/lib/security/java.security 文件中的安全性属性 login.config.url.<integer>

有关更多信息，参见 Sun Java 认证和授权服务（JAAS）Web 站点  。

JAAS 策略文件

当使用缺省策略实现时，JGSS 通过在策略文件中记录许可权将 JAAS 许可权授予实体。通过设置以下其中一个属性，可以指定 JAAS 策略文件:

- Java 系统属性 java.security.policy
- ${java.home}/lib/security/java.security 文件中的安全性属性 policy.url.<integer>

如果正在使用 J2SDK V1.4，则为 JAAS 指定单独的策略文件是可选的。J2SDK V1.4 中的缺省策略提供程序支持 JAAS 所需要的策略文件项。

有关更多信息，参见 Sun Java 认证和授权服务（JAAS）Web 站点  。

Java 主安全性属性文件

Java 虚拟机（JVM）使用许多重要的安全性属性，您可以通过编辑 Java 主安全性属性文件设置它们。此文件（名为 java.security）通常驻留在 iSeries 服务器上的 ${java.home}/lib/security 目录中。

下面的列表描述几个与使用 JGSS 有关的安全性属性。使用描述作为编辑 java.security 文件的指南。

注意：如果适用，描述包括运行 JGSS 样本所需要的适当值。

security.provider.<integer>：您要使用的 JGSS 提供程序。还静态地注册加密提供程序类。IBM JGSS 使用 IBM JCE 提供程序提供的加密和其它安全性服务。严格按照以下示例指定 sun.security.provider.Sun 和 com.ibm.crypto.provider.IBMJCE 包:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

policy.provider：系统策略处理程序类。例如:

```
policy.provider=sun.security.provider.PolicyFile
```

policy.url.<integer>：策略文件的 URL。要使用样本策略文件，包括一个类似以下内容的项:

```
policy.url.1=file:/home/user/jgss/config/java.policy
```

login.configuration.provider：JAAS 登录配置处理程序类，例如:

```
login.configuration.provider=com.ibm.security.auth.login.ConfigFile
```

auth.policy.provider：JAAS 基于主体的访问控制策略处理程序类，例如:

```
auth.policy.provider=com.ibm.security.auth.PolicyFile
```

login.config.url.<integer>：JAAS 登录配置文件的 URL。要使用样本配置文件，包括一个类似于以下内容的项:

```
login.config.url.1=file:/home/user/jgss/config/jaas.conf
```

auth.policy.url.<integer>：JAAS 策略文件的 URL。可以将基于主体的和基于 CodeSource 的构造包括在 JAAS 策略文件中。要使用样本策略文件，包括一个类似以下内容的项:

```
auth.policy.url.1=file:/home/user/jgss/config/jaas.policy
```

凭证高速缓存和服务器密钥表

用户主体将它的 Kerberos 凭证保存在凭证高速缓存中。服务主体将它的保密密钥保存在密钥表中。在运行时，IBM JGSS 按下列方式找到这些高速缓存:

用户凭证高速缓存

JGSS 使用以下顺序来找到用户凭证高速缓存:

1. 由 Java(TM) 属性 KRB5CCNAME 引用的文件
2. 由环境变量 KRB5CCNAME 引用的文件
3. Unix 系统上的 /tmp/krb5cc_<uid>
4. ${user.home}/krb5cc_${user.name}
5. ${user.home}/krb5cc（如果不能获取 ${user.name}）

服务器密钥表

JGSS 使用以下顺序来找到服务器密钥表文件:

1. Java(TM) 属性 KRB5_KTNAME 的值
2. Kerberos 配置文件的 libdefaults 节中的 default_keytab_name 项
3. ${user.home}/krb5_keytab

开发 IBM JGSS 应用程序

有关开发 IBM JGSS 应用程序的更多信息，参见以下主题:

编程步骤
了解开发 JGSS 应用程序所需的步骤，它包括使用传送记号、创建必要的 JGSS 对象、建立和删除上下文以及使用 per-message 服务。

将 JAAS 与 JGSS 应用程序配合使用
了解如何启用 JGSS 的 JAAS Kerberos 登录功能部件。信息包括使用登录功能部件的需求和示例代码的程序片断。

调试
了解如何使用 JGSS 调试来分类并显示有帮助的调试消息。

JGSS javadoc 参考信息
复查 org.ietf.jgss api 包中的类和方法的 javadoc 信息以及 Kerberos 凭证管理工具（kinit、ktab 和 klis）的 Java 版本的 javadoc 信息。

JGSS 样本
使用样本程序来发现如何才能在应用程序中使用 JGSS。样本文档包括 Java 源代码、运行样本的指示信息、配置和策略文件等。

要开发 JGSS 应用程序，需要熟悉高级 GSS-API 规范和 Java 绑定规范。IBM JGSS 1.0 主要基于并遵守这些规范。有关更多信息，参见以下链接。

- RFC 2743: Generic Security Service Application Programming Interface Version 2, Update 1
- RFC 2853: Generic Security Service API Version 2: Java Bindings

IBM JGSS 应用程序编程步骤

JGSS 应用程序中的操作遵循“一般安全性服务应用程序编程接口”（GSS-API）操作模型。有关 JGSS 操作的重要概念的信息，参见 JGSS 概念。

JGSS 传送记号

某些重要的 JGSS 操作会生成 Java 字节数组的格式的记号。将记号从一个 JGSS 对等系统转发至另一个对等系统是应用程序的责任。JGSS 并不以任何方式约束应用程序用于传送记号的协议。应用程序可以将 JGSS 记号与其它应用程序（即非 JGSS）数据一起传送。然而，JGSS 操作仅接受和使用特定于 JGSS 的记号。

JGSS 应用程序中的操作顺序

JGSS 操作需要某些必须按以下列示的次序使用的编程构造。每个步骤适用于启动方和接受方。

注意: 信息包括示例代码的程序片断，它们举例说明了如何使用高级 JGSS API 并假设应用程序导入了 org.ietf.jgss 包。尽管许多高级 API 是重载的，但是程序片断仅显示这些方法的最常用的格式。当然，应使用最适合需要的 API 方法。

1. 创建 GSSManager
   GSSManager 的实例用作创建其它 JGSS 对象实例的生成器。
2. 创建 GSSName
   GSSName 表示 JGSS 主体的身份。当指定空 GSSName 时，某些 JGSS 操作可以定位和使用缺省主体。
3. 创建 GSSCredential
   GSSCredential 体现主体的特定于机制的凭证。
4. 创建 GSSContext
   GSSContext 用于上下文建立和后继 per-message 服务。
5. 在上下文中选择可选的服务
   应用程序必须显式地请求可选的服务，如相互认证。
6. 建立上下文
   启动方对接受方认证自己。然而，当请求相互认证时，接受方又对启动方认证自己。
7. 使用 per-message 服务
   启动方和接受方基于建立的上下文交换安全消息。
8. 删除上下文
   应用程序删除不再需要的上下文。«

»

创建 GSSManager

GSSManager 抽象类用作创建以下 JGSS 对象的生成器:

- GSSName
- GSSCredential
- GSSContext

GSSManager 也具有一些用于确定受支持的安全性机制和名称类型以及用于指定 JGSS 提供程序的方法。使用 GSSManager getInstance 静态方法来创建缺省 GSSManager 的实例:

```
GSSManager manager = GSSManager.getInstance();
```

»

创建 GSSName

GSSName 表示 GSS-API 主体的身份。GSSName 可以获得主体的许多表示，每个都代表一个受支持的下层机制。仅包含一个名称表示的 GSSName 称为 “机制名”（MN）。

GSSManager 具有几个用于从字符串或一个连续的字节数组创建 GSSName 的重载方法。这些方法根据指定的名称类型解释字符串或字节数组。一般情况下，应使用 GSSName 字节数组方法来重建导出的名称。导出的名称通常为类型 GSSName.NT_EXPORT_NAME 的机制名称。其中某些方法允许指定要用来创建名称的安全性机制。

示例: 使用 GSSName

以下基本代码片断显示如何使用 GSSName。

注意: 将 Kerberos 服务名字符串指定为 <service> 或 <service@host>，其中 <service> 为服务的名称，而 <host> 为服务运行所在机器的主机名称。可以（但并不是一定要）使用全限定主机名。当省略字符串的 @ <host> 部分时，GSSName 使用本地主机名。

```
// 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);
```

创建 GSSCredential

GSSCredential 包含为主体创建上下文所需要的所有加密信息并可以包含多个机制的凭证信息。

GSSManager 具有三种凭证创建方法。其中的两种方法采用参数来表示 GSSName、凭证的生存期、获取凭证所用的一个或多个机制和凭证使用类型。第三种方法仅采用使用类型并使用其它参数的缺省值。指定空机制也使用缺省机制。指定空机制数组导致方法返回缺省机制组的凭证。

注意: 因为 IBM JGSS 仅支持 Kerberos V5 机制，所以它为缺省机制。

应用程序一次只可以创建三种凭证类型中的其中一种（*启动*、*接受*或*启动和接受*）。

- 上下文启动方创建*启动*凭证
- 接受方创建*接受*凭证
- 也起启动方作用的接受方创建*启动和接受*凭证。

示例: 获得凭证

以下示例获得启动方的缺省凭证:

```
GSSCredentials fooCreds = manager.createCredentials(GSSCredential.INITIATE)
```

以下示例获得启动方 foo 的 Kerberos V5 凭证，这些凭证具有缺省有效期:

```
GSSCredential fooCreds = manager.createCredential(fooName, GSSCredential.DEFAULT_LIFETIME,
                                                  krb5Mech,GSSCredential.INITIATE);
```

以下示例获得一个全部为缺省值的接受方凭证:

```
GSSCredential serverCreds = manager.createCredential(null, GSSCredential.DEFAULT_LIFETIME,
                                                     (Oid)null, GSSCredential.ACCEPT);
```

创建 GSSContext

IBM JGSS 支持 GSSManager 为创建上下文提供的两种方法:

- 由上下文启动方使用的方法

- 由接受方使用的方法

注意: GSSManager 提供了用于创建上下文的第三种方法，它涉及重新创建先前导出的上下文。然而，因为 IBM JGSS Kerberos V5 机制不支持使用已导出的上下文，因此 IBM JGSS 不支持此方法。

应用程序不能使用启动方上下文来接受上下文，也不能使用接受方上下文来启动上下文。创建上下文的两种受支持的方法都需要一个凭证作为输入。当凭证的值为空时，JGSS 使用缺省凭证。

示例: 使用 GSSContext

以下示例创建上下文，主体（foo）用它可以启动主机（securityCentral）上具有对等系统（superSecureServer）的上下文。示例将该对等系统指定为 superSecureServer@securityCentral。创建的上下文在缺省时间段内有效:

```
GSSName serverName = manager.createName("superSecureServer@securityCentral",
                                        GSSName.NT_HOSTBASED_SERVICE, krb5Mech);
GSSContext fooContext = manager.createContext(serverName, krb5Mech, fooCreds,
                                              GSSCredential.DEFAULT_LIFETIME);
```

以下示例创建 superSecureServer 的上下文，以接受任意对等系统启动的上下文:

```
GSSContext serverAcceptorContext = manager.createContext(serverCreds);
```

注意，应用程序可以创建并同时使用两种类型的上下文。 

请求可选的安全性服务

应用程序可以请求几个可选的安全性服务中的任何一个。IBM JGSS 支持以下可选服务:

- 授权
- 相互认证
- 重放检测
- 无序检测
- 可用的 per-message 机密性
- 可用的 per-message 完整性

要请求可选的服务，应用程序必须通过在上下文中使用适当的请求方法显式地提出请求。只有启动方可以请求这些可选的服务。启动方必须在开始建立上下文之前提出请求。

有关可选服务的更多信息，参见 “因特网工程任务组（IETF）RFC 2743 Generic Security Services Application Programming Interface Version 2, Update 1 中的 ptional Service Support  。

示例: 请求可选的服务

在以下示例中，上下文（fooContext）请求启用相互认证和委托服务:

```
fooContext.requestMutualAuth(true);
fooContext.requestCredDeleg(true);
```

建立上下文

两个通信的对等系统必须建立可以使用 per-message 服务的安全上下文。启动方在其上下文中调用 initSecContext()，将记号返回至启动方应用程序。启动方应用程序将上下文记号传送至接受方应用程序。接受方在其上下文中调用 acceptSecContext()，指定从启动方接收到的上下文记号。根据启动方所选择的下层机制和

可选服务，acceptSecContext() 可能生成一个记号，接受方应用程序必须将该记号转发给启动方应用程序。然后启动方应用程序使用接收到的记号调用 initSecContext() 多次。

应用程序可以多次调用 GSSContext.initSecContext() 和 GSSContext.acceptSecContext()。应用程序也可以在上下文建立期间与对等系统交换多个记号。因此，建立上下文的典型方法使用循环来调用 GSSContext.initSecContext() 或 GSSContext.acceptSecContext()，直到应用程序建立上下文为止。

示例：建立上下文

以下示例说明上下文建立的启动方（foo）端:

```
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());
```

以下示例说明上下文建立的接受方端:

```
// 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());
```

使用 per-message 服务

在建立安全上下文后，两个正在通信的对等系统可以通过建立上下文来交换安全消息。任何一个对等系统都可以发出一条安全消息，而无论它在建立上下文时是用作启动方还是用作接受方。为了使消息安全，IBM JGSS 基于消息来计算加密消息完整性代码（MIC）。可选地，IBM JGSS 可以使用 Kerberos V5 机制来加密消息以帮助确保保密性。

发送消息

IBM JGSS 提供了两组方法来保护消息：wrap() 和 getMIC()。

使用 wrap()

wrap 方法执行以下操作:

- 计算 MIC
- 对消息进行加密（可选）
- 返回记号

调用应用程序使用 MessageProp 类连同 GSSContext 来指定是否将消息加密。

返回的记号包含消息的 MIC 和文本。消息的文本是密文（对于加密的消息）或原始的纯文本（对于没有加密的消息）。

使用 getMIC()

getMIC 方法执行以下操作，但不能对消息进行加密:

- 计算 MIC
- 返回记号

返回的记号仅包含计算的 MIC，而不包括原始消息。所以，除了将 MIC 记号传送至对等系统外，必须以某种方式使对等系统知道原始消息，以便它可以验证 MIC。

示例: 使用 per-message 服务来发送消息

以下示例显示一个对等系统（foo）如何包装一条消息以发送到另一个对等系统（superSecureServer）:

```
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
```

接收消息

包装的消息的接收方使用 unwrap() 将消息解码。unwrap 方法执行以下操作:

- 验证消息中嵌入的加密 MIC
- 返回发送方计算 MIC 所基于的原始消息

如果发送方已将消息加密，unwrap() 在验证 MIC 之前将消息解密，然后返回原始的纯文本消息。MIC 记号的接收方使用 verifyMIC() 来基于给定消息验证 MIC。

对等应用程序使用其自己的协议来互相传送 JGSS 上下文和消息记号。对等应用程序还必须定义用于确定记号是 MIC 还是包装的消息的协议。例如，这种协议的一部分可能与 “简单认证和安全性层”（SASL）应用程序所使用的协议一样简单（和严密）。SASL 协议指定上下文接受方始终是在建立上下文后发送 per-message（已包装）记号的第一个对等系统。

有关更多信息，参见简单认证和安全性层（SASL） 。

示例: 使用 per-message 服务来接收消息

以下示例显示对等系统（superSecureServer）如何将它从另一个对等系统（foo）中接收到的已包装的记号解包:

```
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.
```

删除上下文

当不再需要上下文时，对等系统将删除上下文。在 JGSS 操作中，每个对等系统单方面地决定何时删除上下文而不需要通知其对等系统。

JGSS 未定义删除上下文记号。要删除上下文，对等系统调用 GSSContext 对象的处置方法来释放上下文所使用的任何资源。除非应用程序将对象设置为空，否则已处置的 GSSContext 对象仍是可访问的。然而，使用已处置的（但仍是可访问的）上下文的任何尝试都将抛出一个异常。 

将 JAAS 与 JGSS 应用程序配合使用

IBM JGSS 包括一个可选的 JAAS 登录设施，它允许应用程序使用 JAAS 来获得凭证。在 JAAS 登录设施将主体凭证和密钥保存在 JAAS 登录上下文的主题对象中后，JGSS 可以从该主题检索凭证。

JGSS 的缺省行为是从主题检索凭证和密钥。通过将 Java 属性 javax.security.auth.useSubjectCredsOnly 设置为 false，可以禁用此功能部件。

注意: 尽管纯 Java JGSS 提供程序可以使用登录界面，但本机 iSeries JGSS 提供程序却不能使用。

有关 JAAS 功能部件的更多信息，参见获得 Kerberos 凭证和密钥。

要使用 JAAS 登录设施，应用程序必须以下列方式遵循 JAAS 编程模型:

- 创建 JAAS 登录上下文
- 在 JAAS Subject.doAs 构造的范围中进行操作

以下代码片断举例说明了在 JAAS Subject.doAs 构造的范围中进行操作的概念:

```
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);
}
```

调试

当尝试标识 JGSS 问题时，使用 JGSS 调试能力来生成有帮助的分类消息。通过为 Java 属性 com.ibm.security.jgss.debug 设置适当的值，可以打开一个或多个类别。通过使用逗号来隔开类别名激活多个类别。

调试类别包括:

类别	描述
help	列示调试类别
all	对所有类别打开调试
off	完全关闭调试
app	应用程序调试（缺省值）
ctx	上下文操作调试
cred	凭证（包括名称）操作
marsh	记号的编组
mic	MIC 操作
prov	提供程序操作
qop	QOP 操作
unmarsh	记号的取消编组
unwrap	取消回绕操作
wrap	回绕操作

JGSS 调试类

要有计划地调试 JGSS 应用程序，在 IBM JGSS 框架中使用调试类。应用程序可以使用调试类来打开和关闭调试类别并显示活动类别的调试信息。

缺省调试构造函数读取 Java 属性 com.ibm.security.jgss.debug 来确定要激活（打开）哪些类别。

示例: 调试应用程序类别

以下示例显示如何请求应用程序类别的调试信息:

```
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.
```

样本: IBM Java 一般安全性服务（JGSS）

“IBM Java 一般安全性服务”（JGSS）样本文件包括客户机和服务器程序、配置文件、策略文件和 javadoc 参考信息。

可以查看样本的 HTML 版本或下载样本程序的 javadoc 信息和源代码。下载样本使您能够查看 javadoc 参考信息、检查代码、编辑配置和策略文件并编译和运行样本程序:

- 查看样本的 HTML 版本
- 下载和查看样本 javadoc 信息
- 下载和运行样本程序

样本程序的描述

JGSS 样本包括四个程序:

- 非 JAAS 服务器
- 非 JAAS 客户机
- 启用 JAAS 的服务器
- 启用 JAAS 的客户机

启用 JAAS 的版本与它们的非 JAAS 对应版本完全可以互操作。因此，可以对非 JAAS 服务器运行启用 JAAS 的客户机，并且可以对启用 JAAS 的服务器运行非 JAAS 客户机。

注意: 当运行样本时，可以指定一个或多个可选 Java 属性，包括配置和策略文件的名称、JGSS 调试选项和安全性管理器。还可以打开和关闭 JAAS 功能部件。

可以在单服务器或双服务器配置中运行样本。单服务器配置由一个与主服务器通信的客户机组成。双服务器配置由一个主服务器和一个辅助服务器组成，其中主服务器用作辅助服务器的启动方或客户机。

当使用双服务器配置时，客户机首先启动上下文并与主服务器交换安全消息。下一步，客户机将其凭证委托给主服务器。然后，主服务器代表客户机使用这些凭证来启动上下文并与辅助服务器交换安全消息。也可以使用双服务器配置，其中主服务器用作自己的客户机。在这种情况下，主服务器使用自己的凭证来启动上下文并与辅助服务器交换安全消息。

可以对主服务器同时运行任何数目的客户机。尽管可以直接对辅助服务器运行客户机，但辅助服务器不能使用委托的凭证或使用自己的凭证作为启动方运行。 

查看 IBM JGSS 样本

“IBM Java 一般安全性服务”（JGSS）样本文件包括客户机和服务器程序、配置文件、策略文件和 javadoc 参考信息。使用以下链接来查看 JGSS 样本的 HTML 版本。

有关附加信息，参见下列主题:

- 样本程序的描述

- 下载和运行样本程序

查看样本程序

通过使用以下链接来查看 JGSS 样本程序的 HTML 版本:

- 样本非 JAAS 客户机程序
- 样本非 JAAS 服务器程序
- 样本启用 JAAS 的客户机程序
- 样本启用 JAAS 的服务器程序

查看样本配置和策略文件

通过使用以下链接来查看 JGSS 配置和策略文件的 HTML 版本:

- Kerberos 配置文件
- JAAS 配置文件
- JAAS 策略文件
- Java 策略文件

样本: IBM JGSS 非 JAAS 客户机程序

有关使用样本客户机程序的更多信息，参见下载和运行样本程序。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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]\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 GSSException(GSSException.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");
    }
}
```

样本: IBM JGSS 非 JAAS 服务器程序

有关使用样本服务器程序的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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 scondary 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 taking 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;

    private static boolean beenInitiator     = false;

    private static final String usageString =
         "\t[-?] [-# number] [-d | -n name] [-p port]"
```

```
   + "\n\t[-s serverName] [-h serverHost [:port]] [-P serverPort] [- msg]"
   + "\n"
   + "\n  -?\t\t\thelp; produces this message"
   + "\n  -# number\t\tWhether primary or secondary server"
   +         " \n\t\t\t(1 = primary, 2 = secondary; default = first)"
   + "\n  -n name\t\tthe server's principal name (without realm)"
   + "\n  -p port\t\tthe port on which the server will be listening"
   + "\n  -s serverName\t\tsecondary server's principal name"
   +         " (without realm)"
   + "\n  -h serverHost[:port]\tsecondary server's hostname"
   +         " (and optional port number)"
   + "\n  -P port\t\tsecondary server's port number"
   + "\n  -m msg\t\tmessage to send to secondary server"
   + "\n  -b \t\trun as both client and server"
   +         " using the server's owns credentials";

// Non-static variables are thread-specific
// since each thread runs a separate instance of this class.

private String debugPrefix = null;
private TCPComms tcp       = null;

static {
    try {
        testUtil = new Util();
    } catch (Exception exc) {
        exc.printStackTrace();
        System.exit(1);
    }
}

Server (Socket socket) throws Exception
{
    debugPrefix = program + ": ";
    tcp = new TCPComms(socket);
}

Server (String program) throws Exception
{
    debugPrefix = program + ": ";
    this.program = program;
}

Server (String program, boolean useSubjectCredsOnly) throws Exception
{
    this(program);
    setUseSubjectCredsOnly(useSubjectCredsOnly);
}

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 ? "true" : "false"));

        // 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 GSSException
{
   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()); // XXXXXXX
    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;
        }
    }
}
```

样本: IBM JGSS 启用 JAAS 的客户机程序

有关使用样本客户机程序的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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 ...");
    }
}
```

样本: IBM JGSS 启用 JAAS 的服务器程序

有关使用样本服务器程序的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// 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()); // XXXXXXX

        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;
        }
    }
}
```

样本: Kerberos 配置文件

有关使用样本配置文件的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
# --------------------------------------------------------------------------------
# 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_enctypes  = des-cbc-crc
 default_tgs_enctypes  = 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
```

样本: JAAS 登录配置文件

有关使用样本配置文件的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
/**
 * --------------------------------------------------------------------------------
 * JGSS 样本 的 JAAS 登录配置。
 * --------------------------------------------------------------------------------
 *
 * 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;
};
```

样本: JAAS 策略文件

有关使用样本策略文件的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// -----------------------------------------------------------------------------
// 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";
};
```

样本: Java 策略文件

有关使用样本策略文件的更多信息，参见下载和运行 IBM JGSS 样本。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
// -------------------------------------------------------------------------
// 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";
};
```

样本: 下载和查看 IBM JGSS 样本的 javadoc 信息

要下载和查看 IBM JGSS 样本程序的文档，完成以下步骤:

1. 选择一个要存储 javadoc 信息的现有目录（或创建一个新目录）。
2. 将 javadoc 信息（jgsssampledoc.zip）下载到该目录中。
3. 将 jgsssampledoc.zip 中的文件解压缩到该目录中。
4. 使用浏览器来访问 index.htm 文件。

代码示例不保证声明

IBM 授予您使用所有编程代码示例的非专有版权许可证，您可以由此生成相似的定制功能以满足您特定的需要。

IBM 提供所有样本代码只是出于解释的目的。并未在所有环境下完全测试这些示例。因此，IBM 不保证或默示这些程序的可靠性、可服务性和功能。

本文档中包含的所有程序是以“按现状”的基础提供的，不附有任何形式的保证。明示的不保证声明包括非侵权性、适销性和适用于某特定用途的默示保证。 

样本: 下载和运行样本程序

在修改或运行样本之前，阅读样本程序的描述。

要运行样本程序，执行以下任务:

1. 将样本文件下载到 iSeries 服务器
2. 准备运行样本文件
3. 运行样本程序

有关如何运行样本的更多信息，参见示例: 运行非 JAAS 样本。

代码示例不保证声明

IBM 授予您使用所有编程代码示例的非专有版权许可证，您可以由此生成相似的定制功能以满足您特定的需要。

IBM 提供所有样本代码只是出于解释的目的。并未在所有环境下完全测试这些示例。因此，IBM 不保证或默示这些程序的可靠性、可服务性和功能。

本文档中包含的所有程序是以“按现状”的基础提供的，不附有任何形式的保证。明示的不保证声明包括非侵权性、适销性和适用于某特定用途的默示保证。 

样本: 下载 IBM JGSS 样本

在修改或运行样本之前，阅读样本程序的描述。

要下载样本文件并将其存储到 iSeries 服务器上，完成以下步骤:

1. 在 iSeries 服务器上，选择一个要存储样本程序、配置文件和策略文件的现有目录（或创建一个新目录）。
2. 下载样本程序（ibmjgsssample.zip）。
3. 将 ibmjgsssample.zip 中的文件解压缩至服务器上的目录中。

解压缩 ibmjgsssample.jar 的内容将执行以下操作:

- 将包含样本 .class 文件的 ibmgjsssample.jar 置于选择的目录中

- 创建一个包含配置和策略文件的子目录（名为 config）
- 创建一个包含样本 .java 源文件的子目录（名为 src）

相关信息
您可能要阅读相关任务或查看示例:
- 准备运行样本文件
- 运行样本程序
- 示例: 运行非 JAAS 样本

样本: 准备运行样本程序
在修改或运行样本之前，阅读样本程序的描述。

下载源代码之后，需要执行以下任务后才能运行样本程序:
- 编辑配置和策略文件以适合环境。有关更多信息，参考每个配置和策略文件中的注释。
- 确保 java.security 文件包含 iSeries 服务器的正确设置。有关更多信息，参见 Java 主安全性属性文件。
- 将修改后的 Kerberos 配置文件（krb5.conf）置于 iSeries 服务器上适合于您正在使用的 J2SDK 的版本的目录中:
  - 对于 J2SDK 的 V1.3: /QIBM/ProdData/Java400/jdk13/lib/security
  - 对于 J2SDK 的 V1.4: /QIBM/ProdData/Java400/jdk14/lib/security

相关信息
您可能要阅读相关任务或查看示例:
- 将样本文件下载到 iSeries 服务器
- 运行样本程序
- 示例: 运行非 JAAS 样本

样本: 运行样本程序
在修改或运行样本之前，阅读样本程序的描述。

在下载并修改源代码之后，可以运行样本之一。

要运行样本，必须首先启动服务器程序。在启动客户机程序之前，服务器程序必须正在运行并已准备好接收连接。当您看到 `listening on port <server_port>`时，服务器已准备好接收连接。确保记住或写下 <server_port>，它是在启动客户机时需要指定的端口号。

使用以下命令来启动样本程序:

```
java [-Dproperty1=value1 ... -DpropertyN=valueN] com.ibm.security.jgss.test.<program> [options]
```

其中
- [-DpropertyN=valueN] 是一个或多个可选 Java 属性，包括配置和策略文件的名称、JGSS 调试选项和安全性管理器。有关更多信息，参见以下示例和运行 JGSS 应用程序。
- <program> 是一个必需的参数，它指定要运行的样本程序（`Client`、`Server`、`JAASClient`或 `JAASServer`）。

- [options] 是要运行的样本程序的可选参数。要显示受支持的选项的列表，使用以下命令:

```
java com.ibm.security.jgss.test.<program> -?
```

注意: 通过将 Java 属性 javax.security.auth.useSubjectCredsOnly 设置为 false，关闭启用 JGSS 的样本中的 JAAS 功能部件。当然，启用 JAAS 的样本的缺省值是打开 JAAS，这意味着属性值为 true。除非您已显式地设置属性值，否则非 JAAS 客户机和服务器程序将该属性设置为 false。

相关信息

可能要阅读相关任务或查看示例:

- 将样本文件下载至 iSeries 服务器
- 准备运行样本文件
- 示例: 运行非 JAAS样本«

»

IBM JGSS javadoc 参考信息

IBM JGSS 的 javadoc 参考信息包括 org.ietf.jgss api 包中的类和方法和某些 Kerberos 凭证管理工具的 Java 版本。

尽管 JGSS 包括几个可公开访问的包（例如，com.ibm.security.jgss 和 com.ibm.security.jgss.spi），但您应只使用标准化 org.ietf.jgss 包中的 API。仅使用此包可以确保应用程序符合 GSS-API 规范并确保最优的互操作性和可移植性。

- org.ietf.jgss
- kinit
- ktab
- klist«

第 5 章 使用 IBM Developer Kit for Java 来调整 Java 程序性能

在为 iSeries 服务器构建 Java 应用程序时，Java(TM) 应用程序性能有几个方面应予以考虑。以下是一些指向有关如何获得更佳性能的详细信息和提示的链接:

- 在运行 Java 类文件、JAR 文件或 ZIP 文件之前，使用“创建 Java 程序”（CRTJVAPGM）命令来改进 Java 代码的运行时性能。
- 更改优化级别以获得最佳的静态编译性能。
- 谨慎地设置值以获得最佳的垃圾收集性能。
- 仅使用本机方法来启动运行时间相对较长，且不能在 Java 中直接使用的系统函数。
- 在编译期使用 `javac -o` 选项来执行方法内联并显著改进方法调用性能。
- 在应用程序运行流不正常时，使用 Java 异常。

将这些工具与“性能浏览器”（PEX）配合使用来找出 Java 程序中的性能问题:

- 可以使用 iSeries Java 虚拟机来收集 Java 跟踪事件。
- 要确定每个 Java 方法所花的时间，请使用 Java 调用跟踪。
- Java 记入概要文件确定 Java 程序使用的每个 Java 方法以及所有系统函数所花的相对 CPU 时间量。
- 使用 Java 性能数据收集器来提供有关在 iSeries 服务器上运行的程序的概要文件信息。

任何作业会话都可启动和结束 PEX。通常，收集的数据是关于整个系统的，且与系统上的所有作业（包括 Java 程序）相关。有时，可能有必要从 Java 应用程序中启动和停止性能收集。这能缩短收集时间，并能减少调用或返回跟踪通常生成的大量数据。PEX 不能从 Java 线程中运行。要启动和停止收集，您需要编写通过队列或共享内存与独立作业通信的本机方法。然后，第二个作业在适当的时间启动和停止收集。

除了应用程序级性能数据之外，您还可以使用现有的 iSeries 系统级性能工具。这些工具逐个 Java 线程地报告统计信息。

有关 PEX 报告的示例，参见 Performance Tools for iSeries（SC41-5340）一书。 

Java 运行时性能注意事项

要大幅改进 Java(TM) 代码的运行时性能，请在运行 Java 类文件、JAR 文件或 ZIP 文件之前使用“创建 Java 程序”（CRTJVAPGM）命令。CRTJVAPGM 命令使用字节码来创建 Java 程序（它包含经过优化的 iSeries 服务器本机指令），并将 Java 程序对象与类文件、JAR 文件或 ZIP 文件相关联。

如果在运行 Java 类文件、JAR 文件或 ZIP 文件之前未使用 CRTJVAPGM 命令，则 Java 代码第一次运行时的速度将慢得多，这是因为要创建优化级别为 10 的优化 Java 程序。因为将把该 Java 程序保存下来，且保持与类文件或 JAR 文件相关联，所以后续的运行速度将会快很多。在应用程序开发期间，以解释方式运行字节码能提供可接受的性能，但在生产环境中，您可能想要在运行 Java 代码之前使用 CRTJVAPGM 命令。

“及时”编译器通过编译代码来改进性能，它针对特定 Java 虚拟机运行时环境来优化代码。您应该将不必执行 CRTJVAPGM 的好处与使用所 JIT 带来的略微降低程序启动和运行时处理的速度作比较，以决定适合于您的需要的优化技术。

如果程序运行缓慢，则请输入“显示 Java 程序”（DSPJVAPGM）命令来查看 Java 程序的属性。

高速缓存类装入程序

通过允许对用户类装入程序创建的 JVAPGM 进行高速缓存以便重复使用，可以改进用户类装入程序的性能。除非在下列其中一项中指定启用 Java 属性，否则此选项未启用:

- /QIBM/UserData/Java400/SystemDefault.properties
- /home//SystemDefault.properties
- 在 RUNJVA CL 命令上
- 在 JAVA QSH 命令上

以下属性用来启用此功能:

os400.define.class.cache.file

此属性的值应该是有效 “Java 压缩文档” （JAR）文件的名称（具有完整的路径）。JAR 文件必须包含有效的 JAR 目录（由 jar QSH 命令构建），但除了所必需的单一成员之外不能包含其它内容才能使 jar 能够工作。不应将这个 JAR 文件包括在任何 Java 类路径中。

例如，将下面这一行添加至 /QIBM/UserData/Java400/SystemDefault.properties:

```
os400.define.class.cache.file=/QIBM/ProdData/Java400/QDefineClassCache.jar
```

Developer Kit for Java 将安装合适的 JAR 文件来作为 /QIBM/ProdData/Java400/QDefineClassCache.jar，并安装示例属性文件来作为 /QIBM/ProdData/Java400/SystemDefaultCacheExample.properties。可通过将 SystemDefaultCacheExample.properties 文件复制到 /QIBM/UserData/Java400/ 并将其重命名为 SystemDefault.properties 来以全局方式启用高速缓存功能。

注意: 在进行此项更改之前，确保该目录中没有现有的 SystemDefault.properties 文件。另外，为了只影响单一用户的环境，可将该文件复制到 /home// 并进行类似的重命名。

对于对高速缓存指定的 JAR 文件，可使用 DSPJVAPGM 来确定已对多少个 JVAPGM 进行了高速缓存。DSPJVAPGM 屏幕的 **Java 程序**字段指示了已对多少个 JVAPGM 进行了高速缓存，**Java 程序大小**字段指示了高速缓存的 JVAPGM 消耗了多少存储器。当将 DSPJVAPGM 应用于用于高速缓存的 JAR 文件时，此屏幕的其它字段没有意义。

除了用于启用高速缓存的 os400.define.class.cache.file Java 属性之外，可指定另外两个属性来控制高速缓存特征。

os400.define.class.cache.hours

此属性可用来指定应该将 JVAPGM 保留在高速缓存中多长时间（以小时计）。

os400.define.class.cache.maxpgms

此属性可用来指定最多可以对多少个 JVAPGM 进行高速缓存，当超过此限制时，将首先删除最旧的 JVAPGM。每当引用 JVAPGM 时都将更新用来确定高速缓存的 JVAPGM 的年龄的时间值。

示例:

```
    os400.define.class.cache.hours=48
os400.define.class.cache.maxpgms=10000
```

os400.define.class.cache.hours 的缺省值是 168 个小时（一星期），最大值是 9999。os400.define.class.cache.maxpgms 的缺省值是 5000，最大值是 40000。如果对这些属性之一指定零值，或者不能将值分析为有效的十进制数，则使用缺省值。

有关前面对高速缓存提到的属性的更多信息，参见 系统属性。

选择运行 Java 程序时要使用的方式

在运行 Java(TM) 程序时，您可以选择希望使用的方式。所有方式都验证代码并创建 Java 程序对象来存放预验证格式的程序。可使用下列任何一种方式:

- 解释方式
- 直接处理
- “及时”（JIT）编译
- “及时”（JIT）编译并直接处理

选择方式	详细信息
解释方式	在运行时对每个字节码进行解释。 有关以解释方式运行 Java 程序的信息，参见运行 Java（RUNJVA）命令。
直接处理	方法的机器指令在首次调用该方法期间生成并保存下来，以供程序下次运行时使用。并且，整个系统还共享一个副本。 有关使用直接处理方式来运行 Java 程序的信息，参见运行 Java（RUNJVA）命令。

“及时”（JIT）编译	方法的机器指令在首次调用该方法期间生成，并在 Java 虚拟机的运行期间得到保存。
	
	要使用“及时”编译器，需要将编译器值设置为 `jitc`。可以通过添加环境变量或设置 java.compiler 系统属性来设置该值。请从以下列表中选择一种方法来设置编译器值:
	• 从 iSeries 服务器的命令行提示符下，通过使用“添加环境变量”（ADDENVVAR）命令来添加环境变量。然后，使用“运行 Java”（RUNJVA）命令或 JAVA 命令来运行 Java 程序。例如，使用:
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc)`
	`JAVA CLASS(Test)`
	• 在 iSeries 命令行上设置 java.compiler 系统属性。例如，输入 `JAVA CLASS(Test) PROP((java.compiler jitc))`
	• 在 Qshell Interpreter 命令行上设置 java.compiler 系统属性。例如，输入 `java -Djava.compiler=jitc Test`
	
	在设置此值之后，JIT 编译器便在运行 Java 代码之前对所有 Java 代码执行优化。

选择方式	详细信息

“及时”（JIT）编译并直接处理	最常见的使用“及时”（JIT）编译器的方法是使用 jit_de 选项。当使用此选项运行时，已对直接处理进行了优化的程序会以直接处理方式运行。未对直接优化进行优化的程序以 JIT 方式运行。
	
	要将 JIT 方式与直接处理方式配合使用，您需要将编译器值设置为 jitc_de。可以通过添加环境变量或设置 java.compiler 系统属性来设置该值。请从以下列表中选择一种方法来设置编译器值:
	• 通过在 iSeries 命令行上输入“添加环境变量”（ADDENVVAR）命令来添加环境变量。然后，使用“运行 Java”（RUNJVA）命令或 JAVA 命令来运行 Java 程序。例如，输入
	`ADDENVVAR ENVVAR (JAVA_COMPILER) VALUE(jitc_de)`
	`JAVA CLASS(Test)`
	• 在 iSeries 命令行上设置 java.compiler 系统属性。例如，输入 `JAVA CLASS(Test) PROP((java.compiler jitc_de))`
	• 在 Qshell Interpreter 命令行上设置 java.compiler 系统属性。例如，输入 `java -Djava.compiler=jitc_de Test`
	
	在设置此值之后，将使用为直接处理而创建的类文件的 Java 程序。如果该 Java 程序不是为直接处理创建的，则 JIT 在运行之前优化类文件。有关更多信息，参见“及时”编译器与直接处理的比较。

运行 Java 程序的方法有三种（CL、QSH 和 JNI）。每一种方法都有唯一的方法来指定方式。下表显示了如何完成此操作。

方式	CL 命令	QShell 命令	JNI 调用 API
解释	INTERPRET(*YES)	-Djava.compiler=NONE -interpret	os400.run.mode=″interpret″
DE	INTERPRET(*NO)	-Djava.compiler=NONE	• os400.run.mode=″program_created=pc″ • os400.create.type=″direct″
JIT	INTERPRET(*JIT)	-Djava.compiler=″jitc″	os400.run.mode=″jitc″
JIT_DE（缺省）	INTERPRET(*OPTIMIZE) OPTIMIZE(*JIT)	-Djava.compiler=″jitc_de″	os400.run.mode=″jitc_de″

Java 解释器

Java(TM) 解释器是 Java 虚拟机的一部分，它为特定硬件平台解释 Java 类文件。Java 解释器对每个字节码进行运行，并对该字节码运行一系列机器指令。

静态编译

Java(TM) 转换程序是一个 IBM Operating System/400（OS/400）组件，它对类文件进行预处理，以便为使用 iSeries Java 虚拟机来运行它们作好准备。Java 转换程序创建持续的且与类文件相关联的已优化程序对象。在缺省情况下，该程序对象包含该类的经过编译的 64 位 RISC 机器指令版本。Java 解释器在运行时不解释已优化的程序对象。而是，该程序对象在装入类文件时直接运行。

缺省情况下，使用 JIT 来对 Java 程序进行优化。要使用 Java 转换程序，请执行 CRTJVAPGM，或在 RUNJVA 或 JAVA 命令上指定要使用转换程序。

可以使用创建 Java 程序（CRTJVAPGM）命令来显式地启动 Java 转换程序。当 CRTJVAPGM 命令运行时，它优化类文件或 JAR 文件，因此当程序运行时就不需要执行任何操作。这改进了程序首次运行时的速度。通过使用 CRTJVAPGM 命令（而不是依赖于缺省优化），确保了最佳优化是有可能的，并且改进了与类文件或 JAR 文件相关联的 Java 程序空间的使用。

对类文件、JAR 文件或 ZIP 文件使用 CRTJVAPGM 命令将导致优化文件中的所有类，且生成的 Java 程序对象将是持续的。这样可改进运行时性能。通过使用 CRTJVAPGM 命令或更改 Java 程序（CHGJVAPGM）命令，您还可以更改优化级别或选择除缺省优化级别 10 之外的优化级别。在优化级别 40，内部类绑定在 JAR 文件中的类之间执行，在某些情况下，这些类是内联的。内部类绑定改进了调用速度。内联完全除去了方法调用的开销。在某些情况中，可以在 JAR 文件或 ZIP 文件中的类之间内联方法。在 CRTJVAPGM 命令上指定 `OPTIMIZE(*INTERPRET)` 会导致该命令上指定的任何类都被验证，并准备以解释方式运行。

运行 Java（RUNJVA）命令还可指定 OPTIMIZE(*INTERPRET)。此参数指定无论关联程序对象的优化级别是多少，都对 Java 虚拟机中运行的任何类都进行解释。当调试已使用优化级别 40 变换的类时，这特别有用。要强制进行解释，请使用 INTERPRET(*YES)。

» 有关重复使用由类装入程序创建的 Java 程序的信息，参见高速缓存类装入程序。 «

Java 静态编译性能注意事项

您可以通过所设置的优化级别来确定变换速度。优化级别 10 的变换速度最快，但生成的程序的运行速度通常比更高优化级别所生成的程序慢。优化级别 40 的变换速度较慢，但生成的程序的运行速度可能较快。

少量的 Java(TM) 程序不能优化至级别 40。因此，少数不能在级别 40 运行的程序可能会在级别 30 运行。您可以使用许可内码优化 LICOPT 参数字符串来运行不能在优化级别 40 运行的程序。然而，级别 30 的性能对程序而言可能已足够了。

如果运行在另一个 Java 虚拟机上似乎能工作的 Java 代码时出现问题，则请尝试使用优化级别 30，而不是 40。如果这能起作用，且性能可以接受，则您无需再进行任何其他操作。如果您需要更好的性能，则请参见 LICOPT 参数字符串以了解有关如何启用和禁用各种形式的优化的信息。例如，您可以首先尝试使用 OPTIMIZE(40) LICOPT(NoPreresolveExtRef) 来创建程序。如果应用程序包含对不可用的类的死调用，则这个 LICOPT 值会允许程序运行，而不会有任何问题。

要确定创建 Java 程序时所使用的优化级别，可以使用显示 Java 程序（DSPJVAPGM）命令。要更改 Java 程序的优化级别，请使用创建 Java 程序（CRTJVAPGM）命令。

“及时”编译器

“及时”（JIT）编译器是一个特定于平台的编译器，在首次调用每个方法时，它生成该方法的机器指令。为了获得比解释方式高的性能，JIT 编译器会根据您的需要来编译代码。

要了解 JIT 编译器与直接处理之间的差异，参见 JIT 编译器与直接处理的比较。

有关 java.compiler 属性的信息，请参考 Java(TM) 系统属性。请选择正在运行的版本。

“及时”编译器与直接处理的比较

如果您正在尝试决定是使用“及时”编译器还是使用直接处理方式来运行 Java(TM) 程序，则此表提供的附加信息可帮助您根据自己的情况来作出最佳选择。

“及时”编译器	直接处理
当需要时，提供任何方法的自动编译。此方法比直接处理快得多。	使用“创建 Java 程序”（CRTJVAPGM）命令来编译整个类或 JAR 文件，或者在运行时自动编译文件。
在代码不断更改的程序开发期间使用，以裁剪优化成本。还在小型或使用率相对较低的应用程序的部署期间使用。还可以与在运行时生成或发现代码的高度动态应用程序配合使用。	用于相对较大的应用程序。优化级别为 40 的直接处理代码通常比 JIT 要快。大多数已准备好部署的服务器应用程序使用优化级别为 40 的直接处理，这是因为它们有可能会被多个用户在任何给定的时间使用，而且，反复使用 JIT 的开销确实太高了。
仅限于执行那些可以在运行时快速执行的优化。	因为优化不是在运行时执行的，所以有可能进行更复杂的优化。

优化级别

通过在**优化级别**字段中输入值，指定与类文件对象或 JAR 文件对象相连的 Java(TM) 程序的优化级别。通过使用此选项，可以控制 Java 程序的大小和性能。对于所列示的优化级别，内部格式包括 iSeries 机器指令。这些机器指令根据指定的优化级别进行优化。当程序运行时，iSeries 服务器直接运行机器指令。

以下列表显示了各个优化级别的差异以及它们的作用:

10

Java 程序包含类文件字节码的转换版本，但只作了最小量的附加编译器优化。您可以在调试时显示和更改变量。对于 V5R1 和更高版本，优化级别 10 可能会毫无必要地延长编译程序所需的时间。请使用优化级别 20 或更高。

20

Java 程序包含类文件字节码的编译版本，并且执行附加编译器优化。您可以在调试时显示变量，但不能更改变量。

30

Java 程序包含类文件字节码的编译版本，并且执行比优化级别 20 更多的编译器优化。优化级别越高，调试也越困难，这是因为优化降低了在代码中的精确位置停止及显示程序变量的能力。您可以在调试时显示变量，但不能更改变量。呈示的值可能不是变量的当前值。

40

Java 程序包含类文件字节码的编译版本，并且执行比优化级别 30 更多的编译器优化。优化级别 40 包括跨类优化。在少数情况下，静态初始化程序对不相关类（没有通过继承或包含关系实现相关）的运行次序可能与静态初始化说明中所概述的次序不同。另外，它还包括禁用调用和指令跟踪的优化。

注意: 当优化级别为 40 时，如果 Java 程序未能优化或抛出异常，则请使用优化级别 30。

Java 垃圾收集

垃圾收集是释放程序不再引用的对象所使用的存储器的过程。借助垃圾收集，程序员不必再编写用来显式“释放”或“删除”对象的易出错的代码了。此代码经常会导致“内存泄露”程序错误。垃圾收集器自动检测用户程序不能再访问的对象或对象组。它执行此操作的依据是任何程序结构都不再引用该对象。在收集了该对象之后，便可以分配空间另作他用。

Java(TM) 运行时环境包括一个用来释放不再使用的内存的垃圾收集器。垃圾收集器在有需要时会自动运行。

也可以在使用 java.lang.Runtime.gc() 方法来在 Java 程序的控制下显式启动垃圾收集器。

有关 IBM Developer Kit for Java 的特定说明，参见 IBM Developer Kit for Java 高级垃圾收集。

IBM Developer Kit for Java 高级垃圾收集

IBM Developer Kit for Java(TM) 实现了高级垃圾收集器算法。此算法允许发现和收集无法访问的对象，而 Java 程序的操作不会发生明显的暂停。并行收集器协作地发现正在运行的线程（而不是单一线程）下的对象引用。

许多垃圾收集器是“单独运行”的。这意味着发生收集周期时，除了执行垃圾收集的线程之外的所有线程在垃圾收集器工作时都会停止下来。发生这种情况时，Java 程序将暂停下来，并且，当收集器工作时，相对于 Java, 平台的任何多处理器能力都浪费掉了。iSeries 算法不会同时停止所有程序线程。它允许那些线程在垃圾收集器完成其任务时继续操作。这就防止了出现暂停，并允许在垃圾收集期间使用所有处理器。

垃圾收集根据您启动 Java 虚拟机时指定的参数自动发生。也可以通过使用 java.lang.Runtime.gc() 方法来在 Java 程序的控制下显式启动垃圾收集。

有关基本定义，参见 Java 垃圾收集。

Java 垃圾收集性能注意事项

iSeries Java(TM) 虚拟机上的垃圾收集是以连续异步方式操作的。“运行 Java”（RUNJVA）命令上的垃圾收集初始大小（GCHINL）参数可能会影响应用程序性能。此参数指定两次垃圾收集之间允许的新对象空间大小。较小的值可能会导致垃圾收集的开销过大。较大的值可能会限制垃圾收集并导致内存溢出错误。然而，对于大多数应用程序而言，缺省值应该是正确的。

垃圾收集通过评定是否存在对某个对象的任何有效引用来确定是否不再需要该对象。

Java 本机方法调用性能注意事项

iSeries 服务器上的本机方法调用的性能可能不如其它平台上的本机方法调用。iSeries 服务器上的 Java(TM) 已通过将 Java 虚拟机移到机器接口（MI）之下进行了优化。本机方法调用要求在 MI 代码之上进行调用，并可能要求使用“Java 本机接口”（JNI）调用来调用回到 Java 虚拟机中，其成本很高。本机方法不应执行小型例程，小型例程可以很容易使用 Java 编写。您只应使用本机方法来启动那些运行时间相对较长并且不能直接在 Java 中使用的系统函数。

Java 方法内联性能注意事项

方法内联可以显著改进方法调用性能。任何最终方法都是内联方法的潜在候选。在 iSeries 服务器上，内联功能是在编译期通过 `javac -o` 选项获得的。如果使用 `javac -o` 选项，则类文件和变换后的 Java(TM) 程序的大小会增大。在使用 -o 选项时，您应该考虑应用程序的空间和性能属性。

Java 转换程序对优化级别 30 和优化级别 40 启用内联。优化级别 30 允许在单个类中内联一些最终方法。优化级别 40 允许在 ZIP 文件或 JAR 文件中内联最终方法。您可以使用 AllowInlining 和 NoAllowInlining LICOPT 参数字符串来控制方法的内联。iSeries 解释器不执行方法内联。

Java 异常性能注意事项

iSeries 异常体系结构具有反复中断和重试的能力。它允许混合语言交互作用。在 iSeries 服务器上抛出 Java(TM) 异常的成本可能比其它平台更高。除非在常规应用程序路径中例行使用 Java 异常，否则这应该不会影响应用程序的整体性能。

Java 调用跟踪性能工具

Java(TM) 方法调用跟踪提供了关于每个 Java 方法所花费的时间的重要性能信息。在其它 Java 虚拟机上，您可能已在 java 命令使用了 -prof（记入概要文件）选项。要在 iSeries 服务器上启用方法调用跟踪，必须在“创建 Java 程序”（CRTJVAPGM）命令行上指定“启用性能收集”（ENBPFRCOL）命令。在使用此关键字创建 Java 程序之后，便可以通过使用包括调用／返回跟踪类型的“性能浏览器”（PEX）定义来启动方法调用跟踪集合。

使用“打印性能浏览器报告”（PRTPEXRPT）命令生成的调用／返回跟踪输出显示所跟踪的每个 Java 方法的每个调用的中央处理器（CPU）时间。在某些情况下，您可能无法对调用返回跟踪启用所有类文件。或者，您可能正在调用未对跟踪启用的本机方法和系统函数。在这种情况下，花费在这些方法或系统函数中的所有 CPU 时间积累起来。然后，向最后一个被调用且已启用的 Java 方法报告该时间。

Java 事件跟踪性能工具

iSeries Java(TM) 虚拟机允许跟踪特定 Java 事件。收集这些事件不要求在 Java 代码中进行任何特殊处理。这些事件包括活动，如垃圾收集、线程创建、类装入和锁定。“运行 Java”（RUNJVA）命令不指定这些事件。而是，您创建一个“性能浏览器”（PEX）定义并使用“启动性能浏览器”（STRPEX）命令来收集事件。每个事件都包含有用的性能信息，如时间戳记和中央处理器（CPU）周期。可以用同一个跟踪定义来同时跟踪 Java 事件和其他系统活动，如磁盘输入和输出。

有关 Java 事件的完整描述，参见 Performance Tools for iSeries（SC41-5340）一书。 

Java 记入概要文件性能工具

系统范围中央处理器（CPU）记入概要文件计算 Java(TM) 程序使用的每个 Java 方法和所有系统函数所花费的相对 CPU 时间量。使用“性能浏览器”（PEX）定义，它跟踪性能监控器计数器溢出（*PMCO）运行周期事件。通常，每隔一毫秒指定一个样本。要收集有效的跟踪概要文件，应运行 Java 应用程序，直到它积累了两到三分钟的 CPU 时间为止。这样应能生成超过 100000 个样本。“打印性能浏览器报告”（PRTPEXRPT）命令生成在整个应用程序中花费的 CPU 时间的直方图。这包括每个 Java 方法及所有系统级活动。性能数据收集器（PDC）工具提供关于在 iSeries 服务器上运行的程序的概要文件信息。

注意: CPU 属性记录不显示以解释方式运行的 Java 程序的相对 CPU 使用情况。

Java 虚拟机记入概要文件程序接口

» “Java(TM) 虚拟机记入概要文件程序接口”（JVMPI）是一个将 Java 虚拟机（JVM）记入概要文件的实验接口，此接口最初在 Sun 的 Java 2 SDK Standard Edition （J2SDK）V1.2 中发布。

JVMPI 支持在 JVM 和“及时”（JIT）编译器中放置 hook，当激活 hook 时，它们提供事件信息给记入概要文件代理程序。记入概要文件代理程序是作为动态链接库（DLL）实现的。记入概要文件程序将控制信息发送至 JVM 以启用和禁用 JVMPI 事件。例如，记入概要文件程序可能对方法“入口”和“出口”hook 不感兴趣，它可以告诉 JVM 它不想接收这些事件通知。JVM 和 JIT 嵌入了 JVMPI 事件 hook，如果启用事件的话，这些事件 hook 将事件通知发送至记入概要文件代理程序。记入概要文件程序告诉 JVM 它对哪些事件感兴趣，当那些事件发生时，JVM 将事件通知发送给记入概要文件程序。

有关更多信息，参见 Sun Microsystems 的 JVMPI  。 «

收集 Java 性能数据

要收集 iSeries 服务器上的 Java(TM) 性能数据，请执行下列步骤:

1. 创建指定下列各项的“性能浏览器”（PEX）定义:
   - 用户定义名
   - 数据收集类型
   - 作业名
   - 您想要收集其系统信息的系统事件系列

   注意: 如果您想要的输出是 `java_g -prof` 类型的，并且您知道 Java 程序的特定作业名，则 PEX 定义 *STATS 比 *TRACE 定义更可取。

   这里是 *STATS 定义的一个示例:

   ```
   ADDPEXDFN DFN(YOURDFN) JOB(*ALL/YOURID/QJVACMDSRV)
   DTAORG(*HIER) TEXT('your stats definition')
   ```

   这个 *STATS 定义并不会使所有 Java 事件都运行。只将您自己的 Java 会话中的 Java 事件记入概要文件。此种操作方式可能会增加运行 Java 程序的时间。

   以下是 *TRACE 定义的示例:

   ```
   ADDPEXDFN DFN(YOURDFN) TYPE(*TRACE) JOB(*ALL) TRCTYPE(*SLTEVT)
   SLTEVT(*YES) PGMEVT(*JVAENTRY *JVAEXIT)
   ```

   此 *TRACE 定义从您用 ENBPFRCOL(*ENTRYEXIT) 创建的系统中的任何 Java 程序收集任何 Java 入口事件和出口事件。这导致对这种收集类型的分析要比 *STATS 跟踪慢，这取决于 Java 程序事件的数目以及 PEX 数据收集的持续时间。
2. 根据 PEX 定义上的程序事件类别，启用 *JVAENTRY 和 *JVAEXIT，以使 PEX 识别 Java 入口和出口。

   注意: 如果使用“及时”（JIT）编译器来运行 Java 代码，则并不是象为了直接处理而使用 CRTJVAPGM 命令那样启用入口和出口。而是，当您使用 os400.enbprfcol 系统属性时，JIT 生成带有入口和出口 hook 的代码。
3. 使 Java 程序准备好向“iSeries 性能数据收集器”报告程序事件。可通过对您想要报告其性能数据的任何 Java 程序使用创建 Java 程序（CRTJVAPGM）命令来做到这一点。必须使用 ENBPFRCOL(*ENTRYEXIT) 参数来创建 Java 程序。

   注意: 必须对您想要收集其性能数据的每个 Java 程序重复此步骤。如果不执行此步骤，则 PEX 不收集性能数据，并且运行 Java 性能数据转换程序（JPDC）工具也不会生成输出。
4. 通过使用“启动性能浏览器”（STRPEX）命令来启动 PEX 数据收集。
5. 运行要分析的程序。此程序不应该在生产环境中。它将在短时间内生成大量数据。您应将收集时间限制为五分钟。运行这么长时间的 Java 程序将生成许多 PEX 系统数据。如果收集到太多数据，则将需要过长的时间来处理它。

6. 通过使用“结束性能浏览器”（ENDPEX）命令来结束 PEX 数据收集。

   注意: 如果这不是您第一次结束 PEX 数据收集，则必须对替换文件指定 *YES，否则它不会保存数据。
7. 运行 JPDC 工具。
8. 将集成文件系统目录与带有您选择的查看器的系统相连接: java_g -prof 查看器或 Jinsight 查看器。可以从 iSeries 服务器复制此文件，并将它用作任何合适的记入概要文件工具的输入。

性能数据收集器工具

“性能数据收集器”（PDC）工具提供关于在 iSeries 服务器上运行的程序的概要文件信息。

许多 Java(TM) 虚拟机上的工业标准概要文件选项都依赖于 `java_g` 功能部件的实现。这是 Java 虚拟机的特殊调试版本，它提供了 `-prof` 选项。您在调用 Java 程序时指定此选项。指定此选项时，Java 虚拟机生成一个记录文件，该文件中包含关于 Java 程序的哪些部件在该程序的持续期间曾经工作过的信息。Java 虚拟机以实时方式生成此信息。

在 iSeries 服务器上，“性能浏览器”（PEX）功能部件分析程序和特定于记录的系统事件。DB2(R) 数据库存储此信息并使用 SQL 函数检索它。PEX 信息是生成 Java 概要文件数据的特定程序信息的资源库。此概要文件数据与 `java_g -prof` 程序概要文件信息相兼容。Java 性能数据转换程序（JPDC）工具为称为 Jinsight 的特定 IBM 工具提供 `java_g -prof` 程序输出和程序概要文件信息。

有关如何收集 Java 性能数据的信息，参见收集 Java性能数据。

Java 性能数据转换器工具

“Java(TM) 性能数据转换器”（JDPC）工具提供了一种创建关于在 iSeries 服务器上运行的 Java 程序的 Java 性能数据的方法。此性能数据与 Sun Microsystems 的 Java 虚拟机 `java_g -prof` 选项的性能数据输出和 IBM Jinsight 输出相兼容。

注意: JDPC 工具不生成可读的输出。请使用接受 `java_g -prof` 或 Jinsight 数据的 Java 记入概要文件工具来分析数据。

JDPC 工具访问 DB2/400（使用 JDBC）存储的 iSeries“性能浏览器”（PEX）数据。它将数据转换成 Jinsight 或一般性能类型。然后，JDPC 将输出文件存储在集成文件系统中用户指定位置处。

注意: 在 iSeries 服务器上运行指定的 Java 应用程序时，必须遵循适当的 iSeries PEX 数据收集过程来收集 PEX 数据。必须设置一个 PEX 定义，用来定义某个程序或某个收集和存储过程的入口和出口。有关如何收集 PEX 数据和设置 PEX 定义的详细信息，参见 Performance Tools for iSeries（SC41-5340）一书。 

有关如何运行 JPDC 的信息，参见运行 Java 性能数据转换器。

可以使用 Qshell 命令行接口或运行 Java（RUNJVA）命令来启动 JPDC 程序。

运行 Java 性能数据转换器

要运行“Java(TM) 性能数据转换器”（JPDC）来进行性能数据收集，请执行下列步骤。

1. 输入第一个输入自变量，对于 `java_g -prof`，它是 `general`，对于 Jinsight 输出，它为 `jinsight`。
2. 输入第二个输入自变量，它是用于收集数据的“性能浏览器”（PEX）定义名。

   注意: 您应将此名称限定为四或五个字符，因为要在内部使用此名称的连接。
3. 输入第三个输入自变量，它是 JPDC 工具生成的文件的名称。这个生成的文件将写至当前集成文件系统目录。使用 cd (PF4) 命令来指定集成文件系统当前目录。

4. 输入第四个输入自变量，它是 iSeries 主机关系数据库目录项的名称。使用“使用关系数据库目录项”（WRKRDBDIRE）命令来查看该名称。它是唯一一个指示了 *LOCAL 的关系数据库。

要运行此代码，/QIBM/ProdData/Java400/ext/JPDC.jar 文件必须在 iSeries 服务器上的 Java 类路径中。当程序完成运行时，可以在当前目录中找到一个文本输出文件。

可以使用 iSeries 命令行或 Qshell 环境来运行 JPDC。有关详细信息，参见示例：运行 Java 性能数据转换器。

示例：运行 Java 性能数据转换器

可以使用 iSeries 命令行或 Qshell 环境来运行“Java(TM) 性能数据转换器”（JPDC）。

使用 iSeries 命令行:

1. 在 iSeries 命令行上输入“运行 Java”（RUNJVA）命令或 JAVA 命令。
2. 在类参数行上输入 `com.ibm.as400.jpdc.JPDC`。
3. 在参数行上输入 `general pexdfn mydir/myfile myrdbdire`。
4. 在类路径参数行上输入 `'/QIBM/ProdData/Java400/ext/JPDC.jar'`。

   注意：如果 `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 字符串在 CLASSPATH 环境变量中，则可以忽略类路径。可以使用“添加环境变量”（ADDENVVAR）命令、“更改环境变量”（CHGENVVAR）命令或“使用环境变量”（WRKENVVAR）命令来将此字符串添加到 CLASSPATH 环境变量中。

使用 Qshell 环境:

1. 输入“启动 Qshell”（STRQSH）命令来启动 Qshell Interpreter。
2. 在命令行上输入此命令:

```
java -classpath /QIBM/ProdData/Java400/ext/JPDC.jar com.ibm.as400/jpdc/JPDC
jinsight pexdfn mydir/myfile myrdbdire
```

   注意：如果将 `'/QIBM/ProdData/Java400/ext/JPDC.jar'` 字符串添加到当前环境中，则可以省略类路径。可以使用 ADDENVVAR 命令、CHGENVVAR 或 WRKENVVAR 命令来将此字符串添加到当前环境中。

有关背景信息，参见运行 Java 性能数据转换器。

第 6 章 IBM Developer Kit for Java 的命令和工具

当使用 IBM Developer Kit for Java(TM) 时，可以将 Java 工具与 Qshell Interpreter 或 CL 命令配合使用。

如果您以前有 Java 编程经验，则您可能更习惯于使用 Qshell Interpreter Java 工具，这是因为它们与您配合 Sun Microsystems 的 Java Development Kit 使用的工具类似。有关使用 Qshell 环境的信息，参见 Qshell Interpreter。

如果您是 iSeries 程序员，则您可能想要使用对 iSeries 服务器环境很典型的 Java CL 命令。有关使用 CL 命令和 “iSeries 导航器” 命令的更多信息，请继续阅读本节。

可以将下列任何命令和工具与 IBM Developer Kit for Java 配合使用:

- Qshell 环境包含程序开发通常所需的 Java 开发工具。
- CL 环境包含用于优化和管理 Java 程序的 CL 命令。
- iSeries 导航器命令也可创建和运行经过优化的 Java 程序。

IBM Developer Kit for Java 所支持的 Java 工具

IBM Developer Kit for Java(TM) 支持下列工具:

- Java ajar 工具
- Java appletviewer 工具
- Java extcheck 工具
- » Java idlj 工具 «
- Java jar 工具
- Java jarsigner 工具
- Java javac 工具
- Java javadoc 工具
- Java javah 工具
- Java javakey 工具
- Java javap 工具
- Java keytool
- Java native2ascii 工具
- Java policytool
- Java rmic 工具
- Java rmid 工具
- Java rmiregistry 工具
- Java serialver 工具
- Java tnameserv 工具

除少数例外情况之外，Java 工具（ajar 工具除外）支持 Sun Microsystems 规定的语法和选项。它们全都必须使用 Qshell Interpreter 来运行。

可以使用“启动 Qshell”（STRQSH 或 QSH）命令来启动 Qshell Interpreter。当 Qshell Interpreter 运行时，将出现一个“QSH 命令输入”屏幕。在 Qshell 下运行的 Java 工具和程序所生成的所有输出和消息都出现在此屏幕中。Java 程序的所有输入也从此屏幕中读取。有关更多详细信息，参见 Qshell 中的 Java 命令。

注意: 不能直接从 Qshell 中使用 iSeries 命令输入功能。要进行命令输入，请按 F21 键（CL 命令输入）。

Java 工具

- Java ajar 工具
- Java appletviewer 工具
  - 将 Java appletviewer 工具配合 Remote Abstract Window Toolkit 运行
- Java extcheck 工具
- Java idlj 工具
- Java jar 工具
- Java jarsigner 工具
- Java javac 工具
- Java javadoc 工具

Java ajar 工具

ajar 工具是用来创建和操纵“Java(TM) 压缩文档”（JAR）文件的 jar 工具的备用接口。可以使用 ajar 工具来操纵 JAR 文件和 ZIP 文件。

如果您需要 ZIP 接口或 UNZIP 接口，则请使用 ajar 工具，而不是使用 jar 工具。

与 jar 工具一样，ajar 可列示 JAR 文件的内容、从 JAR 文件中抽取内容，创建新的 JAR 文件，并支持许多 ZIP 格式。另外，ajar 工具还支持在现有 JAR 文件中添加和删除文件。

ajar 工具可以在 Qshell Interpreter 中使用。有关更多详细信息，参见 ajar — 备用 Java 压缩文档。

Java appletviewer 工具

appletviewer 工具允许您在不使用 Web 浏览器的情况下运行 applet。它与 Sun Microsystems 提供的 appletviewer 工具相兼容。

appletviewer 工具可以在 Qshell Interpreter 中使用。您需要使用 Remote Abstract Window Toolkit 才能运行 appletviewer 工具。有关如何设置 Remote AWT 来使用 appletviewer 工具的信息，参见将 Java appletviewer 工具配合 Remote AWT 运行。

有关 appletviewer 工具的更多信息，参见 Sun Microsystems 提供的 appletviewer 工具。

将 Java appletviewer 工具配合 Remote Abstract Window Toolkit 运行: 要使用 Java appletviewer 工具，必须在 Windows(R) 远程屏幕上安装 Remote Abstract Window Toolkit for Java，并使用 sun.applet.AppletViewer 类或在 Qshell Interpreter 中运行具有 Remote AWT 属性的 appletviewer 工具。

例如，如果使用 sun.applet.AppletViewer 类，并在 TicTacToe 目录之外运行 example1.html，则命令行应类似于:

```
JAVA CLASS(sun.applet.AppletViewer) PARM('example1.html') CLASSPATH('/TicTacToe')
PROP((RmtAwtServer '1.1.11.11') (os400.class.path.rawt 1)(java version 1.3))
```

如果在 Qshell Interpreter 中使用 appletviewer 工具，并在 TicTacToe 目录之外运行 example1.html，则命令应该类似于:

```
qsh "enter"
cd TicTacToe "enter"
Appletviewer -J-DRmtAwtServer=1.1.11.11 -J-Dos400.class.path.rawt=1 -J-Djava.version=1.3 example1.html
```

注意: -J 是 Appletviewer 的运行时标志。-D 是属性。

Java extcheck 工具

在 “Java 2 SDK（J2SDK）Standard Edition V1.2 和更高版本中，extcheck 工具能够检测目标 JAR 文件与当前安装的扩展 JAR 文件之间的版本冲突。它与 Sun Microsystems 提供的 keytool 相兼容。

extcheck 工具可以在 Qshell Interpreter 中使用。

有关 extcheck 工具的更多信息，参见 Sun Microsystems 的 extcheck 工具。

Java idlj 工具

» idlj 工具根据给定的 “接口定义语言”（IDL）文件生成 Java 绑定。idlj 工具也称为 “IDL 至 Java” 编译器。它与 Sun Microsystems 提供的 idlj 工具相兼容。此工具仅可用于 Java Development Kit 1.3 和 1.4。

有关 idlj 工具的更多信息，参见 Sun Microsystems 的 idlj 工具 «。

Java jar 工具

jar 工具将多个文件组合成单个 “Java 压缩文档”（JAR）文件。它与 Sun Microsystems 提供的 jar 工具相兼容。

jar 工具可以在 Qshell Interpreter 中使用。

有关 jar 工具的备用接口，参见用来创建和操纵 JAR 文件的 ajar 工具。

有关 iSeries 文件系统的更多信息，参见集成文件系统或集成文件系统中的文件。

有关 jar 工具的更多信息，参见 Sun Microsystems 的 jar 工具。

Java jarsigner 工具

在 Java 2 SDK（J2SDK）Standard Edition V1.2 和更高版本中，jarsigner 工具对 JAR 文件进行签名，并验证已签名的 JAR 文件上的签名。当 jarsigner 工具需要查找对 JAR 文件进行签名所需的专用密钥时，它便访问由 keytool 创建和管理的密钥库。在 J2SDK 中，jarsigner 和 keytool 工具替换了 javakey 工具。它与 Sun Microsystems 提供的 jarsigner 工具相兼容。

jarsigner 工具可以在 Qshell Interpreter 中使用。

有关 jarsigner 工具的更多信息，参见 Sun Microsystems 的 jarsigner 工具。

Java javac 工具

javac 工具编译 Java 程序。它与 Sun Microsystems 提供的 javac 工具相兼容，只有一点例外。

-classpath

不覆盖缺省类路径。而是，它被追加到系统缺省类路径。-classpath 选项覆盖 CLASSPATH 环境变量。

javac 工具可以在 Qshell Interpreter 中使用。

如果已在 iSeries 服务器上安装了 JDK 1.1.x 来作为缺省 JDK，但是您需要从版本 1.2 或更高版本中运行 java 命令，则请输入此命令:

```
javac -djava.version=1.2 <my_dir> MyProgram.java
```

有关 javac 工具的更多信息，参见 Sun Microsystems 的 javac 工具。

Java javadoc 工具

javadoc 工具生成 API 文档。它与 Sun Microsystems 提供的 javadoc 工具相兼容。

javadoc 工具可在 Qshell Interpreter 中使用。

有关 javadoc 工具的更多信息，参见 Sun Microsystems 的 javadoc 工具。

Java 工具

- Java javah 工具
- Java javakey 工具
- Java javap 工具
- Java keytool
- Java native2ascii 工具
- Java policytool
- Java rmic 工具
- Java rmid 工具
- Java rmiregistry 工具
- Java serialver 工具
- Java tnameserv 工具

Java javah 工具

javah 工具使实现 Java(TM) 本机方法更为方便。除少数例外情况外，它与 Sun Microsystems 提供的 javah 工具相兼容。

注意: 编写本机方法意味着您的应用程序不是 100% 的纯 Java 程序。这也意味着应用程序不能直接跨平台移植。本机方法在本质上是特定于平台或系统的。使用本机方法可能会增加应用程序的开发和维护成本。

javah 工具可以在 Qshell Interpreter 中使用。它在当前工作目录中读取 Java 类文件并创建 C 语言头文件。所写的头文件是“iSeries 流文件”（STMF）。在可以将它包括在 iSeries 服务器上的 C 程序中之前，必须将它复制到文件成员中。

javah 工具与 Sun Microsystems 提供的工具相兼容。然而，如果指定下面这些选项，则 iSeries 服务器将忽略它们。

-td iSeries 服务器上的 javah 工具不需要临时目录。

-stubs iSeries 服务器上的 Java 只支持“Java 本机接口”（JNI）格式的本机方法。存根仅对于早于 JNI 的本机方法格式才是必需的。

-trace
与 .c 存根文件输出相关，iSeries 服务器上的 Java 不支持该文件输出。

-v 不支持。

注意: 始终必须指定 -jni 选项。iSeries 服务器不支持 JNI 以前的本机方法实现。

有关 javah 工具的更多信息，请参见 Sun Microsystems 的 javah 工具。

Java javakey 工具

使用 javakey 工具来对密钥进行加密以及生成和管理证书，包括为 applet 生成数字签名。它与 Sun Microsystems 在 Java Development Kit（JDK）V1.1.x 中提供的 javakey 工具相兼容。

在 Java 2 Software Development Kit（J2SDK）Standard Edition V1.2 或更高版本中，javakey 工具已过时了。由于 JDK V1.1.x 中存在错误，所以在 J2SDK V1.2 或更高版本中认为使用 1.1.x javakey 工具签名的代码是未签名的。如果使用 J2SDK V1.2 或更高版本来对代码进行签名，则在 JDK 1.1.x 各版本上认为它是未签名的。

注意: iSeries 安全套接字层（SSL）支持不能访问此工具创建的密钥。而是，必须使用集成到 iSeries 服务器中并使用“数字证书管理器”创建或导入的证书和密钥容器。有关更多信息，参见通过安全套接字层来保证 Java 应用程序的安全。

Applet 封装和 applet 签名依赖于浏览器。请检查您的浏览器文档，以确保浏览器与 Java JAR 文件格式和 javakey applet 签名相兼容。

注意: 用 javakey 工具创建的文件包含敏感信息。适当的集成文件系统安全性措施可保护公用和专用密钥文件。

javakey 工具可以在 Qshell Interpreter 中使用。

有关 iSeries 文件系统的更多信息，参见集成文件系统或集成文件系统中的文件。

有关 javakey 工具的更多信息，参见 Sun Microsystems 的 javakey 工具  。

Java javap 工具

javap 工具对编译型 Java 文件进行反汇编，并打印出 Java 程序。当系统上不再有原始源代码时，这可能会很有帮助。

除少数例外情况外，它与 Sun Microsystems 提供的 javap 工具相兼容。

-b 此选项被忽略。因为 iSeries 服务器上的 Java 只支持 Java Development Kit（JDK）1.1.4 和更新版本，所以向后兼容性不是必需的。

-p 在 iSeries 服务器上，-p 不是有效选项。必须将 -private 拼写完整。

-verify 此选项被忽略。javap 工具在 iSeries 服务器上不执行验证。

javap 工具可以在 Qshell Interpreter 中使用。

注意: 使用 javap 工具对类进行反汇编可能会违反那些类的许可证协议。在使用 javap 工具之前，应查阅这些类的许可证协议。

有关 javap 工具的更多信息，参见 Sun Microsystems 的 javap 工具。

Java keytool

在 Java 2 SDK（J2SDK）Standard Edition V1.2 或更高版本中，keytool 创建公用密钥和专用密钥对以及自签名证书，并管理密钥库。在 J2SDK 中，jarsigner 和 keytool 工具替换了 javakey 工具。它与 Sun Microsystems 提供的 keytool 相兼容。

keytool 可以在 Qshell Interpreter 中使用。

有关 keytool 的更多信息，参见 Sun Microsystems 的 keytool。

Java native2ascii 工具

native2ascii 工具将带有本机编码字符（非拉丁 1 和非 Unicode 字符）的文件转换成带有 Unicode 编码字符的文件。它与 Sun Microsystems 提供的 native2ascii 工具相兼容。

native2ascii 工具可以在 Qshell Interpreter 中使用。

有关 native2ascii 工具的更多信息，参见 Sun Microsystems 的 native2ascii 工具。

Java policytool

在 Java 2 SDK Standard Edition 中，policytool 创建和更改外部策略配置文件，这些文件定义针对于您的安装的 Java 安全性策略。它与 Sun Microsystems 提供的 policytool 相兼容。

policytool 是一个图形用户界面（GUI）工具，可以在 Qshell Interpreter 和 Remote Abstract Window Toolkit 中使用。有关更多信息，参见 IBM Developer Kit for Java Remote Abstract Window Toolkit。

有关 policytool 的更多信息，参见 Sun Microsystems 的 policytool。

Java rmic 工具

rmic 工具为 Java 对象生成存根文件和类文件。它与 Sun Microsystems 提供的 rmic 工具相兼容。

rmic 工具可以在 Qshell Interpreter 中使用。

有关 rmic 工具的更多信息，参见 Sun Microsystems 的 rmic 工具。

Java rmid 工具

在 Java 2 SDK（J2SDK）Standard Edition 中，rmid 工具启动激活系统守护程序，因此可以在 Java 虚拟机中注册和激活对象。它与 Sun Microsystems 提供的 rmid 工具相兼容。

rmid 工具可在 Qshell Interpreter 中使用。

有关 rmid 工具的更多信息，参见 Sun Microsystems 的 rmid 工具。

Java rmiregistry 工具

rmiregistry 工具在指定端口上启动远程对象注册表。它与 Sun Microsystems 提供的 rmiregistry 工具相兼容。

rmiregistry 工具可在 Qshell Interpreter 中使用。

有关 rmiregistry 工具的更多信息，参见 Sun Microsystems 的 rmiregistry 工具。

Java serialver 工具

serialver 工具返回一个或多个类的版本号或序列化唯一标识符。它与 Sun Microsystems 提供的 serialver 工具相兼容。

serialver 工具可以在 Qshell Interpreter 中使用。

有关 serialver 工具的更多信息，参见 Sun Microsystems 的 serialver 工具。

Java tnameserv 工具

在 Java 2 SDK（J2SDK）Standard Edition V1.3 » 或更高版本 « 中，tnameserv（瞬时命名服务）工具提供对命名服务的访问。它与 Sun Microsystems 提供的 tnameserv 工具相兼容。

tnameserv 工具可以在 Qshell Interpreter 中使用。

Qshell 中的 Java 命令

Qshell 中的 java 命令运行 Java(TM) 程序。除少数例外情况外，它与 Sun Microsystems 提供的 java 工具相兼容。

IBM Developer Kit for Java 忽略 Qshell 中 java 命令的这些选项。

选项	描述
-cs	不支持此选项。
-checksource	不支持此选项。
-debug	iSeries 内部调试器支持此选项。
-noasyncgc	总是通过 IBM Developer Kit for Java 运行垃圾收集。
-noclassgc	总是通过 IBM Developer Kit for Java 运行垃圾收集。
-prof	iSeries 服务器有自己的性能工具。
-ss	此选项在 iSeries 服务器上不适用。
-oss	此选项在 iSeries 服务器上不适用。
-t	iSeries 服务器使用它自己的跟踪功能。
-verify	在 iSeries 服务器上总是进行验证。
-verifyremote	在 iSeries 服务器上总是进行验证。
-noverify	在 iSeries 服务器上总是进行验证。

在 iSeries 服务器上，-classpath 选项不覆盖缺省类路径。而是，它被追加到系统缺省类路径。-classpath 选项覆盖 CLASSPATH 环境变量。

Qshell 中的 java 命令支持 iSeries 服务器的新选项。这些是受支持的新选项。

选项	描述
» -chkpath «	此选项检查对 CLASSPATH 中目录的公共写访问权。
-opt	此选项指定优化级别。
» -showversion	此选项指定 JDK 版本。此选项对 JDK 1.3 和 1.4 存在。 «
» -verbose[:class|gc|jni] «	每次垃圾收集扫描时都显示一条消息。
» -Xrun[:]	显示一条消息，指示服务程序和 JVM 启动期间 JVM_OnLoad 函数的可选参数字符串。 «

CL 命令参考资料中的“运行 Java”（RUNJVA）命令详细描述了这些新选项。“创建 Java 程序”（CRTJVAPGM）命令、“删除 Java 程序”（DLTJVAPGM）命令和“显示 Java 程序”（DSPJVAPGM）命令的 CL 命令参考信息包含关于管理 Java 程序的信息。

Qshell 中的 java 工具可以在 Qshell Interpreter 中使用。

有关 Qshell 中的 java 命令的更多信息，参见 Sun Microsystems 的 java 工具。

Java 支持的 CL 命令

IBM Developer Kit for Java(TM) 支持下列 CL 命令。

- “分析 Java 虚拟机”（ANZJVM）命令检索信息并将该信息设置到 Java 虚拟机（JVM）中。此命令通过返回关于活动类的信息来帮助您调试 Java 程序。
- 更改 Java 程序（CHGJVAPGM）命令更改 Java 程序的属性。
- 创建 Java 程序（CRTJVAPGM）命令利用 Java 类文件、ZIP 文件或 JAR 文件来在 iSeries 服务器上创建 Java 程序。
- 删除 Java 程序（DLTJVAPGM）命令删除与 Java 类文件、ZIP 文件或 JAR 文件相关联的 iSeries Java 程序。
- 显示 Java 程序（DSPJVAPGM）命令显示关于 iSeries 上的 Java 程序的信息。
- 转储 Java 虚拟机（DMPJVM）命令将关于指定作业的 Java 虚拟机的信息转储至假脱机打印机文件。
- JAVA 命令和运行 Java（RUNJVA）命令运行 iSeries Java 程序。

有关更多信息，参见程序和 CL 命令 API。

分析 Java 虚拟机（ANZJVM）命令

“分析 Java(TM) 虚拟机”（ANZJVM）命令检索信息并将该信息设置到 Java 虚拟机（JVM）中。其意图在于通过返回关于活动类的信息来帮助您调试 Java 程序。

在运行 ANZJVM 命令时，有一个参数可指定是否应强制垃圾收集周期，如果是的话，将在每次传送之前尝试强制垃圾收集周期。如果尚未运行正在分析的 JVM 的垃圾收集周期，则不能强制周期。还存在用于指定应如何存储信息以及两次传送之间的时间间隔应该有多长的参数。

在 ANZJVM 命令完成之后，将生成假脱机输出文件。有关更多信息，请参考示例: ANZJVM 命令和输出文件。

有关更多信息，请参考 ANZJVM 命令和 ANZJVM 语法图。

有关更多信息，请参考 ANZJVM 命令的注意事项。

运行 ANZJVM 命令

您可以运行“分析 Java(TM) 虚拟机”（ANZJVM）命令来为指定的作业收集关于 Java 虚拟机（JVM）的信息。通过创建 JVM 的副本并将数据与以后创建的另一副本作比较，可以分析数据以帮助查找对象泄漏。使用时间间隔参数来指定两次堆传送之间的时间。当时间间隔设置为零时，将执行两次堆传送，其中第二次传送在第一次传送完成后立即开始。然后将关于两次传送的信息返回。

对堆中的每个类返回以下信息:

1. 类名称
2. 垃圾收集堆信息
   a. 第一次传送
   b. 第二次传送
   c. 垃圾收集堆中对象数方面的更改
3. 使用的对象空间
   a. 第一次传送
   b. 第二次传送
   c. 对象大小方面的更改

4. 全局注册表信息，这与对对象表列示的全局注册表的信息相同。
5. 装入程序名称

强制垃圾收集周期: 为了获取更清晰的堆视图，最好在垃圾收集周期之后尽快地进行查看。ANZJVM 有一个 FRCGC 参数，用于指定是否应强制进行垃圾收集。可能的选项如下:

- *YES
  在每次对堆进行 ANZJVM 清理之前强制进行垃圾收集。
- *NO
  ANZJVM 不强制进行垃圾收集。 «

ANZJVM 命令的注意事项

» 由于 ANZJVM 的运行时间可能很长，所以，JVM 很有可能在 ANZJVM 能够完成之前结束。如果 JVM 结束，则 ANZJVM 返回 JVAB606 消息（即，正在处理 ANZJVM 时 JVM 结束）以及它能够获取的数据。

JVM 可以处理的类的数目也没有上限。如果存在的类比能够处理的要多，则 ANZJVM 应返回可以处理的数据以及一条消息，让您知道还有附加的信息没有报告。当数据需要截断时，ANZJVM 将返回尽可能多的信息。

时间间隔参数的长度限制为 3600 秒（一个小时）。ANZJVM 可以返回其信息的类的数目受系统上的存储器量的限制。 «

示例: ANZJVM 命令

» 以下 ANZJVM 命令对用户名 JOHN 的名为 QJVACMDSRC 的作业（作业号为 099112）收集 JVM 的两个副本，收集间隔为 60 秒。将把来自副本的数据置于 QSYSPRT 打印机设备文件中。

```
ANZJVM JOB(099112/JOHN/QJVACMDSRV)
```

假脱机输出文件的示例如下。

ANZJVM 命令的假脱机输出文件: 以下是运行 ANZJVM 命令后包含在假脱机输出文件中的数据的示例:

```
Mon Feb 26 15:39:12 CST 2002
Job: 099112/JOHN/QJVACMDSRV
Interval: 10 seconds
Total garbage collection cycles prior to running: 29
Total garbage collection cycles after running: 31
GC forced: NO
.....................................................................
. Class loader information                                          .
.....................................................................
0 Default class loader
.....................................................................
. GC heap information                                               .
.....................................................................
Loader
|   Number of pass one objects in the GC heap
|   |          Number of pass two objects in the GC heap
|   |          |          Change in the number of objects in the GC heap
|   |          |          |          Pass one object size (K)
|   |          |          |          |        Pass two object size (K)
|   |          |          |          |        |        Change in object size (K)
|   |          |          |          |        |        |        In global registry
|   |          |          |          |        |        |        |   Class name
0   431359     491363     60004      18979    21619    2640     NO  java/lang/
                                                                     String
0   8          8          0          0        0        0        NO  sun/misc/
                                                                     URLClass
                                                                     Path$
                                                                     JarLoader
```

```
0   4         4         0         0       0       0       NO  java/lang/
                                                              Object
0   7         7         0         0       0       0       NO  java/util/
                                                              zip/
                                                              Inflater
0   2         2         0         0       0       0       NO  java/lang/
                                                              Thread
0   2         2         0         0       0       0       NO  [Ljava/lang/
                                                              Class;
0   13        13        0         0       0       0       NO  java/io/
File                                                                Descriptor
0   2         2         0         0       0       0       NO  java/io/
                                                              Buffered
                                                              Writer
0   4         4         0         0       0       0       NO  [Ljava/lang/
                                                              String;
0   1         1         0         0       0       0       NO  [Ljava/io/
                                                              ObjectStream
                                                              Class$
                                                              ObjectStream
                                                              ClassEntry;
0   404       404       0         24      24      0       NO  [Ljava/util/
                                                              HashMap$
                                                               Entry;
0   423       423       0         15      15      0       NO  java/util/
                                                              jar/
                                                              Attributes$
                                                              Name
0   1         1         0         0       0       0       NO  java/io/
                                                              Buffered
                                                              InputStream
0   2         2         0         0       0       0       NO  java/io/
                                                              Buffered
                                                              Output
                                                              Stream
0   1         1         0         0       0       0       NO  java/security/
                                                              Protection
                                                              Domain
0   1         1         0         0       0       0       NO  sun/security/
                                                              provider/Sun
0   1         1         0         0       0       0       NO  java/io/
File                                                                Permission
0   128       128       0         6       6       0       NO  java/util/
                                                               Hashtable$
                                                              Entry
0   2         2         0         0       0       0       NO  java/net/
                                                              URLClass
                                                              Loader$
                                                              ClassFinder
0   1         1         0         0       0       0       NO  java/lang/
                                                              Runtime
0   1         1         0         0       0       0       NO  java/util/
                                                              BitSet
0   7         7         0         0       0       0       NO  java/util/
                                                              jar/
                                                              JarVerifier
0   2         2         0         0       0       0       NO  java/lang/
                                                              ThreadGroup
0   22        22        0         1       1       0       NO  java/util/
                                                              Locale
0   8         8         0         0       0       0       YES java/io/
                                                              RandomAccess
File0   37        37        0         125     125     0       YES [B
0   1         1         0         0       0       0       NO  sun/misc/
                                                              Launcher
0   871       871       0         117     117     0       YES [C
0   1         1         0         0       0       0       NO  sun/misc/
```

```
                                                                Launcher$
                                                                Factory
0   435       435       0         10        10        0       YES java/lang/
                                                                Class
0   1         1         0         0         0         0       NO  java/util/
                                                                Collections$
                                                                EmptyList
0   1         1         0         0         0         0       NO  java/util/
                                                                Collections$
                                                                EmptyMap
0   1         1         0         0         0         0       NO  java/lang/
                                                                String$
                                                                Case
                                                                Insensitive
                                                                Comparator
0   1         1         0         2         2         0       NO  [I
0   3         3         0         0         0         0       NO  java/lang/
                                                                OutOf
                                                                Memory
                                                                Error
0   1         1         0         0         0         0       NO  [J
0   800       800       0         41        41        0       NO  java/util/
                                                                HashMap$
                                                                Entry
0   1         1         0         0         0         0       NO  java/util/
                                                                Random
0   5         5         0         0         0         0       NO  java/security/
                                                                Access
                                                                Control
                                                                Context
0   1         1         0         0         0         0       YES java/lang/
                                                                ref/
                                                                Reference$
                                                                Reference
                                                                Handler
0   1         1         0         8         8         0       NO  [S
0   1         1         0         0         0         0       NO  [Ljava/lang/
                                                                ref/Soft
                                                                Reference;
0   1         1         0         0         0         0       NO  java/io/
                                                                Object
                                                                Stream
                                                                Class$
                                                                Compare
                                                                Member
                                                                ByName
0   7         7         0         0         0         0       NO  java/util/
                                                                jar/
                                                                JarFile$
                                                                JarFileEntry
0   1         1         0         0         0         0       NO  java/util/
                                                                Collections$
                                                                EmptySet
0   1         1         0         0         0         0       NO  [Ljava/
                                                                security/
                                                                cert/
                                                                Certificate;
0   1         1         0         0         0         0       NO  java/lang/
                                                                ref/
                                                                Reference
                                                                Queue
0   1         1         0         0         0         0       NO  java/util/
                                                                 Hashtable$
                                                                Empty
                                                                Enumerator
0   1         1         0         0         0         0       YES sun/misc/
                                                                Launcher$
```

```
                                                                AppClass
                                                                Loader
0   1         1         0         0       0       0       NO  java/lang/
                                                                Shutdown$
                                                                Lock
0   17        17        0         0       0       0       NO  java/util/
                                                                Vector
0   3         3         0         0       0       0       NO  java/util/
                                                                Stack
0   17        17        0         1       1       0       NO  java/net/URL
0   21        21        0         1       1       0       NO  java/lang/
                                                                ref/
                                                                Finalizer
0   1         1         0         0       0       0       NO  java/io/
                                                                Os400
                                                                FileSystem
0   1         1         0         0       0       0       NO  java/lang/
                                                                Runtime
                                                                Permission
0   1         1         0         0       0       0       NO  [Ljava/io/
                                                                File;
0   2         2         0         0       0       0       NO  sun/io/
                                                                CharToByte
ISO8859_10  1         1         0         0       0       0       NO  sun/misc/
                                                                Launcher$
                                                                ExtClass
                                                                Loader
0   1         1         0         0       0       0       NO  java/lang/
                                                                ref/
                                                                Soft
                                                                Reference
0   1         1         0         0       0       0       NO  sun/security/
                                                                provider/
                                                                PolicyFile
0   1         1         0         0       0       0       NO  java/io/
                                                                Object
                                                                Stream
                                                                Class$
                                                                Compare
                                                                Class
                                                                ByName
0   1         1         0         0       0       0       NO  sun/net/www/
                                                                protocol/
                                                                file/
                                                                Handler
0   8         8         0         0       0       0       NO  java/util/
                                                                jar/
                                                                JarFile
0   30        30        0         2       2       0       NO  java/util/
                                                                Hashtable
0   1         1         0         0       0       0       NO  java/lang/
                                                                ref/
                                                                Reference$
                                                                Lock
0   2         2         0         0       0       0       NO  java/io/
                                                                PrintStream
0   1         1         0         0       0       0       NO  java/util/
                                                                 Hashtable$
                                                                Empty
                                                                Iterator
0   4         4         0         0       0       0       NO  java/io/File
0   391       391       0         12      12      0       NO  java/util/
                                                                jar/
                                                                Attributes
0   2         2         0         0       0       0       NO  sun/misc/
                                                                URLClassPath
0   2         2         0         0       0       0       NO  java/io/
```

```
                                                                FileInput
                                                                Stream
0   2         2         0         0         0         0       NO  java/io/
                                                                Output
                                                                Stream
                                                                Writer
0   11        11        0         0         0         0       NO  java/util/
                                                                ArrayList
0   1         1         0         0         0         0       NO  java/net/
                                                                Unknown
                                                                Content
                                                                Handler
0   3         3         0         0         0         0       NO  java/lang/
                                                                ref/
                                                                Reference
                                                                Queue$Lock
0   2         2         0         0         0         0       NO  java/io/
                                                                FileOutput
                                                                Stream
0   1         1         0         0         0         0       NO  sun/misc/
                                                                URLClass
                                                                Path$
                                                                FileLoader
0   31        31        0         5246      5246      0       NO  [Ljava/lang/
                                                                Object;
0   1         1         0         0         0         0       NO  java/lang/
                                                                Class
                                                                Loader$
                                                                Native
                                                                Library
0   2         2         0         0         0         0       NO  [Ljava/lang/
                                                                Thread;
0   404       404       0         35        35        0       NO  java/util/
                                                                HashMap
0   2         2         0         0         0         0       NO  java/lang/
                                                                Boolean
0   1         1         0         0         0         0       YES java/lang/ref/
                                                                Finalizer$
                                                                Finalizer
                                                                Thread
0   1         1         0         0         0         0       NO  sun/security/
                                                                provider/
                                                                Policy
                                                                Permissions
0   7         7         0         0         0         0       NO  java/util/jar/
                                                                Manifest
0   2         2         0         0         0         0       NO  sun/net/www/
                                                                protocol/
                                                                jar/
                                                                Handler
0   1         1         0         0         0         0       NO  com/sun/
                                                                rsajca/
                                                                Provider
0   1         1         0         0         0         0       NO  java/util/
                                                                Collections
                                                                $Reverse
                                                                Comparator
0   2         2         0         0         0         0       NO  [Ljava/io/
                                                                ObjectStream
                                                                Field;
0   1         1         0         0         0         0       NO  java/security/
                                                                CodeSource
0   34        34        0         5         5         0       NO  [Ljava/util/
                                                                 Hashtable$
                                                                 Entry;
0   2         2         0         0         0         0       NO  java/lang/ref/
                                                                Reference
```

```
                                                                Queue$Null
0   2         2         0         0       0       0       NO  java/util/
                                                                Properties
0   2         2         0         0       0       0       NO  java/util/
                                                                HashSet
0   1         1         0         0       0       0       NO  [Ljava/lang/
                                                                ThreadGroup;
0   1         1         0         0       0       0       NO  java/util/
                                                                HashMap$
                                                                EmptyHash
                                                                Iterator
0   7         7         0         0       0       0       NO  java/io/
                                                                ByteArray
                                                                OutputStream
.....................................................................
. Global registry information                                       .
.....................................................................
Loader
|   Number of pass one objects in the GC heap
|   |         Number of pass two objects in the GC heap
|   |         |         Change in the number of objects in the GC heap
|   |         |         |         Pass one object size (K)
|   |         |         |         |       Pass two object size (K)
|   |         |         |         |       |       Change in object size (K)
|   |         |         |         |       |       |       Class name
0   8         8         0         0       0       0       java/io/
                                                            RandomAccessFile
0   8         8         0         64      64      0       [B
0   10        10        0         1       1       0       [C
0   439       439       0         10      10      0       java/lang/Class
0   1         1         0         0       0       0       java/lang/ref/
                                                            Reference$
                                                            ReferenceHandler
0   1         1         0         0       0       0       sun/misc/
                                                            Launcher$
                                                            AppClassLoader
0   1         1         0         0       0       0       java/lang/ref/
                                                            Finalizer$
                                                            FinalizerThread
```

示例: 更改 Java 程序（CHGJVAPGM）命令

要更改与类文件 myJavaClassFileName 相关联的 Java(TM) 程序，请使用“更改 Java 程序”（CHGJVAPGM）命令。将对 Java 程序类文件字节码进行解释。要启动程序，请使用“运行 Java”（RUNJVA）命令。仅当指定的属性与当前程序的属性不同时，才会重建 Java 程序。

示例 1: 更改解释型 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
CHGJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class')
   OPTIMIZE(*INTERPRET)
```

以下示例与“示例 1”相同，只是对程序进行了优化。由于用 OPTIMIZE(10) 进行了更改，因此该程序包含经编译的机器指令，这些指令在 Java 程序启动时运行。

示例 2: 更改 JAR 文件中经优化的 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
CHGJVAPGM CLSF('/projectB/myJavaappfile.jar')
OPTIMIZE(10)
```

有关语法图和参数详细信息，参见更改 Java 程序（CHGJVAPGM）命令。

许可内码选项参数字符串

下表显示了“许可内码选项”（LICOPT）参数所识别的字符串。这些字符串不区分大小写，为了便于阅读，它们以混合大小写形式显示。

LICOPT 参数字符串

String	描述
AllFieldsVolatile	如果设置的话，则将所有字段看作是易失的。
NoAllFieldsVolatile	如果设置的话，则不将任何字段看作是易失的。
AllowBindingToLoadedClasses	指示作为正在运行的 Java(TM) 虚拟机中的 defineClass 调用的结果而创建的临时类表示法可能与同一 Java 虚拟机中的其它类表示法紧密绑定。
NoAllowBindingToLoadedClasses	指示作为正在运行的 Java 虚拟机中的 defineClass 调用的结果而创建的临时类表示法可能未与同一 Java 虚拟机中的其他类表示法紧密绑定。
AllowClassCloning	当为 JAR 文件生成多个 Java 程序时，允许将来自一个程序的类的副本包括在为另一个程序生成的代码中。这有助于强烈内联。
NoAllowClassCloning	不允许将来自一个程序的类的副本包括在为另一个程序生成的代码中。
AllowInterJarBinding	允许紧密绑定至位于正在编译的类或 JAR 文件外部的类。这有助于强烈优化。
NoAllowInterJarBinding	不允许紧密绑定至位于正在编译的类或 JAR 文件外部的类。这将覆盖 CRTJVAPGM 上的 CLASSPATH 和 JDKVER 参数的存在情况。
» AllowMultiThreadedCreate	指示 CRTJVAPGM 象平常一样执行，只使用一个线程。 «
» NoAllowMultiThreadedCreate	CRTJVAPGM 在创建期间使用多个线程（如果它们可用的话）。 «
AnalyzeObjectLifetimes	使用可视类执行分析，以确定哪些对象具有较短的寿命。具有较短寿命的对象不会存在于分配该对象的方法之外，并且可能服从更强烈的优化。
NoAnalyzeObjectLifetimes	不执行短寿命对象的分析。
AllowBindingWithinJar	指示 ZIP 文件或 JAR 文件中的类表示法可能与同一 ZIP 文件或 JAR 文件中的其它类表示法紧密绑定。
NoAllowBindingWithinJar	指示 ZIP 文件或 JAR 文件中的类表示法可能未与同一 ZIP 文件或 JAR 文件中的其它类表示法紧密绑定。
AllowInlining	告知转换程序允许内联本地方法。对于优化级别 30 和 40，这是缺省值。
NoAllowInlining	不告知转换程序允许内联本地方法。
AssumeUnknownFieldsNonvolatile	当不能确定外部类中的字段的属性时，此参数通过假定该字段不是易失的来生成代码。

String	描述
NoAssumeUnknownFieldsNonvolatile	当不能确定外部类中的字段的属性时，此参数通过假定该字段是易失的来生成代码。
BindErrorHandling	指定在下列情况下应该执行哪些操作：作为指定 AssumeUnknownFieldsNonvolatile、PreresolveExtRef 或 PreLoadExtRef “许可内码” 选项的结果，Java 虚拟机类装入程序检测到类表示法中包含方法表示法，而在当前上下文中不能使用方法表示法。
BindInit	对本地初始化方法使用绑定调用。
NoBindInit	不对本地初始化方法使用绑定调用。
BindSpecial	对本地特殊方法使用绑定调用。
NoBindSpecial	不对本地特殊方法使用绑定调用。
BindStatic	对本地静态方法使用绑定调用。
NoBindStatic	不对本地静态方法使用绑定调用。
BindTrivialFields	在程序创建期间绑定 trivial 字段引用。
NoBindTrivialFields	在首次访问时解析字段引用。
BindVirtual	对本地最终虚拟方法使用绑定调用。
NoBindVirtual	不对本地最终虚拟方法使用绑定调用。
DeferResolveOnClass	使用假定为类名的字符串参数（例如，java.lang.Integer）。将 PreresolveExtRef 设置为优化级别 40 时，不会对用 DeferResolveOnClass 指定的类进行预解析操作。如果代码中未使用的路径中的某些类不在 CLASSPATH 中，则这很有用。它使您能够通过对每个丢失的类指定 “DeferResolveOnClass=’somepath.someclass’” 来使用优化级别 40，而不考虑上述情况。允许有多个 DeferResolveOnClass 项。
DevirtualizeFinalJDK	允许 CRTJVAPGM 利用标准 JDK 知识来实现对那些被认为是最终方法或最终类成员的 JDK 方法的调用。对于优化级别 30 和 40，这是缺省值。
NoDevirtualizeFinalJDK	不允许 CRTJVAPGM 利用标准 JDK 知识来实现对那些被认为是最终方法或最终类成员的 JDK 方法的调用。
DevirtualizeRecursive	导致在一些递归方法中生成特殊代码，并消除递归方法调用所产生的大部分开销。但是，对于递归方法的初始入口将生成附加的检查逻辑，因此，在浅递归的情况下，性能可能也不会提高。
NoDevirtualizeRecursive	不会导致在一些递归方法中生成特殊代码。
DisableIntCse	该项将导致：当为特定类型的整数表达式生成代码时，某些公共子表达式的优化被禁用。这可以通过对 “优化转换程序” 提供其他优化机会来提高整体优化水平。
NoDisableIntCse	该项将导致：当为某些类型的整数表达式生成代码时，某些公共子表达式的优化不被禁用。这通常会导致在较低的优化级别上反而可更好地执行代码。
DoExtBlockCSE	执行扩展基本块公共子表达式消除。
NoDoExtBlockCSE	不执行扩展基本块公共子表达式消除。
DoLocalCSE	执行本地公共子表达式消除。
NoDoLocalCSE	不执行本地公共子表达式消除。

String	描述
EnableCseForCastCheck	如果设置的话，则生成强制类型转换检查代码，可对该代码执行 DAG 至较早实例。
NoEnableCseForCastCheck	不设置；不生成可被执行 DAG 至较早实例的强制类型转换检查代码。
ErrorReporting	运行时错误报告字段 **: 提供当遇到验证或类格式错误时使编译失败的选项。 0 = 立即报告所有错误； 1 = 延迟报告字节码验证错误； 2 = 将字节码验证错误和类格式错误的报告延迟到运行时进行。
» HideInternalMethods	导致使克隆的类中的方法成为内部方法，从而允许在不存在对它们的引用或所有引用都是内联的时省略它们。对于优化级别 40，缺省值是 HideInternalMethods，对于 0 与 30 之间的优化级别，是 NoHideInternalMethods。 «
InlineArrayCopy	导致在一些标量数组实例中内联 System.arraycopy 方法。
NoInlineArrayCopy	防止内联 System.arraycopy 方法。
InlineInit	内联 java.lang 类的初始化方法。
NoInlineInit	不内联初始化方法。
InlineMiscFloat	从 java.lang.Math 内联其它浮点 / 双精度方法。
NoInlineMiscFloat	不内联其它浮点 / 双精度方法。
InlineMiscInt	从 java.lang.Math 内联其他整型 / 长整型方法。
NoInlineMiscInt	不内联其他整型 / 长整型方法。
InlineStringMethods	允许从 java/lang/String 内联特定方法。
NoInlineStringMethods	禁止从 java/lang/String 内联特定方法。
InlineTransFloat	从 java.lang.Math 内联超越浮点 / 双精度方法。
NoInlineTransFloat	不内联超越浮点 / 双精度方法。
OptimizeJsr	对具有单一目标的 “jsr” 字节码生成更好的代码。
NoOptimizeJsr	不对具有单一目标的 “jsr” 字节码生成更好的代码。
PreloadExtRef	指示在方法入口处可能预装入（没有对类进行初始化）了引用的类。
NoPreloadExtRef	指示在方法入口处可能没有预装入引用的类。但是，PreresolveExtRef 参数将覆盖此设置，并导致预装入和初始化所引用的类。
PreresolveExtRef	在方法入口处预解析所引用的方法。
NoPreresolveExtRef	在首次访问时解析方法引用。用来解析在其它机器上运行的程序中出现的 “找不到类” 异常。
ProgramSizeFactor	当 JAR 文件足够大，从而可能需要多个 Java 程序时，这个数值（缺省值为 100）用来确定每个程序可以增大至多大。
ShortCktAthrow	如果设置的话，则尝试将 athrows 短路。
NoShortCktAthrow	不设置，不尝试将 athrows 短路。
ShortCktExSubclasses	如果设置的话，则识别 “异常” 的某些子类并直接将它们短路。
NoShortCktExSubclasses	如果不设置的话，不识别 “异常” 的某些子类并直接将它们短路。

String	描述
StrictFloat	禁止与 Java 规范不严格相符的浮点优化。
NoStrictFloat	允许与 Java 规范不严格相符的浮点优化。

双星号（**）指示这些字符串需要用以下语法形式输入数值: stringname=number（中间没有空格）。

示例: 创建 Java 程序（CRTJVAPGM）命令

要创建 Java(TM) 程序并将其与类文件 myJavaClassFileName 相关联，请使用“创建 Java 程序”（CRTJVAPGM）命令。当使用 OPTIMIZE(*INTERPRET) 来创建 Java 程序时，会解释 Java 程序类文件字节码。要启动程序，请使用“运行 Java”（RUNJVA）命令。

示例 1: 创建解释型 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
CRTJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class')
OPTIMIZE(*INTERPRET)
```

此示例与“示例 1”相同，只是对程序进行了优化。由于是用 OPTIMIZE(40) 创建的，因此程序包含经编译的机器指令，这些指令在 Java 程序启动时运行。

示例 2: 创建经优化的 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
CRTJVAPGM CLSF('/projectB/team2/myJavaclassfile.class')
OPTIMIZE(40)
```

有关语法图和参数详细信息，参见创建 Java 程序（CRTJVAPGM）命令。

示例: 删除 Java 程序（DLTJVAPGM）命令

“删除 Java(TM) 程序”（DLTJVAPGM）命令删除与指定的类文件（名为 myJavaClassFileName）相关联的 Java 程序。

注意: DLTJVAPGM 命令不删除类文件或 ZIP 文件。

示例 1: 删除 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
DLTJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class')
```

有关语法图和参数详细信息，参见删除 Java 程序（DLTJVAPGM）命令。

示例: 转储 Java 虚拟机（DMPJVM）命令

“转储 Java(TM) 虚拟机”（DMPJVM）命令为指定作业转储关于 Java 虚拟机的信息。

示例 1: 转储 Java 虚拟机

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
DMPJVM JOB(099246/FRED/QJVACMDSRV)
```

DMPJVM 命令转储正在名为 099246/FRED/QJVACMDSRV 的作业中运行的 Java 虚拟机的信息。

示例输出:

```
JAVA VIRTUAL MACHINE INFORMATION: 099246/FRED/QJVACMDSRV
 ..................................................................
 . Classpath
 ..................................................................
/QIBM/ProdData/Java400/jdk117/lib/jdkptf117.zip:/QIBM/ProdData/Java400/jdk1
17/lib/classes.zip:/QIBM/ProdData/Java400/ext/IBMmisc.jar:/QIBM/ProdData/Ja
va400/ext/db2_classes.jar:/QIBM/ProdData/Java400/ext/jssl.jar:/QIBM/ProdDat
a/Java400/ext/ibmjssl.jar:/QIBM/ProdData/Java400/:/home/fred
 ..................................................................
 . Garbage collection
 ..................................................................
Garbage collector parameters
   Initial size: 2048 K
   Max size: *NOMAX
Current values
   Heap size: 9476 K
   Garbage collections: 0
 ..................................................................
 . Thread information
 ..................................................................
Information for 3 thread(s) of 3 thread(s) processed
Thread: 00000001 Thread-0
  TDE: B000200002941000
  Thread priority: 5
  Thread status: Destroy wait
  Thread group: main
  Runnable: java/lang/Thread
 Stack:
   None
 Locks:
   None
 ..................................................................
Thread: 00000003 t2
  TDE:  B000100005B37000
  Thread priority: 5
  Thread status: Timed wait
  Thread group: main
  Runnable: dbgtest2
 Stack:
    java/io/BufferedInputStream.read()I+11 (BufferedInputStream.java:154)
    pressEnter.theFirstMethod(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V+1
        0 (dbgtest2.java:15)
    dbgtest2.run()V+69 (dbgtest2.java:44)
    java/lang/Thread.run()V+11 (Thread.java:466)
 Locks:
   None
 ..................................................................
Thread: 00000002 t1
  TDE:  B000100005B33000
  Thread priority: 5
  Thread status: Java wait
  Thread group: main
  Runnable: dbgtest2
 Stack:
    pressEnter.theFirstMethod(Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;
    Ljava/lang/String;)V+0 (dbgtest2.java:14)
    dbgtest2.run()V+69 (dbgtest2.java:44)
```

```
   java/lang/Thread.run()V+11 (Thread.java:466)
Locks:
  None
.....................................................................
```

示例: 显示 Java 程序（DSPJVAPGM）命令

DSPJVAPGM（显示 Java 程序）命令显示与所指定的名为 myJavaClassFileName 的类文件相关联的 Java 程序。

示例 1: 显示 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
DSPJVAPGM CLSF('/projectA/team2/myJavaClassFileName.class') OUTPUT(*)
```

有关语法图和参数详细信息，参见显示 Java 程序（DSPJVAPGM）命令。

JAVA 命令

JAVA 命令的功能与 “运行 Java(TM)” （RUNJVA）命令完全相同。可以将它们互换使用。有关信息及可以配合 JAVA 命令使用的参数，参见运行 Java（RUNJVA）命令。

示例: 使用运行 Java（RUNJVA）命令

“运行 Java(TM)” （RUNJVA）命令运行与类相关联的 iSeries Java 程序。

示例 1: 运行 Java 程序

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
RUNJVA CLASS ('/projectA/myJavaclassname')
```

有关语法图和参数详细信息，参见运行 Java（RUNJVA）命令。

Java 支持的 iSeries 导航器命令

“iSeries 导航器” 是用于 Windows(R) 桌面的一个图形界面。它是 iSeries Access for Windows 的一部分，包括了管理员或用户完成其日常工作所需的许多 iSeries 功能。

“iSeries 导航器” 支持将 Java(TM) 作为包含在 iSeries Access for Windows 的 “文件系统” 选项中的插件。要使用 “iSeries 导航器” Java 插件，需要在 iSeries 服务器上安装 IBM Developer Kit for Java。然后，要在个人计算机上安装 Java 插件，请通过 Client Access 文件夹中的 “选择性安装” 选择 “文件系统”。

类文件、JAR 文件、ZIP 文件和 Java 文件都驻留在集成文件系统中。 “iSeries 导航器” 允许您在右窗格中查看这些文件。请对要使用的类文件、JAR 文件、ZIP 文件或 Java 文件进行右键单击。这将显示上下文菜单。

» 从上下文菜单中选择**相关联的 Java 程序 —> 新建...** 将启动 Java 转换程序，此程序可创建与类文件、JAR 文件或 ZIP 文件相关联的 iSeries Java 程序。 « 将有一个对话框允许您指定有关如何创建程序的详细信息。可以为 Java 变换或 Java 解释创建程序。

注意: 如果选择变换，则类文件中的字节码将变换为 RISC 指令，此时的性能将优于使用解释时的性能。

» 从上下文菜单中选择**相关联的 Java 程序 —> 编辑...** 将更改与 Java 类文件、ZIP 文件或 JAR 文件相连接的 Java 程序的属性。 «

» 从上下文菜单中选择**相关联的 Java 程序 —> 运行...** 将在 iSeries 服务器上运行类文件。 « 还可以选择 JAR 或 ZIP 文件并运行位于该 JAR 或 ZIP 文件中的类文件。将出现一个对话框，它允许您指定有关如何运行该程序的详细信息。 » 如果已选择了**相关联的 Java 程序 —> 新建...**，则运行程序时将使用与类文件相关联的 iSeries Java 程序。 « 如果没有与类文件相关联的 iSeries Java 程序，则在运行程序之前创建 iSeries Java 程序。

» 从上下文菜单中选择**相关联的 Java 程序 —> 删除...** 将删除与类文件、JAR 文件或 ZIP 文件相关联的 iSeries Java 程序。 «

从上下文菜单中选择**属性**将显示一个属性对话框，该对话框包含 **Java 程序**和 **Java 选项**选项卡。这些选项卡允许您查看有关如何为类文件、JAR 文件或 ZIP 文件创建了关联 iSeries Java 程序的详细信息。

注意: 这些面板是“显示 Java 程序”信息。

从上下文菜单中选择**编译 Java 文件**将把所选的任何 Java 文件转换成它们的类文件字节码。

» 请查看“iSeries 导航器”附带包括的帮助信息以了解**新建 Java 程序**、**编辑 Java 程序**、**运行 Java 程序**、**Java 程序**、**Java 选项**、**编译 Java 文件**和**删除 Java 程序**这些“iSeries 导航器”对话框的参数和选项。 «

第 7 章 可选包

可选包定义了用于扩展核心 Java 平台 API 的“应用程序编程接口”（API）。以下是可以与 IBM Developer Kit for Java(TM) 配合使用的可选包:

Java 认证和授权服务
“Java 认证和授权服务”（JAAS）允许将特定用户或身份与当前 Java 线程相关联。

Java 密码术扩展
“Java 密码术扩展”（JCE）提供了用于加密、密钥生成和密钥协议以及“消息认证代码”（MAC）算法的框架和实现。JCE 还支持安全流和密封对象。

Java 命名和目录接口
“Java 命名和目录接口”（JNDI）是 JavaSoft 的平台应用程序接口（API）的一部分。借助 JNDI，您可以连接至多个命名和目录服务。您可以使用此接口来构建功能强大且可移植的目录启用 Java 应用程序。

Java 安全套接字层
“Java 安全套接字层”（JSSL）是一组启用安全因特网通信的 Java 包。它实现了 SSL 和“传输层安全性”（TLS）协议的 Java 版本，并包括有关数据加密、服务器认证、消息完整性和可选客户机认证的功能。

JavaMail
JavaMail API 提供了一组模仿电子（电子邮件）系统的抽象类。此 API 提供了独立于平台且独立于协议的框架，可用来构建基于 Java 的电子邮件和消息传递应用程序。

Java 打印服务
“Java 打印服务”API 允许在所有 Java 平台上打印。Java 1.4 提供了一个框架，在这个框架中，Java 运行时环境和第三方可以提供流生成器插件，用于生成各种可打印格式，如 PDF、Postscript 和“高级功能显示”(TM)（AFP(TM)）。

Java 命名和目录接口

“Java(TM) 命名和目录接口”（JNDI）是 JavaSoft 的平台应用程序接口（API）的一部分。借助 JNDI，您就可以无缝地连接至多个命名和目录服务。您可以使用此接口来构建功能强大且可移植的目录启用 Java 应用程序。

JavaSoft 与领导业界的伙伴一起开发出 JNDI 规范，这些伙伴包括 IBM、SunSoft、Novell、Netscape 和 Hewlett-Packard 公司。

有关 JNDI 的更多信息，参见 Sun Microsystems 的 Java 命名和目录接口  。有关特定于 IBM 的信息，参见 IBM JNDI LDAP 提供程序编程指南。

IBM JNDI LDAP 提供程序编程指南

本编程指南假定您熟悉“Java(TM) 命名和目录接口”（JNDI）以及“轻量级目录访问协议”（LDAP）的操作方式。有关更多信息，参见 Sun Microsystems 的 JNDI 文档  。

IBM 提供了 JNDI 的可以与 SDK 或“Java 运行时环境”（JRE）1.2.2 配合使用的 LDAP 服务提供程序。不支持将 IBM JNDI LDAP 提供程序与 SDK 或 JRE 1.3 或更新版本配合使用；而是，应使用 JNDI 和 Sun Microsystems 的 JNDI LDAP 提供程序，它们是 SDK 和 JRE 1.3 的一部分。还可将 Sun Microsystems 的 JNDI LDAP 提供程序与 SDK 和 JRE 1.2.2 配合使用，但您必须从 Sun Microsystems JNDI  Web 站点下载这些组件，由 Sun Microsystems 提供支持。本编程指南描述将 IBM JNDI LDAP 提供程序与 SDK 或 JRE 1.2.2 配合使用。

要使用 IBM JNDI LDAP 提供程序来编译或运行代码，请将以下内容添加至类路径:

```
/QIBM/ProdData/Java400/ext/ibmjndi.jar:/QIBM/ProdData/Java400/ext/jndi.jar
```

本编程指南讨论了下列主题:

创建初始上下文
本主题描述如何创建初始上下文来与 LDAP 服务器连接。 JNDI 支持两种不同的方法来让客户机使用“轻量级目录访问协议”（LDAP）服务器:

- 客户机在上下文创建时标识服务器。
- 直接将 URL 字符串传送至上下文的方法。

LDAP V3 URL
本主题定义 LDAP URL 语法。

服务器绑定和 SASL 支持
在允许特定的操作之前，服务器必须对客户机进行认证。LDAP 将此称为绑定至服务器。LDAP 协议将其认证扩展为还支持“简单认证和安全性层”（SASL）机制。这些机制允许使用更复杂的方法来向服务器标识客户机，而不会因为要以明文发送用户标识和密码而危及用户的安全性。

搜索和获取属性
JNDI 允许灵活地搜索“轻量级目录访问协议”（LDAP）目录。

在目录中添加和删除项
JNDI 允许在目录中添加和删除项。本主题包含有关如何执行这些任务的示例。

更改属性
JNDI 允许对目录项更改、创建或除去属性。

重命名目录项
JNDI 允许在任何相对于基本上下文的位置处重命名目录项。本主题标识了一个将对用来重命名目录项的 rename 方法产生影响的属性。

参照和搜索引用
LDAP 服务器可能会返回参照或搜索引用。任何操作都可能会返回参照，它指示服务器没有请求的目标项。只有搜索操作才会返回搜索引用。

LDAP 控件
LDAP V3 规范添加了用于发送和接收扩展信息的控件。发送至服务器的控件称为请求控件。从服务器接收的控件称为响应控件。

二进制属性
LDAP 协议对检索到的二进制和文本属性不加任何区别。而是，它期望客户机应用程序了解如何处理数据。本主题描述了三种不同的方法来处理对检索到的属性执行的操作以及是否将它们转换为字符串。

模式
您可以检索、查看和更新 LDAP 服务器的模式结构。只支持提供由“轻量级目录访问协议”规范定义的模式信息的服务器。

SASL 插件
您可以编写自己的“简单认证和安全性层”（SASL）插件。本主题提供了用于帮助您开始创建插件的代码示例。

客户机端高速缓存
高速缓存提供了一种方法来以本地方式存储最近请求的信息。由于将以本地方式检索重复的查询而不是返回远程服务器来获取已获得的信息，所以这能改进性能。

检索 IBMJNDI 类版本
本主题指示可使用什么静态方法来检索 LDAP 的 IBMJNDI 类版本。

一致性注意事项和附加属性
请查看本主题以了解使用 IBM LDAP 提供程序和 Sun Microsystems 的“LDAP 服务提供程序 JNDI 实现者指导方针”时必须注意的事项。还指示了支持但已不赞成使用的属性。

创建初始上下文

“Java(TM) 命名和目录接口”（JNDI）支持两种不同的方法来让客户机使用“轻量级目录访问协议”（LDAP）服务器。第一种方法（也是最常见的方法）是让客户机在创建上下文时标识服务器。然后，通过将基于 DN 的名称传送至上下文的方法来对这个打开的连接执行操作。以下两个属性支持此类操作:

java.naming.factory.initial（Context.INITIAL_CONTEXT_FACTORY）
必须将此属性设置为 com.ibm.jndi.LDAPCtxFactory。

java.naming.provider.url（Context.PROVIDER_URL）
此属性以 URL 字符串的形式标识 LDAP 服务器的名称和端口。如果不能标识 LDAP 服务器的名称和端口，则 IBM JNDI LDAP 提供程序缺省为 ldap://localhost:389。

以下代码创建与主机 `ldapserver` 的连接并检索一个项:

示例 1: 创建与主机 ldapserver 的连接

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
Properties env = new Properties();
env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
env.put("java.naming.provider.url", "ldap://ldapserver");
DirContext ctx = new InitialDirContext(env);
Attributes entry = ctx.getAttributes("cn=example,o=IBM,c=US");
```

第二种通过使用 JNDI 来使用 LDAP 服务器的方法是将 URL 字符串直接传送至上下文的方法。然而，此过程具有需要为每个操作创建新连接这一开销，如果所有操作都是对单一服务器绑定的，则应避免使用此方法。以下属性支持此类操作:

java.naming.factory.url.pkgs（Context.URL_PKG_PREFIXES）
如果在创建上下文时不需要将 URL 字符串作为名称输入传送至上下文的方法和连接至 LDAP 服务器，则必须将此属性设置为 com.ibm.jndi。

以下代码重复前一个示例，但将连接服务器的操作延迟到调用 getAttributes 方法之后进行:

示例 2: 创建与主机 ldapserver 的连接并延迟连接服务器

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
Properties env = new Properties();
env.put("java.naming.factory.url.pkgs", "com.ibm.jndi");
DirContext ctx = new InitialDirContext(env);
Attributes entry = ctx.getAttributes("ldap://ldapserver/cn=example,o=IBM,c=US");
```

提供程序还支持将先前这两种方法混合使用。也就是说，有可能使用 java.naming.factory.initial 来建立与 LDAP 服务器的连接，然后将 URL 字符串作为名称输入传送至方法。无论是否已定义 java.naming.factory.url.pkgs，此方法都能起作用。

注意: 必须通过调用 close 方法来关闭任何由 **InitialDirContext** 打开的与 LDAP 服务器的连接。 «

LDAP V3 URL

» IBM Java(TM) 命名和目录接口（JNDI）轻量级目录访问协议（LDAP）提供程序完全支持 RFC 2255 中定义的 LDAP “统一资源定位器”（URL）格式。LDAP URL 是通过以下语法定义的:

```
scheme "://" [host [ ":" port ]] ["/"
    [dn ["?" [attributes] ["?" [scope]
    ["?" [filter] ["?" extensions]]]]]]
```

其中:

- *scheme* 指示 URL 方案。这个类库支持传统的 ldap（用于正常的 LDAP 连接）或 ldaps（用于 “安全套接字层”（SSL）连接）。
- *host* 是 LDAP 服务器的名称。如果未指定 LDAP 服务器名，则缺省名称是 localhost。
- *port* 表示 LDAP 服务器的端口号。如果未指定端口号，则对于非 SSL，缺省端口号是 389，对于 SSL，缺省端口号是 636。
- *dn* 标识操作的基本对象。
- *attributes* 表示要返回的属性的逗号分隔列表。如果未指定属性列表，则缺省情况是返回所有属性。
- *scope* 表示搜索作用域。这些文件的有效值如下:
  - **base**
    这个值表示基本对象。
  - **sub**
    这个值表示文件层次结构的次级。
  - **one**
    这个值表示文件层次结构的一层。

  如果未指定作用域，则缺省值是 base。
- *filter* 表示搜索过滤器。如果未指定搜索过滤器，则缺省搜索过滤器是（objectclass=*）。
- *extension* 向 LDAP URL 提供扩展性机制，从而允许扩展 URL 的能力。IBM JNDI LDAP 提供程序所支持的唯一扩展是 bindname。

服务器绑定和 SASL 支持

» 在许多情况下，在允许特定的操作之前，服务器必须对客户机进行认证。“轻量级目录访问协议”（LDAP）将此称为绑定至服务器。

当绑定到服务器时，客户机需指定要使用的 LDAP 协议。已定义了两个版本的 LDAP 协议，即 V2 和 V3。如果服务器只支持 V2 的话，当客户机尝试作为 V3 客户机来绑定时将返回协议错误。IBM 的 “Java(TM) 命名和目录接口”（JNDI）LDAP 提供程序支持作为 V2 或 V3 客户机进行绑定。

当绑定到服务器时，可使用下列属性:

java.naming.ldap.version

此属性指定 LDAP 协议版本。有效值是 2 或 3。如果未设置此属性，则提供程序尝试作为 V3 客户机绑定，并且在返回了协议错误时自动降级为 V2。如果设置了此属性，则提供程序不会尝试降级。

除了设置协议版本之外，绑定还向服务器标识用户以进行认证。

java.naming.security.principal (Context.SECURITY_PRINCIPAL)

此属性指定客户机的标识。在几乎所有情况下，它都具有专有名称的格式。

java.naming.security.credentials (Context.SECURITY_CREDENTIALS)

此属性指定客户机的凭证（即客户机的密码）。

LDAP 还支持不同类型的认证机制。V2 LDAP 协议只支持一种类型的绑定，该绑定称为简单绑定。对于此机制，将明文标识和凭证发送至服务器。V3 协议将认证扩展为还支持 “简单认证和安全性层”（SASL）机制。这些机制允许使用更复杂的方法来向服务器标识客户机，而不会因为要以明文发送用户标识和密码而危及用户的安全性。

提供程序支持两种不同的指定认证机制的方法。一种方法需要认证类的名称。这种方法允许通过指定位于提供程序外部的认证类来扩展提供程序。因此，您可以编写自己的 SASL 插件。下列属性支持这种指定认证机制的方法。

java.naming.security.sasl

此属性指定要使用的认证类的名称。下列类是作为提供程序的一部分交付的。

com.ibm.ldap.LDAPSimpleBind

此属性指定要发送至服务器以进行认证的明文标识和凭证。V2 和 V3 服务器都支持此机制。注意，当更低的层没有执行认证或加密时，不建议在开放网络上使用明文密码。

com.ibm.ldap.LDAPSaslExternal

外部 SASL 方法尝试使用已协商的下层安全性协议（如 SSL）来进行绑定。在大多数情况下，应保持安全性主体和凭证处于未初始化状态。

com.ibm.ldap.LDAPSaslCRAM_MD5

CRAM-MD5 SASL 使用提问 — 响应协议来将安全性主体和凭证发送至服务器以进行认证。

com.ibm.ldap.LDAPSaslGSSAPI

在通过单独的方法（如 kinit 或集成登录）获取凭证之后，GSSAPI SASL 方法尝试使用 kerberos 认证来进行绑定。在大多数情况下，应保持安全性主体和凭证绑定自变量处于未初始化状态。

java.naming.sasl.mode

此属性指定要传送给所装入的 SASL 插件的模式设置。提供程序中的所有预定义 SASL 插件都忽略此设置。

支持的第二种指定认证机制的方法与 Sun Microsystems 的 LDAP 工具箱兼容。并不是指示要装入的认证类，而是指定认证机制的名称。如果未设置 java.naming.security.sasl 属性，则 IBM JNDI LDAP 提供程序使用此方法来指定认证机制。

java.naming.security.authentication（Context.SECURITY_AUTHENTICATION）

此属性指定要使用的认证机制的名称。此属性支持下列值。

- none
  不执行认证（匿名绑定）。
- simple
  使用简单认证。
- EXTERNAL
  使用外部 SASL 机制。
- CRAM-MD5
  使用 CRAM-MD5 SASL 机制。
- GSSAPI
  使用 GSS 或 Kerberos SASL 机制。

在成功认证客户机之后，此类库设置以下属性:

java.naming.authorization.identity
此属性设置为客户机的权限身份。正常情况下，这与指定的客户机标识相同。然而，SASL 机制可将初始绑定 DN 映射至另一个值。将权限身份存储在客户机证书中的外部 SASL 就是这样的一个示例。

以下示例演示设置属性以指示版本协议为 3，并使用 CRAM-MD5 机制来作为 Larry Meade 进行认证:

示例: 设置属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
Properties env = new Properties();
env.put("java.naming.factory.initial", "com.ibm.jndi.LDAPCtxFactory");
env.put("java.naming.ldap.version", "3");
env.put("java.naming.provider.url", "ldap://ldapserver");
env.put(Context.SECURITY_PRINCIPAL, "cn=Larry Meade, o=IBM, c=US");
env.put(Context.SECURITY_CREDENTIALS, "secret");
env.put(Context.SECURITY_AUTHENTICATION, "CRAM-MD5");
DirContext ctx = new InitialDirContext(env);
```

可将前一个示例更改为显式地指示 SASL 类名称。需要将前一个示例中的 SECURITY_AUTHENTICATION 一行替换为下面这一行:

```
env.put("java.naming.security.sasl", "com.ibm.ldap.LDAPSaslCRAM_MD5");
```

搜索和获取属性

“Java(TM) 命名和目录接口”（JNDI）在搜索“轻量级目录访问协议”（LDAP）目录方面提供了极大的灵活性。在 IBM JNDI LDAP 提供程序中，两个最常用的方法是 search 和 getAttributes。然而，下列方法也从 LDAP 服务器检索数据:

- lookup
- lookupLink
- list
- listBindings
- getSchema
- getSchemaClassDefinition

下列属性会对搜索操作产生影响:

java.naming.ldap.derefAliases
此属性定义如何处理别名对象（在 X.501 中定义）。此属性接受下列值:

- always
  这个值在搜索和定位搜索的基本对象时取消引用别名。这是缺省值。
- never
  这个值在搜索或定位搜索的基本对象时不取消引用别名。要提高性能，这是建议的设置。
- finding
  这个值在定位搜索的基本对象时取消引用别名，但在搜索基本对象的下属时不这样做。
- searching
  这个值在搜索时取消引用基本对象的下属中的别名，但在定位搜索的基本对象时不这样做。

java.naming.batchsize（Context.BATCHSIZE）

这个值设置返回的 NamingEnumeration 所存放的搜索结果数的建议大小设置。如果未指定值，则缺省 batchsize 是 1。这帮助确保类库的最小可能内存足迹。0 值将禁用 batchsize 并指示在收集到所有结果之前搜索是分块的。

java.naming.ldap.typesOnly

此属性与 getAttributes 和 search 方法相关，并且仅当返回对象标志为 false 时才与后者相关。此属性接受下列值:

- true
  这个值只返回属性标识符，但不返回值。
- false
  这个值既返回属性标识符也返回值。这是缺省值。

搜索调用的结果是 NamingEnumeration。要获取结果，必须使用传统的 hasMoreElements 和 nextElement 方法或特定于 NamingEnumeration 的 hasMore 和 next 方法来对枚举进行遍历。如果您想要追踪或查看 ReferralException 的话，后两个方法允许捕获异常。

注意:

- 为了避免意外地保持资源处于已分配状态和连接处于打开状态，应当将 NamingEnumeration 遍历至末尾（即直到 hasMore 和 hasMoreElements 方法返回 false 为止），或者应调用枚举的 close 方法。
- IBM JNDI LDAP 提供程序自动地用 * 字符来替换专有名称和属性类型名称中的无效 UTF-8 字符编码。这样做是为了防止单一无效值导致整个可能很长的搜索失败。

以下示例对 surname 为 smith 的所有项执行搜索并指示只返回 cn 属性:

示例: 搜索和获取 cn 属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
SearchControls constraints = new SearchControls();
constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
String attrList[] = {"cn"};
constraints.setReturningAttributes(attrList);
NamingEnumeration results =
    ctx.search("o=IBM,c=US", "(sn=smith)", constraints);
while (results.hasMore()) {
    SearchResult si =(SearchResult)results.next();
    System.out.println(si.getName());
    Attributes attrs = si.getAttributes();
    if (attrs == null) {
        System.out.println("   No attributes");
        continue;
    }
    NamingEnumeration ae = attrs.getAll();
    while (ae.hasMoreElements()) {
        Attribute attr =(Attribute)ae.next();
```

```
            String id = attr.getID();
            Enumeration vals = attr.getAll();
            while (vals.hasMoreElements())
                System.out.println("   "+id + ": " + vals.nextElement());
        }
    }
```

以下示例使用 list 方法来显示某个基本专有名称（DN）之下的名称:

示例: 显示基本 DN 之下的名称

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
String url="ldap://ldapserver:389/o=IBM,c=US";
NamingEnumeration listResults=ctx.list(url);
while (listResults.hasMore()) {
    NameClassPair ncp = (NameClassPair) listResults.next();
    System.out.println(ncp.getName());
}
```

在目录中添加和删除项

“Java(TM) 命名和目录接口”（JNDI）允许在目录中添加和删除项。以下示例添加具有 objectclass、roomnumber 和 telephonenumber 属性的新项:

示例: 在目录中添加项

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
BasicAttribute objClasses = new BasicAttribute("objectclass");
objClasses.add("person");
objClasses.add("organizationalPerson");
objClasses.add("inetOrgPerson");

BasicAttributes attrs = new BasicAttributes();
attrs.put(objClasses);
attrs.put("roomnumber", "2000");
attrs.put("telephonenumber", "1-800-use-LDAP");

ctx.createSubcontext(name, attrs);
```

以下示例除去项:

示例: 删除目录中的项

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
ctx.destroySubcontext(name);
```

更改属性: JNDI 允许对目录项更改、创建或除去属性。以下示例替换一个项的 roomnumber 属性:

示例: 更改属性

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
ctx.modifyAttributes(name,
    DirContext.REPLACE_ATTRIBUTE,
    new BasicAttributes("roomnumber", "5000"));
```

以下示例将新的 telephonenumber 属性值添加至一个项并除去 roomnumber 属性:

示例: 更改目录中的项

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
ModificationItem[] mods=new ModificationItem[2];
mods[0] = new ModificationItem(DirContext.ADD_ATTRIBUTE,
    new BasicAttribute("telephonenumber", "456-7777"));
mods[1] = new ModificationItem(DirContext.REMOVE_ATTRIBUTE,
    new BasicAttribute("roomnumber"));
ctx.modifyAttributes(name, mods);
```

重命名目录项: 可以使用 rename 方法来在任何相对于基本上下文的位置处重命名目录项。

以下属性会对 rename 方法产生影响:

java.naming.ldap.deleteRDN

此属性在重命名项时除去旧的 RDN。缺省设置为 true。

如果将此属性设置为 false，则保留旧的 RDN 来作为项的属性值。

下面是调用 rename 方法的示例:

示例: 重命名目录项

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
String oldname="cn=bill smith";
String newname="cn=bill smith, ou=programmer";
ctx.rename(oldname, newname);
```

参照和搜索引用

“轻量级目录访问协议”（LDAP）服务器可能会返回参照或搜索引用。任何操作都可能会返回参照，它指示服务器没有请求的目标项。只有搜索操作才会返回搜索引用。搜索引用指示服务器能够定位 baseObject 所引用的项，但无法搜索位于该 baseObject 处以及其下的作用域中的所有项。服务器可返回一个或多个搜索引用。

可以配置上下文来以三种方法之一处理参照和搜索引用:

1. 可以将其设置为自动跟随引用并在被引用服务器上执行操作。IBM 的 “Java(TM) 命名和目录接口”（JNDI）LDAP 提供程序能够自动识别和避免参照循环；即，参照指向回到链中先前已追踪的参照的情况。
2. 可以将其设置为在接收到参照或搜索引用时抛出 ReferralException。如果自动处理有些不完善的话，这就非常有用；例如，每个服务器都需要不同的绑定。
3. 可以将其设置为忽略引用并继续，就象没发生任何事情一样。对于搜索引用这种情况，这表示只返回在起源服务器上找到的项。

IBM JNDI LDAP 提供程序定义了下列环境属性来处理参照和搜索引用:

java.naming.referral（Context.REFERRAL）

此属性设置为 follow、throw 或 ignore。如果未设置此属性，则将缺省值设置为自动跟随参照。

java.naming.ldap.referral.limit

此属性定义追踪参照时类库进行的参照跳转数。如果未指定属性值，则缺省值是 10。

java.naming.ldap.referral.bind

如果将此属性设置为 true，则在自动跟随参照时，类使用与起源上下文相同的 SASL 机制和凭证来绑定至任何被引用服务器。如果设置为 false，则类不绑定（即进行匿名访问）。缺省行为是绑定。

在追踪参照或搜索引用时，下列规则适用:

- 如果引用包含端口，则使用该端口。否则使用来自主连接的端口。
- 维护来自主连接的安全性连接类型。即，如果主连接基于 SSL，则追踪的所有参照也都基于 SSL。

注意: 即使设置为 follow，上下文也仍可能抛出 ReferralException。如果超过参照跳转限制，或者如果上下文不能连接或绑定至任何被引用服务器，则会发生这种情况。

以下示例捕获并显示某个搜索请求上的参照和搜索引用。有关处理参照异常的更多信息，参见 Sun Microsystems 的 ReferralException  。

示例: 捕获和显示参照和搜索引用

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
ctx.addToEnvironment(ctx.REFERRAL, "throw");
try {
    NamingEnumeration results = ctx.search(url);
    while (true) {
        try {
            if (!results.hasMore())
                break;
            SearchResult si =(SearchResult) results.next();
            System.out.println(si.getName());
        } catch (ReferralException re) {
            System.out.println("Reference caught");
            do {
                System.out.println(re.getReferralInfo());
            } while (re.skipReferral());
        }
    }
} catch (ReferralException re) {
    System.out.println("Referral caught");
    do {
        System.out.println(re.getReferralInfo());
    } while (re.skipReferral());
}
```

LDAP 控件

“轻量级目录访问协议”（LDAP）V3 规范添加了用于发送和接收扩展信息的控件。发送至服务器的控件称为**请求控件**，告知服务器按指定的属性对搜索结果进行排序的请求控件就是一个示例。具体支持哪些请求控件完全取决于服务器（即，某个控件在一种类型的服务器上可能能够起作用，但在另一服务器上却可能会失败）。从服务器接收的控件称为**响应控件**。

“Java(TM) 命名和目录接口”（JNDI）1.2 规范将请求控件分成两个截然不同的类别: 连接至服务器时使用的请求控件以及在任何其它操作上使用的请求控件。有关请求控件的更多信息，请参考 Sun 的 JNDI 文档中的 LdapContext。

IBM JNDI LDAP 提供程序附带了一个名为 ManageDsaIT 的预定义控件。此控件强制服务器将搜索引用视为正常的 LDAP 项，从而允许查看和更改它们，而不是查看或更改它们所引用的数据。以下示例演示了如何启用此控件:

示例: 启用 ManageDsaIT 控件

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
import com.ibm.jndi.ldap.control.ManageDsaIT;

Control[] cntl = new Control[1];
cntl[0] = new ManageDsaIT();
ctx.setRequestControls(cntl);
```

仅当客户机绑定至服务器时连接控件才是活动的。

以下是有关如何启用连接控件的示例:

```
LdapContext ctx = new InitialLdapContext(env, cntl);
```

以下 API 用来检索上次接收到的响应控件:

```
Control[] cntl = ctx.getResponseControls();
```

有关启用能够将原始控件数据映射至特定控件类的控件生成器的详细信息，请参考 Sun 的文档。 «

二进制属性

» “轻量级目录访问协议”（LDAP）协议对检索到的二进制和文本属性不加任何区别。而是，它期望客户机应用程序了解如何处理数据。由于能够转换文本属性并将它们还原为 Java 字符串，所以 IBM JNDI LDAP 提供程序对您非常有帮助。然而，提供程序需要知道哪些属性是二进制的以及哪些属性表示字符数据。提供程序支持三种不同的方法来处理对检索到的属性执行的操作以及是否将它们转换为字符串。

当检索到属性时，提供程序检查已知二进制属性名的列表。提供程序已编程为能够识别下面这一组公共 LDAP 二进制属性:

- userPassword
- userCertificate
- cACertificate
- authorityRevocationList
- certificateRevocationList
- deltaRevocationList
- crossCertificatePair
- x500UniqueIdentifier
- photo
- personalSignature
- audio
- jpegPhoto
- javaSerializedObject
- thumbnailPhoto
- thumbnailLogo
- supportedAlgorithms
- protocolInformation

您可使用以下属性来指定它们自己的二进制属性名列表:

java.naming.ldap.attributes.binary（LDAPCtx.ATTRIBUTES_BINARY）
除提供程序定义的缺省集合之外的用户定义二进制属性名的空格分隔列表。

以下示例标识两个附加的用户定义二进制属性:

示例: 指定二进制属性名的列表

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
ctx.addToEnvironment(LDAPCtx.ATTRIBUTES_BINARY,
    "gifPhoto fingerPrint");
```

第二种处理二进制属性的方法是识别某些 V3 服务器支持的二进制描述选项。如果将 “;binary” 附加至属性名，如 “jpegPhoto;binary” ，则指示该属性是二进制值。当作为 V3 用户绑定时，提供程序识别二进制属性描述选项。

最后，提供程序尝试转换任何未定义为二进制并且不包含二进制属性描述选项的属性。如果转换失败，则将数据作为二进制返回。然而，由于有可能不正确地转换 UTF-8 数据，所以您不应依赖于此项工作。 «

模式

» 您可以检索、查看和更新 “轻量级目录访问协议” （LDAP）服务器的模式结构。只支持提供了轻量级目录访问协议（V3） 文档定义的模式信息的服务器。

要检索服务器的模式，请使用 getSchema 方法。返回的模式是作为层次结构表示的，每个下属层都是模式的不同组件。IBM 的 “Java(TM) 命名和目录接口” （JNDI）LDAP 提供程序能够分析下列模式组件:

- AttributeTypes，存储在 AttributeDefinition 亚目下面
- 对象类，存储在 ClassDefinition 亚目下面
- 语法描述，存储在 SyntaxDefinition 亚目下面
- 匹配规则，存储在 MatchingRule 亚目下面
- IBM 属性类型，存储在 IBMAttributeDefinition 亚目下面

除 IBMAttributeDefinition 之外的项的内容与轻量级目录访问协议（V3）: 属性语法定义 中定义的模式具有一一对应关系。

IBMAttributeDefinition 扩展了一个属性模式以存放特定于 IBM 的信息。它是使用以下 Backus-Naur 范式（BNF）定义的:

```
IBMAttributeTypesDescription = "(" whsp
    numericoid whsp
    [ "DBNAME"   qdescrs ]            ; at most 2 names (table, column)
    [ "ACCESS-CLASS" whsp IBMAccessClass whsp ]
    [ "LENGTH" wlen whsp ]            ; maximum length of attribute
    [ "EQUALITY" [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "ORDERING" [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "APPROX"   [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "SUBSTR"   [ IBMwlen ] whsp ]   ; create index for matching rule
    [ "REVERSE"  [ IBMwlen ] whsp ]   ; reverse index for substring
    whsp ")"

IBMAccessClass =
    "NORMAL"            / ; this is the default
    "SENSITIVE"         /
    "CRITICAL"          /
    "RESTRICTED"        /
    "SYSTEM"            /
    "OBJECT"

IBMwlen = whsp len
```

将把服务器返回的但此类库不支持的模式定义保存下来，不对它们进行分析。这些项刚好包含两个属性，一个用于 objectclass 等于模式类型名（例如 objectclass=adddef），另一个用于值列表。可查看不支持的模式定义，但不能更新它们。

以下示例检索整个模式层次结构:

示例: 检索模式层次结构

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
DirContext schemaCtx = ctx.getSchema("");
SearchControls cons = new SearchControls();
cons.setSearchScope(SearchControls.SUBTREE_SCOPE);
NamingEnumeration ne = schemaCtx.search("",
    "(|(NUMERICOID=*)(objectclass=*))", cons);
```

以下示例检索 cn 属性的模式:

示例: 检索 cn 属性的模式

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
DirContext schemaCtx = ctx.getSchema("");
Attributes attrs = schemaCtx.getAttributes("AttributeDefinition/cn");
```

以下示例尝试为新的对象类添加模式定义:

示例: 添加模式定义

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
DirContext schemaCtx = ctx.getSchema("");
BasicAttributes attrs = new BasicAttributes();
attrs.put("NAME", "javaObject");
attrs.put("NUMERICOID", "1.3.6.1.4.1.42.2.27.4.2.2");
Attribute may = new BasicAttribute("MAY");
may.add("javaClassName");
may.add("javaSerializedObject");
attrs.put(may);
attrs.put("DESC", "Serialized Java object");
attrs.put("AUXILIARY", "true");
attrs.put("SUP", "top");
schemaCtx.createSubcontext("ClassDefinition/javaObject", attrs);
```

已特殊地将 com.ibm.jndi.LDAPSchemaCtx 类扩展为使用文件中的模式定义。两个公用构造函数支持从磁盘读取模式信息。其中一个构造函数接收单一文件名作为自变量，另一个构造函数接收文件名数组。dumpSchema 方法将模式定义保存至文件。下面演示此项支持:

示例: dumpSchema 方法

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
LDAPSchemaCtx ctx = new LDAPSchemaCtx("schema.file");
ctx.dumpSchema("schema.sav");
```

SASL 插件

» 您可以编写自己的“简单认证和安全性层”（SASL）插件。SASL 插件必须从抽象基类 LDAPSaslBind 派生。必须实现 bind 方法，这个方法接着调用 SendBindRequest 方法来与服务器通信。以下是一个简单的绑定插件的示例:

示例: 绑定插件

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
import java.io.IOException;
import com.ibm.asn1.ASN1Exception;
import com.ibm.ldap.*;

public class SimpleBind extends LDAPSaslBind
{
    public boolean bind(String dn, String credentials)
        throws IOException, ASN1Exception, LDAPException
        {
                return SendBindRequest("SIMPLE", dn, credentials);
        }

}
```

如果 SASL 协议包含服务器提问，则必须将 SendBindRequest 方法调用多次，这可能需要使用 getServerCredentials 方法来检索服务器提问信息。以下是 CRAM-MD5 SASL 插件的一个示例:

示例: CRAM-MD5 SASL 插件

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
import java.net.*;
import java.io.*;
import java.security.*;
import com.ibm.util.*;
import com.ibm.asn1.ASN1Exception;
import com.ibm.ldap.*;

/*

    Challenge-Response Authentication Mechanism / MD5 hash.
    See RFC 2195 ("IMAP/POP AUTHorize Extension for Simple
    Challenge/Response") and draft-ietf-ldapext-authmeth-02
    ("Authentication Methods for LDAP") for details.

*/
public class CramMD5 extends LDAPSaslBind
{
    public boolean bind(String dn, String credentials)
        throws IOException, ASN1Exception, LDAPException
    {
        String clientCreds;

        // Send initial bind request
        if (SendBindRequest("CRAM-MD5", dn, null) == true)
        return false;

        // Generate md5 hash from client's secret and server's
        // challenge and send to server
           try {
            clientCreds = "dn: " + new String(stringUTF(dn)) + " " +
                HMAC_MD5(credentials, getServerCredentials());
        } catch (NoSuchAlgorithmException nsae) {
            throw new IOException(nsae.toString());
        }
        putCredentials(clientCreds);
        return SendBindRequest();
    }

    /*

        Hashed Message Authentication Code. See RFC 2104
```

```
        ("HMAC: Keyed-Hashing for Message Authentication")
        for details.

    */
    public static String HMAC_MD5(String secret, String text)
        throws NoSuchAlgorithmException
    {
        MessageDigest md5;
        byte[] ipad, opad, key;
    int i;

        // Initialize
        md5 = MessageDigest.getInstance("MD5");
        ipad = new byte[64];
        opad = new byte[64];
        key = secret.getBytes();

        // If key is larger than block size then hash key
        if (key.length > 64)
        key = md5.digest(key);

        // Perform XOR of ipad and opad with key (padded to 64 bytes).
        for (i = 0; i < key.length; ++i) {
            ipad[i] = (byte)(0x36 ^ key[i]);
            opad[i] = (byte)(0x5c ^ key[i]);
        }
        while (i < 64) {
            ipad[i] = 0x36;
            opad[i++] = 0x5c;
        }

        // Hash ipad XOR result and text
        md5.update(ipad);
        key = md5.digest(text.getBytes());

        // Hash opad XOR result and previous hash
        md5.update(opad);
        key = md5.digest(key);

        // Return hex representation of hash (32 bytes)
        return Hex.toString(key, false);
    }
}
```

客户机端高速缓存


高速缓存提供了一种方法来以本地方式存储最近请求的信息。由于将以本地方式检索重复的查询而不是返回远程服务器来获取已获得的信息，所以这能改进性能。缺点是没有办法确定高速缓存中的值在检索之后是否已更改。因特网草稿轻量级目录访问协议（LDAP）和 X.500 目录的简单高速缓存方案通过定义“使用期限”属性解决了此问题，该属性指示了一个项在变旧之前可以在高速缓存中合理停留的时间长度。客户机端高速缓存的实现是这个类库完全支持此草稿。

这个类库中使用的高速缓存算法完全基于搜索请求。不以本地方式复制数据；因此，不可能以两种不同的方法从高速缓存查询项。例如，在搜索“cn=Joe*”之后查询“cn=Joe Smith”并不会从高速缓存检索值，尽管结果可能作为“cn=joe*”结果集的一部分驻留在本地。复制 LDAP 数据会带来许多微妙的问题并且超出此高速缓存方案的范畴，特别是在 Java(TM) 中更是如此。与从经过优化的服务器检索结果相比，在 Java 中以本地方式搜索复杂查询所花费的时间可能要更长。

不应该将高速缓存用于有可能快速更改的数据，检索关键信息时也不应该使用高速缓存。在这个类库中，高速缓存在缺省情况下处于禁用状态，当它处于启用状态时，提供了一种方法来绕过高速缓存并直接从服务器检索结果。然而，LDAP 数据通常是静态的并且经常基于信息，因此对于许多应用程序而言，具有准确使用期限值的高速缓存是合理的模型。

当处于启用状态时，将对从服务器检索数据的操作使用高速缓存，并包括下列方法:

- getAttributes
- search
- lookup
- lookupLink
- list
- listBindings
- getSchema
- getSchemaClassDefinition

高速缓存是在将 LDAPCache 类型的对象实例化时创建的。多个上下文可共享单一高速缓存，这表示一个上下文可从高速缓存中检索另一上下文存放在高速缓存中的值。IBM JNDI LDAP 提供程序中的高速缓存还具有线程安全性。LDAPCache 的另一个功能是可以将其序列化至磁盘，并且以后可以恢复。可以更改高速缓存的大小（高速缓存的大小基于所检索的项数）。

上下文不直接使用 LDAPCache，但它依靠 LDAPCacheControl 来管理高速缓存。LDAPCacheControl 提供了共享高速缓存的个性化视图。LDAPCacheControl 允许逐个请求地调整设置，如使用期限值，以及是否绕过高速缓存并直接从服务器检索结果。尽管可逐个上下文地使用 LDAPCacheControl，但也可以在多个上下文之间共享它。将把上下文中的任何 LDAPCacheControl 引用转发至由 lookup、lookupLink 和 listBindings 方法创建的子上下文。

提供程序将 java.naming.control.cache 属性定义成一种将高速缓存控制与上下文相关联的方法。可以在创建上下文时这样做，以后也可以通过 addToEnvironment 方法进行此操作。

要启用高速缓存，必须将 LDAPCacheControl 类型的对象实例化并使其与上下文相关联。LDAPCacheControl 构造函数需要一个 LDAPCache 对象来作为自变量。以下是在启用了高速缓存的情况下创建上下文的示例:

示例: 在启用了高速缓存的情况下创建上下文

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

```
import com.ibm.jndi.LDAPCtx;
import com.ibm.ldap.LDAPCache;
import com.ibm.ldap.LDAPCacheControl;

LDAPCache cache = new LDAPCache();
LDAPCacheControl cacheControl = new LDAPCacheControl(cache);
env.put("java.naming.control.cache", cacheControl);
DirContext ctx = new InitialDirContext(env);
```

有可能逐个请求地调整设置。在将设置清除之前，它们将保持有效。以下代码强制 60 秒的使用期限值:

```
cacheControl.putTTL(60);
cacheControl.putHonorServerTTL(false);
ctx.search(...);
```

以下代码绕过高速缓存中存储的任何值并直接从服务器检索结果:

```
cacheControl.putReadFlag(false);
ctx.search(...);
```

注意: 由于存在 java.naming.batchsize 属性，所以在完全枚举结果的时候就将停止将项添加至高速缓存的操作。

检索 IBMJNDI 类版本

可使用以下静态方法来获取“轻量级目录访问协议”（LDAP）类的版本。

```
String version = com.ibm.jndi.LDAPCtx.getVersion();
```

一致性注意事项和附加属性

在 IBM 的“Java(TM) 命名和目录接口”（JNDI）LDAP 提供程序与 Sun 的 LDAP 服务提供程序 JNDI 实现者指导方针（草稿 0.2） 之间存在着下列已知差异。

1. 不支持命名联合。
2. 在 iSeries 服务器上，IBM JNDI LDAP 提供程序不支持使用“安全套接字层”（SSL）协议。
3. 不应使用 IBM LDAP 提供程序来在 LDAP 目录中存储序列化的 Java 对象。
4. 传送不需要的 URL 组件不会导致 ConfigurationException 或 InvalidNameException。
5. IBM JNDI LDAP 提供程序拥有自己的“简单认证和安全性层”（SASL）插件支持。由于 Java SASL“应用程序编程接口”（API）目前是作为仅预览的包存在的，所以 IBM JNDI LDAP 提供程序不支持此 API。
6. 提供程序支持主动通知事件，但不支持名称空间或对象更改事件。
7. java.naming.referral 的缺省值是 follow 而不是 ignore。当将此属性设置为 ignore 时，提供程序既不自动地将 ManageDSAIt 控件添加至请求，也不在接收到参照时抛出 PartialResultException。

还支持下列属性:

com.ibm.jndi.ldap.so_timeout
此属性定义上下文块等待来自服务器的数据的毫秒数。缺省超时是 5 分钟。将零超时解释为无穷超时。

已不赞成使用下列属性:

- java.naming.ldap.noBind
- java.naming.control.server

JSSL

“Java 安全套接字层”（JSSL）是一组启用安全因特网通信的 Java 包。它实现了 SSL 和“传输层安全性”（TLS）协议的 Java 版本，并包括有关数据加密、服务器认证、消息完整性和可选客户机认证的功能。通过使用 JSSL，您可以开发能够通过 TCP/IP 来在客户机与运行任何应用程序协议（如 HTTP、Telnet、NNTP 和 FTP）的服务器之间安全传送数据的应用程序。

有关 JSSL 的更多信息，参见 Sun Microsystems 的 Java 安全套接字扩展（JSSE） 。

JavaMail

JavaMail(TM) API 提供了一组模仿电子（电子邮件）系统的抽象类。此 API 提供了用于读取和发送邮件的一般邮件功能，并要求服务提供程序实现协议。

服务提供程序实现特定的协议。例如，“简单邮件传送协议”（SMTP）就是用于发送电子邮件的传送协议。“邮局协议 3”（POP3）就是用于接收电子邮件的标准协议。“因特网消息访问协议”（IMAP）是 POP3 的备用协议。

除了服务提供程序之外，JavaMail 还要求“JavaBean 激活框架”（JAF）处理非纯文本的邮件内容。这包括“多用途因特网邮件扩展”（MIME）、“统一资源定位器”（URL）页面和文件附件。

所有 JavaMail 组件都是作为 IBM Developer Kit for Java 的一部分交付的。这些组件包括:

- **mail.jar**
  这个 JAR 文件包含 JavaMail API、SMTP 服务提供程序、POP3 服务提供程序和 IMAP 服务提供程序。
- **activation.jar**
  这个 JAR 文件包含“JavaBean 激活框架”。

有关更多信息，请参考 Sun Microsystems 的 JavaMail 文档。 

Java 打印服务

 “Java(TM) 打印服务”API 允许在所有 Java 平台上打印。Java 1.4 提供了一个框架，在这个框架中，Java 运行时环境和第三方可以提供流生成器插件，用于生成各种可打印格式，如 PDF、Postscript 和“高级功能显示”（AFP）。这些插件从二维（2D）图形调用生成输出格式。

有关更多信息，请参考 Sun Microsystems 的 Java 打印服务文档  。 

第 8 章 使用 IBM Developer Kit for Java 来调试程序

如果需要调试 Java(TM) 程序，则请选择下列选项之一:

- 调试 Java 程序
- 调试 Java 和本机方法程序
- 从另一个屏幕调试 Java 程序
- » 调试通过定制类装入程序装入的 Java 类 «
- 调试 servlet

在调试 Java 程序时，Java 程序实际上是以批处理立即（BCI）作业形式在 Java 虚拟机中运行。源代码显示在交互式屏幕上，但 Java 程序并不在那里运行。它正在另一个作业中运行，该作业是一个被服务作业。当 Java 程序结束时，被服务作业结束，并显示一条消息，指出被服务作业已结束。

不可能调试使用“及时”（JIT）编译器运行的 Java 程序。如果某个文件没有相关联的 Java 程序，则缺省情况是运行 JIT。可通过若干种方法禁用这种情况来允许调试:

- 在启动 Java 虚拟机时指定属性 `java.compiler=NONE`。
- 在“运行 Java”（RUNJVA）命令上指定 OPTION(*DEBUG)。
- 在“运行 Java”（RUNJVA）命令上指定 INTERPRET(*YES)。
- 在启动 Java 虚拟机之前，使用 CRTJVAPGM OPTIMIZATION(10) 来创建相关联的 Java 程序。

  注意: 这些解决方案都不会影响正在运行的 Java 虚拟机。如果 Java 虚拟机不是使用这些备用方法之一启动的，则必须将其停止并重新启动才能进行调试。

这两个作业之间的接口是在您指定“运行 Java”（RUNJVA）命令上的 *DEBUG 选项时建立的。

有关系统调试器的更多信息，参见 WebSphere Development Studio: ILE C/C++ Programmer's Guide（SC09-2712）一书  和联机帮助信息。

调试 Java 程序

您可以使用许多不同的方法来调试 Java(TM) 程序。可以使用 `*DEBUG` 选项来在运行程序之前查看源代码。然后，您可以设置断点，也可以在程序运行时单步跳过或单步跳入程序来分析错误。

要调试 Java 程序，请执行下列步骤:

1. 通过使用 DEBUG 选项编译 Java 程序，该选项即为 `javac` 工具上的 -g 选项。有关更多详细信息，参见使用 *DEBUG 选项来调试 Java 程序。
2. 将类文件（.class）和源文件（.java）插入到 iSeries 服务器上的同一个目录中。
3. 通过在 iSeries 命令行上使用“运行 Java”（RUNJVA）命令来运行 Java 程序。在“运行 Java”（RUNJVA）命令上指定 OPTION(*DEBUG)。

   注意: 只能调试一个类。如果对 CLASS 关键字输入了 JAR 文件名，则不支持 OPTION(*DEBUG)。
4. 将显示 Java 程序源。
5. 按 F6 键（添加／清除断点）来设置断点，或按 F10 键（步进）来单步执行程序。有关设置断点的更多信息，参见设置断点。有关单步执行的详细信息，参见单步执行 Java 程序来进行调试。

提示:

1. 当使用断点和步进时，请检查 Java 程序的逻辑流，然后根据需要查看和更改变量。
2. 在 RUNJVA 命令上使用 OPTION(*DEBUG) 将禁用“及时”（JIT）编译器。没有相关联的 Java 程序的文件将以解释方式运行。

使用 *DEBUG 选项来调试 Java 程序

请使用 *DEBUG 选项来在运行程序之前查看源代码。*DEBUG 选项允许在代码中设置断点。

要使用 *DEBUG 选项，请在命令行上输入“运行 Java(TM)”（RUNJVA）命令，后跟类文件的名称和 OPTION(*DEBUG)。例如，iSeries 命令行应该象下面这样:

```
RUNJVA CLASS(classname) OPTION(*DEBUG)
```

注意: 如果您无权使用“启动服务作业”（STRSRVJOB）命令，则 OPTION(*DEBUG) 将被忽略。

要查看调试屏幕，请参见 Java 程序的初始调试屏幕。

Java 程序的初始调试屏幕

在调试 Java(TM) 程序时，请对程序遵循下面这些示例屏幕。这些屏幕显示了一个名为 Hellod 的示例程序。

- 输入 ADDENVVAR ENVVAR(CLASSPATH) VALUE ('/MYDIR')。
- 输入此命令: RUNJVA CLASS(HELLOD) OPTION(*DEBUG)。插入您的 Java 程序的名称来代替 HELLOD。
- 等待“显示模块源”屏幕出现。这就是 HELLOD Java 程序的源。

```
+------------------------------------------------------------------------------+
|                            显示模块源                                        |
|                                                                              |
|  类文件名: HELLOD                                                            |
|       1  import java.lang.*;                                                 |
|       2                                                                      |
|       3  public class Hellod extends Object                                  |
|       4  {                                                                   |
|       5  int k;                                                              |
|       6  int l;                                                              |
|       7  int m;                                                              |
|       8  int n;                                                              |
|       9  int o;                                                              |
|      10  int p;                                                              |
|      11  String myString;                                                    |
|      12  Hellod myHellod;                                                    |
|      13  int myArray[];                                                      |
|      14                                                                      |
|      15  public Hellod()                                                     |
|                                                                  尚有...     |
|  调试 . . .                                                                  |
|                                                                              |
|  F3 = 结束程序    F6 = 添加／清除断点    F10 = 步进          F11 = 显示变量      |
|  F12 = 继续执行   F17 = 观察变量         F18 = 使用观察      F24 = 其它键        |
|                                                                              |
+------------------------------------------------------------------------------+
```

- 按 F14 键（使用模块列表）。
- “使用模块列表”屏幕显示。可通过输入选项 1（添加程序）来添加其它要调试的类和程序。用选项 5（显示模块源）显示它们的源。

```
+------------------------------------------------------------------------------+
|                              使用模块列表                                    |
|                                                  系统:     AS400             |
|  输入选项，按“执行”键。                                                      |
|    1 = 添加程序    4 = 除去程序    5 = 显示模块源                            |
|    8 = 使用模块断点                                                          |
|                                                                              |
```

```
选项     程序/模块          库                类型
                            *LIBL             *SRVPGM
         HELLOD                               *CLASS       已选择
                                                                                底部
命令
===>
F3 = 退出    F4 = 提示    F5 = 刷新    F9 = 检索    F12 = 取消
F22 = 显示类文件名
```

- 在添加要调试的类时，可能需要输入一个比“程序 / 模块”输入字段长的包限定类名。要输入较长的名称，请遵循下面这些步骤:
  1. 输入“选项 1”（添加程序）。
  2. 将“程序 / 模块”字段保留为空白。
  3. 将库字段保留为 *LIBL。
  4. 对“类型”输入 *CLASS。
  5. 按“执行”键。
  6. 将显示一个弹出窗口，在该窗口中，有更多的空间可用来输入包限定的类文件名。

设置断点

您可以通过断点来控制程序的运行。断点使正在运行的程序停止在特定语句处。

要设置断点，请执行下列步骤:

1. 将光标放在您要设置断点的代码行上。
2. 按 F6 键（添加 / 清除断点）来设置断点。
3. 按 F12 键（继续执行）来运行程序。

注意: 就在运行设置了断点的代码行之前，将显示程序源，指示遇到了断点。

```
                              显示模块源

当前线程:     00000019     停止的线程:     00000019
类文件名:     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();
调试 . . .

F3 = 结束程序    F6 = 添加／清除断点    F10 = 步进    F11 = 显示变量
F12 = 继续执行   F17 = 观察变量    F18 = 使用观察    F24 = 其它键
  断点被添加至第 41 行。
```

当遇到断点时，如果您想要设置只在当前线程中才会遇到的断点，则使用 TBREAK 命令。

有关系统调试器命令的更多信息，参见 WebSphere Development Studio: ILE C/C++ Programmer's Guide（SC09-2712）一书  和联机帮助信息。

有关当程序在某一断点停止运行时对变量求值的信息，参见对 Java(TM) 程序中的变量求值。

单步执行 Java 程序来进行调试

在进行调试时，您可以单步执行程序。您可以单步跳过或单步跳入其它函数。Java(TM) 程序和本机方法可以使用步进功能。

当首次显示程序源时，您便可以启动步进。程序将在运行第一条语句之前停下来。请按 F10 键（步进）。继续按 F10 键（步进）便可以单步执行程序。按 F22 键（单步跳入）可以单步跳入程序调用的任何函数。您还可以在遇到断点时启动步进。有关设置断点的信息，参见设置断点。

```
                          显示模块源

当前线程:     00000019     停止的线程:     00000019
类文件名:     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();
调试 . . .

F3 = 结束程序    F6 = 添加／清除断点    F10 = 步进    F11 = 显示变量
F12 = 继续执行   F17 = 观察变量    F18 = 使用观察    F24 = 其它键
步进在线程 00000019 中的第 42 行处完成
```

要停止步进并继续运行程序，请按 F12 键（继续执行）。

有关步进的更多信息，参见 WebSphere Development Studio: ILE C/C++ Programmer's Guide（SC09-2712）一书  和联机帮助信息。

有关当程序在某一步停止运行时对变量求值的信息，参见对 Java 程序中的变量求值。

对 Java 程序中的变量求值

当程序在断点或步进处停止运行时，有两种方法来对变量求值:

- 在调试命令行上输入 `EVAL VariableName`。
- 将光标置于显示的源代码中的变量名上，并按 F11 键（显示变量）。

使用 EVAL 命令来对 Java(TM) 程序中的变量进行求值。

注意: 还可以使用 EVAL 命令来更改变量的内容。有关 EVAL 命令的变体的更多信息，参见 WebSphere Development Studio: ILE C/C++ Programmer's Guide（SC09-2712）一书  和联机帮助信息。

在查看 Java 程序中的变量时，请注意下列各项:

- 如果对作为 Java 类实例的变量求值，则屏幕的第一行显示的对象类型。还显示该对象的标识符。在屏幕的第一行后面，将显示对象中每个字段的内容。如果变量为空，则屏幕的第一行指示它为空。星号显示（空对象的）每个字段的内容。
- 如果对作为 Java 字符串对象的变量求值，则将显示该字符串的内容。如果该字符串为空，则显示空值。
- 您不能更改作为字符串的变量。
- 如果对作为数组的变量求值，则显示“ARR”，后跟该数组的标识符。您可以使用变量名的下标来对数组的元素求值。如果数组为空，则显示空值。
- 不能更改作为数组的变量。如果数组不是字符串或对象的数组，则可以更改该数组的元素。
- 对于数组变量，可以指定 arrayname.length 来查看数组中的元素数。
- 如果您想查看作为类字段的变量的内容，则可以指定 classvariable.fieldname。
- 如果尝试在初始化变量之前对该变量求值，则将发生以下两种情况之一。显示变量不可用于显示消息，或显示该变量的未经初始化的内容，这可能是一个奇怪的值。

调试 Java 和本机方法程序

您可以同时调试 Java(TM) 程序和本机方法程序。当在交互式屏幕上调试源时，可以调试服务程序（*SRVPGM）中使用 C 编程的本机方法。 » 必须使用调试数据来编译和创建 *SRVPGM。 «

要立即调试 Java 程序和本机方法程序:

1. 在显示 Java 程序源时，按 F14 键（使用模块列表），以显示“使用模块列表”（WRKMODLST）屏幕。
2. 选择选项 1（添加程序）以添加服务程序。
3. 选择选项 5（显示模块源）以显示您想调试的 *MODULE 和源。
4. 按 F6 键（添加／清除断点）以在服务程序中设置断点。有关设置断点的更多信息，参见设置断点。
5. 按 F12 键（继续执行）来运行程序。

注意: 当在服务程序中遇到断点时，程序将停止运行，并显示服务程序的源。

从另一个屏幕调试 Java 程序

在调试 Java(TM) 程序时，每当遇到断点时，就会显示程序源。这可能会干扰 Java 程序的显示输出。为避免发生这种情况，请从另一个屏幕调试 Java 程序。Java 程序的输出显示在运行 Java 命令的屏幕上，程序源显示在另一个屏幕上。

只要正在运行的 Java 程序不使用“及时”（JIT）编译器，也有可能以此方式对其进行调试。

要从另一个屏幕调试 Java，请执行下列操作:

1. 开始准备调试时，必须挂起 Java 程序。可以通过使 Java 程序进入以下状态来挂起该程序:
   - 等待键盘输入。
   - 等待一段时间间隔。
   - 循环测试变量，这需要设置一个最终能使 Java 程序跳出循环的值。
2. 在挂起 Java 程序之后，请转至另一屏幕来执行这些步骤:
   a. 在命令行上输入“使用活动作业”（WRKACTJOB）命令。
   b. 查找正在运行该 Java 程序的批处理立即（BCI）作业。在“子系统／作业”列表中查找 QJVACMDSRV。在“用户”列表中查找您的“用户标识”。在“类型”下面查找 BCI。
   c. 输入选项 5 以使用该作业。
   d. “使用作业”屏幕的顶部将显示“编号”、“用户”和“作业”。输入 `STRSRVJOB Number/User/Job`。
   e. 输入 `STRDBG CLASS(classname)`。Classname 是要调试的 Java 类的名称。它可以是您在 Java 命令上指定的类名称，也可以是另一个类。
   f. 该类的源出现在“显示模块源”屏幕中。
   g. 每当想要在该 Java 类中停止时，按 F6 键（添加／清除断点）来设置断点。按 F14 键来添加其它要调试的类、程序或服务程序。有关设置断点的更多信息，参见设置断点。
   h. 按 F12 键（继续执行）继续运行程序。
3. 停止挂起原先的 Java 程序。当遇到断点时，“显示模块源”屏幕出现在原来输入“启动服务作业”（STRSRVJOB）命令和“启动调试”（STRDBG）命令的屏幕上。当 Java 程序结束时，将出现消息被服务作业已结束。
4. 输入“结束调试”（ENDDBG）命令。
5. 输入“结束服务作业”（ENDSRVJOB）命令。

注意: 当在原始作业中启动 Java 虚拟机时，确保禁用“及时”（JIT）。可使用 `java.compiler=NONE` 属性做到这一点。如果进行调试时 JIT 正在运行中，则可能会发生意外的结果。

参见 QIBM_CHILD_JOB_SNDINQMSG 环境变量以了解有关此变量的更多信息，此变量控制在调用 Java 虚拟机之前 BCI 作业是否进行等待。

QIBM_CHILD_JOB_SNDINQMSG 环境变量

QIBM_CHILD_JOB_SNDINQMSG 环境变量是这样一个变量: 它控制在启动 Java(TM) 虚拟机之前，运行 Java 虚拟机的批处理立即（BCI）作业是否进行等待。

如果该环境变量设置为 1，则“运行 Java”（RUNJVA）命令运行时，将把一条消息发送到用户的消息队列。将在 BCI 作业中启动 Java 虚拟机之前发送该消息。该消息类似于:

```
Spawned (child) process 023173/JOB/QJVACMDSRV is stopped (G C)
```

要查看此消息，请输入 SYSREQ 并选择选项 4。

BCI 作业在您对此消息作出应答之前将一直等待。（G）应答将启动 Java 虚拟机。

在应答该消息之前，可在 BCI 作业调用的 *SRVPGM 或 *PGM 中设置断点。

注意: 不能在 Java 类中设置断点，因为此时 Java 虚拟机尚未启动。

调试通过定制类装入程序装入的 Java 类

» 要调试通过定制类装入程序装入的类，请执行下列步骤。

1. 将 DEBUGSOURCEPATH 环境变量设置为包含源代码的目录，或者，对于包限定的类这种情况，设置为包名称的起始目录。

   例如，如果定制类装入程序装入了位于目录 /MYDIR 之下的类，则执行下列操作:

   ```
   ADDENVVAR ENVVAR(DEBUGSOURCEPATH) VALUE('/MYDIR')
   ```

2. 从“显示模块源”屏幕将该类添加至调试视图。

   如果已将该类装入到 Java(TM) 虚拟机（JVM）中，则仅仅象平常那样添加 *CLASS 并显示源代码以进行调试。

   例如，要查看 pkg1/test14.class 的源代码，请输入:

   ```
   Opt      程序 / 模块          库            类型
   1        pkg1.test14_        *LIBL         *CLASS
   ```

   如果尚未将该类装入到 JVM 中，则执行相同的步骤来添加 *CLASS，如前所述。然后，将出现 **Java 类文件不可用**消息。此时，可继续程序处理。当类中的任何方法与输入的给定名称相匹配时，JVM 将自动停止。将显示类的源代码，并可进行调试。 «

调试 servlet

» 调试 servlet 是调试通过定制类装入程序装入的类的特殊情况。Servlet 在 IBM HTTP Server 的 Java(TM) 运行时中运行。一种调试 servlet 的方法是遵循用于通过定制类装入程序装入的类的指示信息。

另一种调试 servlet 的方法如下: «

1. 在 Qshell Interpreter 中使用 `javac -g` 命令来编译 servlet。
2. 将源代码（.java 文件）和编译后的代码（.class 文件）复制到 /QIBM/ProdData/Java400。
3. 对 .class 文件运行“创建 Java 程序”（CRTJVAPGM）命令，并使用优化级别 10，即 OPTIMIZE(10)。
4. 启动服务器。
5. 对要运行 servlet 的作业运行“启动服务作业”（STRSRVJOB）命令。
6. 输入 STRDBG CLASS(myServlet)，其中，myServlet 是 servlet 的名称。应该会显示源。
7. 在 servlet 中设置一个断点，并按 F12 键。
8. 运行 servlet。当 servlet 碰到断点时，您可以继续进行调试。

Java 平台调试器体系结构

» “Java(TM) 平台调试器体系结构”（JPDA）由三个部件组成:

- Java 虚拟机调试接口（JVMDI）
- Java 调试线协议（JDWP）
- Java 调试接口（JDI）

JDPA 的所有这三个部件都使调试器的任何使用 JDWP 的前端能够执行调试操作。调试器前端可以以远程方式运行，也可以作为 iSeries 应用程序运行。

Java 虚拟机调试接口

在 “Java(TM) 2 SDK（J2SDK）Standard Edition V1.2 或更高版本中， “Java 虚拟机调试接口”（JVMDI）是 Sun Microsystems 的平台应用程序接口（API）的一部分。JVMDI 允许任何人使用 iSeries C 代码来为 iSeries 服务器编写 Java 调试器。由于调试器使用 JVMDI 接口，所以它不需要知道 Java 虚拟机的内部结构。JVMDI 是 JPDA 中的最低级接口，它最接近 Java 虚拟机。

调试器与 Java 虚拟机在同一具有多线程能力的作业中运行。调试器使用 “Java 本机接口”（JNI）“调用” API 来创建 Java 虚拟机。然后，它将一个 hook 放在用户类 main 方法的开头，并调用 main 方法。当 main 方法开始时，就会遇到此 hook，于是开始调试。可以使用一些典型的调试功能，例如，设置断点、步进、显示变量和更改变量。

调试器处理正在运行 Java 虚拟机的作业与正在处理用户界面的作业之间的通信。此用户界面或者在 iSeries 服务器上，或者在另一个系统上。

QSYS 库中一个名为 QJVAJVMDI 的服务程序支持 JVMDI 功能。

Java 调试线协议

“Java 调试线协议”（JDWP）是调试器进程与 JVMDI 之间的已定义通信协议。可以从远程系统或通过本地套接字使用 JDWP。它从 JVMDI 中除去了一层，但却是更为复杂的接口。

在 QShell 中启动 JDWP

要启动 JDWP 并运行 Java 类 SomeClass，请在 QShell 中输入以下命令:

```
    java -interpret -Xrunjdwp:transport=dt_socket,
address=8000,server=y,suspend=n SomeClass
```

在本示例中，JDWP 在 TCP/IP 端口 8000 上侦听来自远程调试器的连接，但可使用您所要的任何端口号; dt_socket 是处理 JDWP 传输的 SRVPGM 的名称，不能更改。

有关可以与 -Xrunjdwp 配合使用的附加选项，参见 Sun Microsystems 的 Sun VM 调用选项 。

从 CL 命令行启动 JDWP

要将 -Xrun 选项与 CL 命令配合使用，可将 os400.xrun.option 属性定义成 QShell 命令行上已使用过的字符串。要启动 JDWP 并运行 Java 类 SomeClass，请输入以下命令:

```
JAVA CLASS(SomeClass) INTERPRET(*YES)
PROP((os400.xrun.option 'jdwp:transport=dt_socket,address=8000,
                                        server=y,suspend=n'))
```

许多 JVMDI 功能在优化级别 10 和 20 不能工作。因此，建议使用解释器来运行应用程序，这是因为所有功能都能配合它工作。

Java 调试接口

“Java 调试接口”（JDI）是为进行工具开发而提供的高级 Java 语言接口。JDI 将 JVMDI 和 JDWP 的复杂性隐藏到一些 Java 类定义背后。JDI 包含在 rt.jar 文件中，因此调试器的前端存在于任何安装有 Java 的平台上。

如果要为 Java 编写调试器，则应使用 JDI，因为它是最简单的接口，并且代码具有平台独立性。

有关 JDPA 的更多信息，参见 Sun Microsystems 的 Java 平台调试器体系结构概述 。

查找内存泄漏

» ANZJVM 通过使用由指定时间间隔分隔的两个垃圾收集堆副本来查找对象泄漏。要查找对象泄漏，您应查看堆中每个类的实例数。应该将具有异乎寻常高数目个实例的类记录为可能存在泄漏情况。

您还应记录两个垃圾收集堆副本之间每个类的实例数的变化情况。如果某个类的实例数持续增大，则应将该类记录为可能存在泄漏情况。两个副本之间的时间间隔越长，存在实际泄漏对象的情况就越确定。通过在采用较大时间间隔的情况下运行许多次 ANZJVM，您应该能够比较确定地诊断出正在泄漏哪些内容。 «

第 9 章 IBM Developer Kit for Java 故障诊断

如果在使用 IBM Developer Kit for Java(TM) 时遇到问题，则执行任何步骤来确定问题的根源。

- » 在使用 IBM Developer Kit for Java 时，您可能会注意到一些限制。本主题标识了任何已知的限制、约束或独特行为。«
- 查找运行过 Java 命令的作业的作业记录。并且，查找运行 Java 程序的批处理立即（BCI）作业所生成的作业记录以分析故障原因。
- 为授权程序分析报告（APAR）收集有用数据。
- 应用程序临时性修订（PTF）。
- 如果检测到 IBM Developer Kit for Java 中的潜在缺陷，则您应了解如何获取支持。

限制

当使用 IBM Developer Kit for Java(TM) 时，您可能会注意到在如何使用方面有一些限制。此列表标识了任何已知的限制、约束或独特行为。

- 当装入某个类但找不到其超类时，错误信息指示找不到原始类。例如，如果类 B 从类 A 扩展而来，在装入类 B 时找不到类 A，则错误信息指示找不到类 B，尽管实际上是找不到类 A。当您看到指示找不到某个类的错误时，请进行检查，以确保该类及其所有超类都在 CLASSPATH 中。这也适用于那些由正在装入的类实现的接口。
- » 垃圾收集堆的大小限制为 132 GB 以下。
- 结构化的对象数被限制在 13200 万个以下。 «
- java.net backlog 参数在 iSeries 服务器上的行为可能与其它平台不同。例如:
  - Listen backlogs 0, 1
    - Listen(0) 表示允许一个挂起连接；它不禁用套接字。
    - Listen(1) 表示允许一个挂起注解，含义和 Listen(0) 相同。
  - Listen backlogs > 1
    - 这允许将许多挂起请求保留在侦听队列上。如果有新连接请求到达，而队列请求数达到限制，则删除其中一个挂起的请求。
- » 无论使用的是什么 JDK 版本，都只能在具有多线程能力（即，具有线程安全性）的环境中使用 Java 虚拟机。iSeries 服务器具有线程安全性，但一些文件系统却并非如此。集成文件系统主题包含不具有线程安全性的文件系统的列表。«

查找作业记录以进行 Java 问题分析

使用运行过 Java(TM) 命令的作业所生成的作业记录以及运行过 Java 程序的批处理立即（BCI）作业的作业记录来分析 Java 故障的原因。这两个作业记录都可能包含重要的错误信息。

查找 BCI 作业的作业记录的方式有两种。可以查找记录在运行 Java 命令的作业的作业记录中的 BCI 作业名。然后，使用该作业名来查找 BCI 作业的作业记录。

也可以通过遵循下列步骤来查找 BCI 作业的作业记录:

1. 在 iSeries 命令行上输入“使用提交的作业”（WRKSBMJOB）命令。
2. 转至列表的底部。

3. 查找列表中的最后一个作业，它名为 QJVACMDSRV。
4. 对该作业输入选项 8（使用假脱机文件）。
5. 将显示一个名为 QPJOBLOG 的文件。
6. 按 F11 键来查看该假脱机文件的视图 2。
7. 验证日期和时间是否与故障发生时的日期和时间相匹配。

   注意: 如果日期和时间与您注销时的日期和时间不匹配，则请继续在已提交作业的列表中查找。尝试查找日期和时间与您注销时的日期和时间相匹配的 QJVACMDSRV 作业记录。

如果找不到 BCI 作业的作业记录，则可能是未生成这样的作业记录。如果将 QDFTJOBD 作业描述的 ENDSEP 值设置得太大，或 QDFTJOBD 作业描述的 LOG 值指定 *NOLIST，则会发生这种情况。检查这些值，并更改它们，以便为 BCI 作业生成作业记录。

要为运行过“运行 Java”（RUNJVA）命令的作业生成作业记录，请执行下列步骤:

1. 输入 SIGNOFF *LIST。
2. 然后再次注册。
3. 在 iSeries 命令行上输入“使用假脱机文件”（WRKSPLF）命令。
4. 转至列表的底部。
5. 查找名为 QPJOBLOG 的文件。
6. 按 F11 键。
7. 验证日期和时间是否与您输入注销命令时的日期和时间相匹配。

   注意: 如果日期和时间与您注销时的日期和时间不匹配，则请继续在已提交作业的列表中查找。尝试查找日期和时间与您注销时的日期和时间相匹配的 QJVACMDSRV 作业记录。

收集用于 Java 问题分析的数据

要收集用于授权程序分析报告（APAR）的数据，请执行下列步骤:

1. 包括问题的完整描述。
2. 保存在运行时导致问题的 Java(TM) 类文件。
3. 可以使用 SAV 命令来保存集成文件系统中的对象。您可能需要保存此程序必须运行的其它类文件。您也可能想要保存和发送整个目录，以供 IBM 在尝试再现问题（如果有必要的话）时使用。这是一个关于如何保存整个目录的示例。

   示例: 保存目录

   注意: 请阅读代码示例不保证声明以了解重要的法律信息。

   ```
   SAV DEV('/QSYS.LIB/TAP01.DEVD') OBJ(('/mydir'))
   ```

   如果有可能的话，保存问题所涉及的任何 Java 类的源文件。这对于 IBM 再现并分析问题很有帮助。
4. 保存任何包含运行程序所必需的本机方法的服务程序。
5. 保存任何运行 Java 程序所必需的数据文件。
6. 添加一个关于如何再现该问题的完整描述。这应该包括:
   - CLASSPATH 环境变量的值。
   - 关于运行的 Java 命令的描述。
   - 关于如何对程序所需的任何输入作出响应的描述。
7. 包括发生故障前后发生的任何垂直许可内码（VLIC）作业记录。

8. 添加来自运行 Java 虚拟机的交互式作业和 BCI 作业的作业记录。

获取 IBM Developer Kit for Java 的支持

IBM Developer Kit for Java(TM) 的支持服务是根据 iSeries 软件产品的一般条款和条件提供的。支持服务包括程序服务、语音支持和咨询服务。有关更多信息，请使用 IBM iSeries 主页 的“支持”主题下提供的联机信息。请使用“IBM 5722-JV1（IBM Developer Kit for Java）支持服务”。或者与当地 IBM 代表联系。

IBM 可能会要求您获取更新级别的 IBM Developer Kit for Java 才能接收“连续程序服务”。有关更多信息，参见支持多个 Java Development Kit（JDK）。

通过程序服务或语音支持，支持解决 IBM Developer Kit for Java 程序的缺陷。通过咨询服务，支持解决应用程序编程或调试问题。

通过咨询服务，支持 IBM Developer Kit for Java 应用程序接口（API）调用，除非出现下列情况:

1. 通过在相对简单的程序中重建来演示时，很明显是 Java API 缺陷。
2. 这是一个请求文档澄清的问题。
3. 这是一个关于样本或文档位置的问题。

通过咨询服务支持所有编程辅助。这包括 IBM Developer Kit for Java 许可程序（LP）产品中提供的程序样本。在不受支持的基础上，可从因特网上的 IBM iSeries主页 获得其它样本。

IBM Developer Kit for Java LP 提供了关于解决问题的信息。如果您相信 IBM Developer Kit for Java API 中有潜在的缺陷，则需要一个演示该错误的简单程序。

第 10 章 IBM Developer Kit for Java 的代码示例

以下是 IBM Developer Kit for Java(TM) 的代码示例的列表。

注意: 请阅读代码示例不保证声明以了解重要的法律信息。

CL 命令

- » ANZJVM «
- CHGJVAPGM
- CRTJVAPGM
- DLTJVAPGM
- DMPJVM
- DSPJVAPGM
- RUNJVA

国际化

- DateFormat
- NumberFormat
- ResourceBundle

JDBC

- » Access 属性
- Blob «
- CallableStatement 接口
- » 通过另一个语句的游标来使用语句更改值
- Clob
- 创建 UDBDataSource 并将其与 JNDI 绑定
- 创建 UDBDataSource 并获取用户标识和密码
- 创建 UDBDataSourceBind 并设置 DataSource 属性 «
- DatabaseMetaData 接口
- » 创建 UDBDataSource 并将其与 JNDI 绑定
- Datalink
- 单值类型 «
- 嵌入式 SQL 语句
- » 结束事务
- 无效的用户标识和密码
- JDBC
- 在一个事务中工作的多个连接
- 在绑定 UDBDataSource 之前获取初始上下文
- ParameterMetaData

- 通过另一个语句的游标来从表中除去值 «
- ResultSet 接口
- » ResultSet 灵敏度
- 灵敏和不灵敏 ResultSet
- 使用 UDBDataSource 和 UDBConnectionPoolDataSource 来设置连接池
- SQLException «
- » 暂挂和继续事务
- 暂挂的 ResultSet
- 测试连接池的性能
- 测试两个 DataSource 的性能
- 更新 BLOB
- 更新 CLOB
- 将一个连接与多个事务配合使用
- 使用 BLOB
- 使用 CLOB
- 使用 DB2CachedRowSet 属性和 DataSource
- 使用 DB2CachedRowSet 属性和 JDBC URL
- 使用 JTA 来处理事务
- 使用带有多个列的元数据 ResultSet
- 并行地使用本机 JDBC 和 Toolbox JDBC
- 使用 PreparedStatement 来获取 ResultSet
- 通过使用 execute(Connection) 方法来使用现有的数据库连接
- 使用 execute(int) 方法来将数据库请求收集到一起
- 使用 populate 方法
- 通过使用 setConnection(Connection) 方法来使用现有的数据库连接
- 使用 Statement 对象的 executeUpdate 方法«

Java 认证和授权服务

- JAAS HelloWorld 示例
- JAAS SampleThreadSubjectLogin 示例

» Java 一般安全性服务

- 样本非 JAAS 客户机程序
- 样本非 JAAS 服务器程序
- 样本启用 JAAS 的客户机程序
- 样本启用 JAAS 的服务器程序

» Java 命名和目录接口

  - 在目录中添加项
  - 删除目录中的项
  - 在目录中添加项

– 重命名目录项
– 指定二进制属性名列表«

» **Java 安全套接字扩展**

– 使用 SSLContext 对象的 SSL 客户机和服务器«

» **Java 与其它编程语言**

– 调用 CL 程序
– 调用 CL 命令
– 调用另一个 Java 程序
– 从 C 中调用 Java
– 从 RPG 中调用 Java
– 输入和输出流
– 调用 API
– Java 的 OS/400 PASE 本机方法
– 套接字

» **可选的包**

– JCE «

性能工具

– Java 性能数据转换器

» **运行不带 GUI 的主机**

– 设置 Remote AWT «

SQLJ

– Java 应用程序中的嵌入式 SQL 语句

安全套接字层

– 套接字生成器
– 服务器套接字生成器
– 安全套接字层
– 安全套接字层服务器

第 11 章 IBM Developer Kit for Java 参考

» 以下是 Developer Kit for Java(TM) 的参考资料:

Javadoc

- 特定于 iSeries 的 JAAS Javadoc
- JAAS API 规范

Java 2 Platform Standard Edition V1.3.1

- Java 2 Platform Standard Edition V1.3.2 API 规范
- Abstract Window Toolkit（AWT）
- Java IDL
- 输入方法框架
- 国际化
- JDBC API
- JNI — Java 本机接口
- Java 远程方法调用（RMI）
- RMI — 远程方法调用
- 安全性
- Java 2 SDK 工具«

代码不保证声明信息

本文档包含编程示例。

IBM 授予您使用所有编程代码示例的非专有版权许可证，您可以由此生成相似的定制功能以满足您特定的需要。

IBM 提供所有样本代码只是出于解释的目的。并未在所有环境下完全测试这些示例。因此，IBM 不保证或默示这些程序的可靠性、可服务性和功能。

本文档中包含的所有程序是以“按现状”的基础提供的，不附有任何形式的保证。明示的不保证声明包括非侵权性、适销性和适用于某特定用途的默示保证。

中国印刷