SYBASE®

程序员参考

jConnect™ for JDBC™

6.05

文档 ID：DC38606-01-0605-01

最后修订日期：2005 年 9 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目录

关于本手册 ..... **vii**

第 1 章 **简介** **1**
- 什么是 JDBC？ 1
- 什么是 jConnect？ 2

第 2 章 **编程信息** **5**
- 设置 jConnect 5
  - 设置 jConnect 版本 5
  - 调用 jConnect 驱动程序 11
- 建立连接 12
  - 设置连接属性 12
  - 连接到 Adaptive Server 23
  - 使用 JNDI 连接到服务器 25
- 实现自定义套接字插件 31
  - 创建和配置自定义套接字 32
- 处理国际化和本地化 35
  - 使用 jConnect 传递 Unicode 数据 35
  - Connect 字符集转换程序 36
- 处理数据库 42
  - 实现高可用性故障切换支持 42
  - 执行服务器到服务器的远程过程调用 47
  - 使用 ASE 12.5 和更高版本的宽表支持 48
  - 访问数据库元数据 49
  - 对结果集使用游标 50
  - 批处理更新支持 61
  - 通过存储过程的结果集更新数据库 63
  - 使用数据类型 64
- 实现高级功能 70
  - 使用事件通知 70
  - 处理错误消息 73
  - 将 Java 对象作为列数据存储在表中 77
  - 使用动态类装载 81

JDBC 2.0 可选软件包扩展支持 85
对 JDBC 标准的限制与解释 95
使用 JDBC 3.0 方法占位模块 95
使用 Connection.isClosed 和 IS_CLOSED_TEST 96
对未处理的结果使用 Statement.close 97
调整多线程 98
使用 ResultSet.getCursorName 98
使用具有较大参数值的 setLong 98
支持的新数据类型 99
使用 COMPUTE 语句 100
执行存储过程 100

第 3 章 安全性 103
概述 103
限制 104
SSL 104
Kerberos 104
为 Kerberos 配置 jConnect 应用程序 105
GSSMANAGER_CLASS 连接属性 106
设置 Kerberos 环境 108
示例应用程序 111
互操作性 113
故障排除 114
相关文档 115

第 4 章 故障排除 117
使用 jConnect 进行调试 117
获取 Debug 类的一个实例 118
在应用程序中打开调试程序 118
在应用程序中关闭调试程序 119
为调试程序设置 CLASSPATH 119
使用 Debug 方法 119
捕获 TDS 通信 121
PROTOCOL_CAPTURE 连接属性 121
Capture 类中的 pause 和 resume 方法 122
解决连接错误 122
网关连接被拒绝 122
无法连接到 4.9.2 SQL Server 123
管理 jConnect 应用程序所使用的内存 124
解决存储过程错误 125
RPC 返回比已注册参数更少的输出参数 125
在返回输出参数时出现读取 / 状态错误 125

在非链式事务模式中执行存储过程 125
解决自定义套接字执行错误 126

第 5 章 性能和调优 127
改善 jConnect 性能 127
BigDecimal 范围重设 128
REPEAT_READ 连接属性 128
SunIoConverter 字符集转换 129
对动态 SQL 中的预准备语句的性能调优 129
选择预准备语句和存储过程 130
可移植应用程序中的预准备语句 131
具有 jConnect 扩展的预准备语句 131
Connection.prepareStatement 133
DYNAMIC_PREPARE 连接属性 133
SybConnection.prepareStatement 134
ESCAPE_PROCESSING_DEFAULT 连接属性 135
游标性能 136
LANGUAGE_CURSOR 连接属性 136

第 6 章 迁移 jConnect 应用程序 137
向 jConnect 6.x 迁移应用程序 137
更改 Sybase 扩展 138
扩展更改示例 139
方法名称 139
Debug 类 140

第 7 章 Web 服务器网关 141
关于 Web 服务器网关 141
使用 TDS 贯通 142
配置 jConnect 和网关 142
使用要求 147
阅读 index.html 文件 147
运行示例 Isql 小程序 148
使用 TDS 贯通服务器小程序 149
检查要求 151
安装服务器小程序 151
调用服务器小程序 152
跟踪活动的 TDS 会话 153
恢复 TDS 会话 153
在 Solaris 上使用 TDS 和 Netscape Enterprise Server 3.5.1 153

附录 A SQL 例外与警告消息 155

附录 B **jConnect 示例程序 173**
运行 IsqlApp 173
运行 jConnect 示例程序和代码 175
示例应用程序 175
示例代码 177

索引 179

关于本手册

*Sybase jConnect for JDBC 程序员参考*介绍 jConnect™ for JDBC™ 产品，并说明如何用它访问存储在关系数据库管理系统中的数据。

读者

本手册适用于熟悉 Java 编程语言、JDBC 和 Transact-SQL®（Sybase® 版本的结构化查询语言 (SQL)）的数据库应用程序程序员。

相关文档

以下文档可能会有所帮助：

- *Sybase jConnect for JDBC 安装指南*
- *Sybase jConnect for JDBC 发行公告*
- JDBC 的 jConnect 扩展的 javadoc 文挡。Java Software 的 Java 开发工具包 (JDK) 包含用于从源代码文件中提取注释的 javadoc 脚本。此脚本已用于从 jConnect 源文件中提取 jConnect 软件包、类和方法的文档。使用完全安装或 javadocs 选项安装 jConnect 时， javadoc 信息位于 *javadocs* 目录中：

  Installation_directory/docs/en/javadocs

其它信息来源

使用 Sybase Getting Started CD、 SyBooks™ CD 和 Sybase Product Manuals Web 站点可以了解有关产品的更多信息：

- Getting Started CD 包含 PDF 格式的发行公告和安装指南，也可能包含 SyBooks CD 中未收纳的其它文档或更新信息。它随软件一起提供。若要阅读或打印 Getting Started CD 上的文档，需要使用 Adobe Acrobat Reader，该软件可以通过 CD 上提供的链接从 Adobe Web 站点免费下载。
- SyBooks CD 含有产品手册，随软件提供。 Eclipse-based SyBooks 浏览器使您能够以简单易用的基于 HTML 的格式阅读手册。

  有些文档可能是以 PDF 格式提供的，您可以通过 SyBooks CD 上的 PDF 目录访问这些文档。要阅读或打印 PDF 文件，需要使用 Adobe Acrobat Reader。

  有关安装和启动 SyBooks 的说明，请参见 Getting Started CD 上的 *SyBooks 安装指南* 或 SyBooks CD 上的 *README.txt* 文件。

- Sybase Product Manuals Web 站点是 SyBooks CD 的联机版本，您可以使用标准 Web 浏览器进行访问。除了产品手册之外，还可以找到有关 EBFs/Maintenance （EBF/ 维护）、 Technical Documents （技术文档）、 Case Management （案例管理）、 Solved Cases （解决的案例）、 Newsgroups（新闻组）和 Sybase Developer Network（Sybase 开发员网络）的链接。

  若要访问 Sybase Product Manuals Web 站点，请转到位于 http://www.sybase.com/support/manuals/ 上的 Product Manuals（产品手册）。

Web 上的 Sybase 认证

Sybase Web 站点上的技术文档不断在更新。

❖ **查找有关产品认证的最新信息**

1 将 Web 浏览器定位到位于 http://www.sybase.com/support/techdocs/ 上的 Technical Documents （技术文档）。

2 从左边的导航栏中选择 “Products”（产品）。

3 从产品列表中选择一种产品名称并单击 “Go”（查找）。

4 选择 “Certification Report”（认证报告）过滤器，指定时间范围，然后单击 “Go”（查找）。

5 单击 “Certification Report”（认证报告）标题显示此报告。

❖ **查找组件认证的最新信息**

1 将 Web 浏览器定位到位于 http://certification.sybase.com/ 的可用性和认证报告。

2 在 “按产品搜索” (Search by Products) 下选择产品系列和产品；或在 “按平台搜索” (Search by Platform) 下选择平台和产品。

3 选择 “搜索” (Search) 以显示所选项目的可用性和认证报告。

❖ **创建 Sybase Web 站点 （包括支持页）的个人化视图**

建立 MySybase 配置文件。 MySybase 是一项免费服务，它允许您创建 Sybase Web 页的个人化视图。

1 将 Web 浏览器定位到位于 http://www.sybase.com/support/techdocs/ 上的 Technical Documents （技术文档）。

2 单击 “MySybase”（我的 Sybase）并创建 MySybase 配置文件。

Sybase EBF 和软件维护

❖ **查找有关 EBF 和软件维护的最新信息**

1 将 Web 浏览器定位到位于 http://www.sybase.com/support 上的 Sybase 支持页。

2 选择 “EBFs/Maintenance”（EBF/ 维护）。如果出现提示信息，请输入您的 MySybase 用户名和口令。

3 选择一个产品。

4 指定时间范围并单击 “Go”（查找）。即会显示一列 EBF/ 维护版本。

锁形图标表示因为您没有注册为 “Technical Support Contact”（技术支持联系人），因此您没有某些 EBF/ 维护版本的下载授权。如果您未注册，但拥有 Sybase 代表或支持合同提供的有效信息，请单击 “编辑角色”(Edit Roles)，在 MySybase 配置文件中添加 “技术支持联系人”角色。

5 单击信息图标可显示 EBF/ 维护报告，单击产品说明可下载软件。

约定

本手册使用以下字体和语法约定：

- 在段落文本内以 Helvetica 来显示类、接口、方法和软件包。例如：

  SybConnection 类

  SybEventHandler 接口

  setBinaryStream 方法

  com.sybase.jdbcx 软件包

- 以斜体来显示对象、实例和参数名称。例如：

  “在以下示例中， *ctx* 是一个 DirContext 对象。”

  “eventHdler 是您实现的 SybEventHandler 类的实例。”

  “*classes* 参数是一个字符串，它列出您要调试的特定类。”

- 以等宽字体来显示代码段。以斜体显示代码段中的变量 （即代表填入的值的单词）。例如：

```
Connection con = DriverManager.getConnection("jdbc:
    sybase:Tds:host:port", props);
```

可访问性功能

此文档提供具有专门可访问性的 HTML 版本形式。可以利用适应性技术 （如屏幕阅读器）浏览 HTML，也可以用屏幕放大器查看。

<*jConnect for JDBC*> 和 HTML 文档已进行了测试，符合美国政府 “第 508 节可访问性”的要求。符合 “第 508 节”的文档一般也符合非美国的可访问性原则，如针对 Web 站点的 World Wide Web 协会 (W3C) 原则。

此产品的联机帮助也是以 HTM 格式提供的，您可以使用屏幕阅读器进行浏览。

注释 您可能需要对可访问性工具进行配置以实现最优化。某些屏幕阅读器按照大小写来辨别文本，例如将 “ALL UPPERCASE TEXT”看作首字母，而将 “MixedCase Text”看作单词。对工具进行配置，规定语法约定，您可能会感觉更方便。有关工具的信息，请查阅文档。

有关 Sybase 如何支持可访问性的信息，请参见位于 http://www.sybase.com/accessibility 的 Sybase Accessibility。 Sybase Accessibility 站点包括到有关 “第 508 节”和 W3C 标准的信息的链接。

如果需要帮助

对于购买了支持合同的客户安装的每一个 Sybase 产品，都会有一位或多位指定人员获得与 Sybase 技术支持部门联系的授权。如果使用手册或联机帮助不能解决问题，可让指定人员与 Sybase 技术支持部门联系或与所在区域的 Sybase 子公司联系。

第 1 章

简介

本章介绍了 jConnect for JDBC 产品及其概念和组件。

本章包括以下内容：

主题	页码
什么是 JDBC？	1
什么是 jConnect？	2

什么是 JDBC？

JDBC （ Java 数据库连接）由 Sun MicroSystems, Inc 公司 Java 软件部门开发，是关于应用程序编程接口 (API) 的规范，该规范允许 Java 应用程序使用结构化查询语言 (SQL) 访问多个数据库管理系统。 JDBC 驱动程序管理器可以处理多个连接到不同数据库的驱动程序。

标准 JDBC API 中包含一个接口集，利用这些接口可以打开到数据库的连接，执行 SQL 命令并处理结果。表 1-1 中介绍了这些接口。

表 1-1：JDBC 接口

接口	说明
java.sql.Driver	定位数据库 URL 的驱动程序
java.sql.Connection	用于连接到特定数据库
java.sql.Statement	执行 SQL 语句
java.sql.PreparedStatement	处理带参数的 SQL 语句
java.sql.CallableStatement	处理数据库存储过程调用
java.sql.ResultSet	获取 SQL 语句的结果
java.sql.DatabaseMetaData	用于访问关于数据库的连接的信息。
java.sql.ResultSetMetaData	用于访问有关 ResultSet 属性描述的信息。

每个关系数据库管理系统都需要一个驱动程序来实现这些接口。有四种类型的 JDBC 驱动程序：

- *类型 1 JDBC-ODBC 桥* — 将 JDBC 调用转换成 ODBC 调用，然后将其传递给 ODBC 驱动程序。有些 ODBC 软件必须驻留在客户机中。有些客户端数据库代码也可驻留在客户机中。
- *类型 2 native-API partly-Java 驱动程序* — 将 JDBC 调用转换成数据库特定的调用。该驱动程序能够直接与数据库服务器通信，还需要客户机中的一些二进制代码。
- *类型 3 net-protocol all-Java 驱动程序* — 利用独立于 DBMS 的网络协议与中间层服务器进行通信。然后，中间层网关将请求转换成供应商特定的协议。
- *类型 4 native-protocol all-Java 驱动程序* — 将 JDBC 调用转换成供应商特定的 DBMS 协议，允许客户端应用程序直接与数据库服务器进行通信。

什么是 jConnect？

jConnect 是 Sybase 高性能 JDBC 驱动程序。 jConnect 包含以下两种驱动程序：

- 三层环境下的 net-protocol/all-Java 驱动程序，以及
- 两层环境下的 native-protocol/all-Java 驱动程序。

jConnect 使用的协议是 TDS 5.0（Tabular Data Stream™，版本 5），该协议是 Adaptive Server® 和 Open Server™ 应用程序的本机协议。 jConnect 执行 JDBC 标准，为所有的 Sybase 系列产品提供最佳的连接性，允许访问 25 个以上的企业级系统和遗留系统，包括：

- Adaptive Server Enterprise
- Adaptive Server Anywhere™
- Adaptive Server IQ （以前为 Sybase IQ™）
- Replication Server®
- DirectConnect™

注释　由于将 Sybase SQL Server™ 的名称更改为了 Adaptive Server Enterprise，因此， Sybase 可以使用 Adaptive Server 和 Adaptive Server Enterprise 这两个名称泛指所有受支持的 Sybase SQL Server 和 Adaptive Server Enterprise 版本。在本文档以下部分中 Adaptive Server Enterprise 即指 Adaptive Server。

此外，jConnect for JDBC 能够访问 Oracle，AS/400 以及其它使用 Sybase DirectConnect 的数据源。

在某些情况下， JDBC 的 jConnect 实现会偏离 JDBC 1.x， JDBC 2.x 和 JDBC 3.x 规范。有关详细信息，请参见第 95 页的 “对 JDBC 标准的限制与解释”。

第 2 章

编程信息

本章描述构成 jConnect for JDBC 的基本组件和编程要求。介绍了如何调用 jConnect 驱动程序、设置连接属性和连接到数据库服务器。还包含有关使用 jConnect 功能的信息。

注释 有关 JDBC 编程的信息，请访问 http://java.sun.com/jdbc。

本章包含下列主题：

主题	页码
设置 jConnect	5
建立连接	12
实现自定义套接字插件	31
处理国际化和本地化	35
处理数据库	42
实现高级功能	70
对 JDBC 标准的限制与解释	95

设置 jConnect

本节描述在使用 jConnect 之前需要执行的任务。

设置 jConnect 版本

jConnect 有多个版本。使用版本设置可确定：

- LANGUAGE 连接属性的缺省值
- 特定版本的可用功能

- 缺省字符集 （如果没有使用 CHARSET 连接属性指定字符集）
- CHARSET_CONVERTER 连接属性的缺省值
- CANCEL_ALL 连接属性的缺省值，该连接属性用于设置 Statement.cancel 的行为，该命令在缺省情况下将取消对其调用该命令的对象和任何其它已开始执行并且正在等待结果的 Statement 对象
- 是否要从服务器请求宽表支持
- 是否要请求服务器支持以在 unichar (Unicode) 列中存储字符数据

  注释 只有 Adaptive Server 12.5 和更高版本支持宽表和 unichar 字符数据。

- 是否要从服务器请求对 date 和 time SQL 数据类型的支持

  注释 只有 Adaptive Server 12.5.1 和更高版本支持 date 和 time SQL 数据类型。

表 2-1 列出了可用的版本设置及其功能。

表 2-1：jConnect 版本设置及其功能

JCONNECT_ VERSION	功能	注释
“6.05”	• jConnect 从服务器请求对 date 和 time SQL 数据类型的支持。除 Adaptive Server 12.5.1 及更高版本以外的服务器将忽略此请求。 • jConnect 从服务器请求对 unichar 和 univarchar 数据类型的支持。除 Adaptive Server 12.5 及更高版本以外的服务器将忽略此请求。 • LANGUAGE 连接属性的缺省值是 null。 • 缺省情况下， Statement.cancel 只取消对其调用该命令的 Statement 对象。 • 计算列 — jConnect 驱动程序支持访问这些特定的计算列，包括其元数据信息。 • 长标识符 — jConnect 驱动程序完全支持应用程序中的长标识符。定义对象名称或标识符的长度的长标识符限制增加到了 255 字节。新的长标识符适用于大多数用户定义的标识符，包括表名、列名和索引名。这是长标识符限制放宽的结果。	对于 jConnect 6.05 版，缺省值为“6.05”。 有关 date 和 time 数据类型的详细信息，请参见第 68 页的 “使用 date 和 time 数据类型”。 有关 unichar 和 univarchar 数据类型和 Unicode 的详细信息，请参见第 35 页的 “使用 jConnect 传递 Unicode 数据”。 有关宽表的详细信息，请参见第 48 页的 “使用 ASE 12.5 和更高版本的宽表支持”。

JCONNECT_VERSION	功能	注释
“6”	• jConnect 从服务器请求对 date 和 time SQL 数据类型的支持。除 Adaptive Server 12.5.1 及更高版本以外的服务器将忽略此请求。 • jConnect 从服务器请求对 unichar 和 univarchar 数据类型的支持。除 Adaptive Server 12.5 及更高版本以外的服务器将忽略此请求。 • jConnect 从服务器请求对宽表的支持。除 Adaptive Server 12.5 及更高版本以外的服务器将忽略此请求。 • LANGUAGE 连接属性的缺省值是 null。 • 如果 CHARSET 连接属性没有指定字符集， jConnect 将使用数据库的缺省字符集。 CHARSET_CONVERTER 的缺省值为 CheckPureConverter 类。DISABLE_UNICHAR_SENDING 的缺省值设置为 false。 • 缺省情况下， Statement.cancel 只取消对其调用该命令的 Statement 对象。 • JDBC 2.0 方法可用于将 Java 对象作为列数据存储和检索。	对于 jConnect 6.0，缺省值是 VERSION_6。 有关 date 和 time 数据类型的详细信息，请参见第 68 页的“使用 date 和 time 数据类型”。 有关 unichar 和 univarchar 数据类型和 Unicode 的详细信息，请参见第 35 页的“使用 jConnect 传递 Unicode 数据”。 有关宽表的详细信息，请参见第 48 页的“使用 ASE 12.5 和更高版本的宽表支持”。
“5”	• LANGUAGE 连接属性的缺省值是 null。 • 如果 CHARSET 连接属性没有指定字符集， jConnect 将使用数据库的缺省字符集。 CHARSET_CONVERTER 的缺省值为 CheckPureConverter 类。 • 缺省情况下， Statement.cancel 只取消对其调用该命令的 Statement 对象。 • JDBC 2.0 方法可用于将 Java 对象作为列数据存储和检索。	对于 jConnect 5.x，缺省值是 VERSION_5。

JCONNECT_ VERSION	功能	注释
“4”	• LANGUAGE 连接属性的缺省值是 null。 • 如果 CHARSET 连接属性没有指定字符集， jConnect 将使用数据库的缺省字符集。 CHARSET_CONVERTER 的缺省值为 CheckPureConverter 类。 • 缺省情况下， Statement.cancel 只取消对其调用该命令的 Statement 对象。 • JDBC 2.0 方法可用于将 Java 对象作为列数据存储和检索。	服务器消息会按照您本地环境的语言设置进行本地化。支持的语言包括：中文、美国英语、法语、德语、日语、韩语、波兰语、葡萄牙语和西班牙语。 Statement.cancel 的缺省行为符合 JDBC。 请使用 CANCEL_ALL 设置 Statement.cancel 的行为。请参见第 11 页的 “CANCEL_ALL 连接属性”。 有关作为列数据的 Java 对象的详细信息，请参见第 77 页的 “将 Java 对象作为列数据存储在表中”。
“3”	• LANGUAGE 连接属性的缺省值是 us_english。 • 如果 CHARSET 连接属性没有指定字符集，jConnect 将使用数据库的缺省字符集。 • CHARSET_CONVERTER 的缺省值为 CheckPureConverter 类。 • 缺省情况下， Statement.cancel 会取消对其调用该命令的对象和任何其它已开始执行并且正在等待结果的 Statement 对象。	请参见 VERSION_2 的注释。
“2”	• LANGUAGE 连接属性的缺省值是 us_english。 • 如果 CHARSET 连接属性没有指定字符集，缺省字符集将是 iso_1。 • CHARSET_CONVERTER 的缺省值为 TruncationConverter 类，除非 CHARSET 连接属性指定多字节或 8 位字符集 （在这种情况下， CHARSET_CONVERTER 的缺省值为 CheckPureConverter 类）。 • 缺省情况下， Statement.cancel 会取消对其调用该命令的对象和任何其它已开始执行并且正在等待结果的 Statement 对象。	LANGUAGE 连接属性决定来自 jConnect 和服务器的消息的显示语言。 有关 CHARSET 和 CHARSET_CONVERTER 连接类的详细信息，请参见第 36 页的 “Connect 字符集转换程序”。 Statement.cancel 的 VERSION_2 缺省行为不符合 JDBC。请使用 CANCEL_ALL 设置 Statement.cancel 的行为。请参见第 11 页的 “CANCEL_ALL 连接属性”。

JCONNECT_VERSION 连接属性

如本节所述，您可以使用 JCONNECT_VERSION 来替换 SybDriver 版本设置并为特定连接指定不同的版本设置。

使用 JCONNECT_VERSION 可以为特定会话指定版本设置。您可以根据您需要的特性将 JCONNECT_VERSION 的值设置为 “2”、“3”、“4”、“5”、“6” 或 “6.05”。

在引用版本常量时，请使用此语法：

```
com.sybase.jdbcx.SybDriver.VERSION_6
```

虽然首选方法是使用 JCONNECT_VERSION 连接属性，但您也可以使用 SybDriver 中的常量值。

请使用 SybDriver.setVersion 设置 jConnect 版本。 setVersion 方法影响所有由 SybDriver 对象创建的连接的 jConnect 缺省行为。但是，可以使用 JCONNECT_VERSION 连接属性设置单个连接的版本特有行为。以下代码示例显示了如何装载 jConnect 驱动程序和设置版本：

```
import com.sybase.jdbcx.SybDriver;
SybDriver sybDriver = (SybDriver)
   Class.forName
  ("com.sybase.jdbc3.jdbc.SybDriver").newInstance();
sybDriver.setVersion
   (com.sybase.jdbcx.SybDriver.VERSION_6);
DriverManager.registerDriver(sybDriver);
```

可以多次调用 setVersion 来更改版本设置。新连接会继承在建立连接时与版本设置相关联的行为。在会话期间更改版本设置不会影响当前连接。 jConnect 提供一个 com.sybase.jdbcx.SybDriver.VERSION_LATEST 常量，该常量可用于确保总是请求所使用的 jConnect 驱动程序的最高版本值。但是，将版本设置为 com.sybase.jdbcx.SybDriver.VERSION_LATEST 后，如果用较新的 jConnect 驱动程序替换当前的 jConnect 驱动程序，可能会发现行为发生变化。

CANCEL_ALL 连接属性

CANCEL_ALL 是布尔值连接属性，用于指定 Statement.cancel 方法的行为。

注释　在 jConnect 4.0 和更低版本中， CANCEL_ALL 的缺省值是“true”。在 jConnect 4.1 版和更高版本中，为符合 JDBC 规范，如果您将连接属性 JCONNECT_VERSION 设置为 “4”或更高版本，则 CANCEL_ALL 的缺省设置为 “false”。

CANCEL_ALL 的设置对 Statement.cancel( ) 有以下影响：

- 如果 CANCEL_ALL 为 “false”，调用 Statement.cancel 将只取消对其调用该命令的 Statement 对象。因此，如果 stmtA 是一个 Statement 对象，则 stmtA.cancel 会取消 stmtA 中所包含的 SQL 语句在数据库中的执行，但其它语句不受影响。不管 stmtA 是正处于高速缓存中等待执行还是已经开始执行并正在等待结果，它都会被取消。
- 如果 CANCEL_ALL 为 “true”，调用 Statement.cancel 将不仅取消对其调用该命令的对象，还将取消同一连接上已开始执行并且正在等待结果的任何其它 Statement 对象。

以下示例将 CANCEL_ALL 设置为“false”。在该示例中，*props* 是用于指定连接属性的 Properties 对象：

```
props.put("CANCEL_ALL", "false");
```

注释　若要取消执行一个连接上的所有 Statement 对象，而不管其是否已开始在服务器上执行，请使用扩展方法 SybConnection.cancel。

调用 jConnect 驱动程序

若要注册并调用 Sybase jConnect 驱动程序，请使用以下两种建议方法之一：

- 按如下方式使用 Class.forName：

```
Class.forName("com.sybase.jdbc3.jdbc.SybDriver")
.newInstance();
```

- 向 jdbc.drivers 系统属性添加 jConnect 驱动程序。在初始化阶段，DriverManager 类试图装载 jdbc.drivers 中列出的驱动程序。这不如 Class.forName 调用方法的效率高。可以在此属性中列出多个驱动程序，程序之间用冒号 (:) 分隔。以下代码示例显示了如何在程序内向 jdbc.drivers 添加驱动程序：

```
Properties sysProps = System.getProperties();
String drivers = "com.sybase.jdbc3.jdbc.SybDriver";
String oldDrivers =
sysProps.getProperty("jdbc.drivers");
if (oldDrivers != null)
   drivers += ":" + oldDrivers;
 sysProps.put("jdbc.drivers", drivers.toString());
```

注释 System.getProperties 不能在 Java 小程序中使用。请改用 Class.forName 方法。

建立连接

本节介绍描述如何使用 jConnect 建立到 Adaptive Server 或 Adaptive Server Anywhere 数据库的连接。

设置连接属性

表 2-2 列出了 jConnect 的连接属性并给出了其缺省值。*必须在建立连接前设置连接属性。*

有两种设置驱动程序连接属性的方法：

- 在应用程序中使用 DriverManager.getConnection 方法。
- 在定义 URL 时设置连接属性。

注释 在 URL 中设置的驱动程序连接属性不会替换任何在应用程序中使用 DriverManager.getConnection 方法设置的相应的连接属性。

若要获得任何驱动程序的当前属性列表，请使用 Driver.getDriverPropertyInfo(String url, Properties props)，该方法返回 DriverPropertyInfo 对象的数组。该数组列出：

- 驱动程序属性
- 驱动程序属性所基于的当前设置
- URL 和传入的 props

驱动程序连接属性名称不区分大小写 （jConnect 使用 String.equalsIgnoreCase(String) 方法比较属性名称）。

表 2-2：连接属性

属性	说明	缺省值
APPLICATIONNAME	指定一个应用程序名称。它是一个用户定义的属性。可以编程使服务器端解释为此属性提供的值。	Null
BE_AS_JDBC_COMPLIANT_AS_POSSIBLE	调整其它属性以确保 jConnect 方法的应答方式尽可能符合 JDBC 3.0 标准。 如果将该属性设置为 “true”，将影响 （并替换）下列属性： • CANCEL_ALL （设置为 “false”） • LANGUAGE CURSOR （设置为 “false”） • SELECT_OPENS_CURSOR （设置为 “true”） • FAKE_METADATA （设置为 “true”） • GET_BY_NAME_USES_COLUMN_LABEL （设置为 “false”）	False
CACHE_COLUMN_METADATA	为了在将 DYNAMIC_PREPARE 设置为 “true” 时提高性能，jConnect 会在连续执行时对 ResultSet Metadata 进行高速缓存。	无
CANCEL_ALL	确定 Statement.cancel 方法的行为。请参见 第 11 页的 “CANCEL_ALL 连接属性”。	取决于版本设置。请参见 第 5 页的 “设置 jConnect 版本”。

属性	说明	缺省值
CAPABILITY_TIME	仅在 JCONNECT_VERSION >= 6 时使用。此时 jConnect 与支持 TIME 数据类型的服务器相连，并且所有类型为 java.sql.Time 的参数或**转义文字** {t ...} 都将作为 TIME 进行处理。 jConnect 的以前版本将这些参数作为 DATETIME 进行处理并在 java.sql.Time 参数前加上 “1970-01-01”。如果基础数据类型为 datetime 或 smalldatetime，则日期部分也会被存储在数据库中。在 jConnect 的新版本中，在处理 TIME 时，服务器会将时间转换成基础数据类型并会在前面加上其自己的基年。这可能导致旧数据与新数据之间不兼容。如果您要对 java.sql.Time 使用 datetime 或 smalldatetime 数据类型，为了能够向后兼容，您应将 CAPABILITY_TIME 保持为 *false*。将此属性保持为 *false* 会强制 jConnect 将 java.sql.Time 参数或**转义文字** {t ...} 作为 DATETIME 进行处理，无论服务器在处理 TIME 数据类型方面的能力如何。 将此属性设置为 *true* 会使 jConnect 在连接到 ASE 12.5.1 或更高版本时将 java.sql.Time 参数作为 TIME 数据类型进行处理。如果您要使用 smalldatetime 或 datetime 列来存储时间值， Sybase 建议您将此属性保持为 *false*。	False
CAPABILITY_WIDETABLE	如果您为提高性能而不使用 JDBC ResultSetMetaData （如列名），则可以将此属性设置为 “false”。这样将减少通过网络交换的数据，提高性能。除非要使用 EAServer，否则 Sybase 建议您使用缺省设置。请参见第 48 页的 “使用 ASE 12.5 和更高版本的宽表支持”。	False

属性	说明	缺省值
CHARSET	为传递给数据库的字符串指定字符集。如果 CHARSET 值为 Null，jConnect 将使用服务器的缺省字符集向服务器发送 string 数据。如果指定 CHARSET，数据库必须能够处理此格式的字符。如果数据库不能处理此格式的字符，将生成一条消息，说明不能正确完成字符转换。 **注释**　如果在使用 jConnect 6.05 时将 DISABLE_UNICHAR_SENDING 设置为“false”，那么当客户端试图向服务器发送无法用连接所使用的字符集表示的字符时，jConnect 将能够检测出来。发生这种情况时，jConnect 会将字符数据作为 unichar 数据发送给服务器，这样可使客户端能够在 unichar/univarchar 列和参数中插入 Unicode 数据。	Null
CHARSET_CONVERTER_CLASS	指定希望 jConnect 使用的字符集转换程序类。 jConnect 使用来自 SybDriver.setVersion 的版本设置或随 JCONNECT_VERSION 属性传入的版本来确定要使用的缺省字符集转换程序类。有关详细信息，请参见第 37 页的 “选择字符集转换程序”。	视版本而定
CLASS_LOADER	该属性设置为您创建的 DynamicClassLoader 对象。 DynamicClassLoader 用于装载在应用程序启动时存储在数据库中但不在 CLASSPATH 中的 Java 类。有关详细信息，请参见第 81 页的 “使用动态类装载”。	Null
CONNECTION_FAILOVER	与 Java 命名和目录接口 (JNDI) 一起使用。请参见第 29 页的 “CONNECTION_FAILOVER 连接属性”。	True

属性	说明	缺省值
DISABLE_UNICHAR_SENDING	当客户端应用程序向服务器发送 unichar 字符时 （与非 unichar 字符一起），会对发送到数据库的任何字符数据产生轻微的性能影响。在 jConnect 6.05 中，此属性缺省为“false”。使用 jConnect 较早版本的客户端如果希望向数据库发送 unichar 数据，必须将此属性设置为 “false”。请参见第 35 页的 “使用 jConnect 传递 Unicode 数据”。	视版本而定
DISABLE_UNPROCESSED_PARAM_WARNINGS	禁用警告。在处理存储过程的结果时，jConnect 经常读取行数据之外的返回值。如果不处理返回值， jConnect 将引发一个警告。若要禁用这些警告 （这样有助于提高性能），请将此属性设置为 “true”。	False
DYNAMIC_PREPARE	决定是否在数据库中预编译动态 SQL 准备语句。请参见第 133 页的 “DYNAMIC_PREPARE 连接属性”。	False
ENABLE_SERVER_PACKETSIZE	允许您指定是否要使用服务器指定的包大小。缺省情况下，此属性设置为 “true”，即使用服务器指定的包大小。	True
ENCRYPT_PASSWORD	允许安全登录。当设置为 “true” 时，登录和远程站点口令都会被加密并被发送到服务器。	False
ESCAPE_PROCESSING_DEFAULT	避免处理 SQL 语句中的 JDBC 函数转义。缺省情况下， jConnect 会分析提交到数据库的所有 SQL 语句，以查找有效的 JDBC 函数转义。如果应用程序不在其 SQL 调用中使用 JDBC 函数转义，可将此连接属性设置为 “false” 以避免此处理。这样做可以使性能获得略微改善。	True
EXPIRESTRING	包含许可证有效期 date。除 jConnect 的评估副本之外，其余副本都永不过期。这是一个只读属性。	Never

属性	说明	缺省值
FAKE_METADATA	返回假元数据。如果在调用 ResultSetMetaData 方法 getCatalogName、getSchemaName 和 getTableName 时此属性设置为 “true”，由于服务器不提供有用的元数据，调用过程将返回空字符串 ("")。 如果此属性设置为 “false”，调用这些方法将抛出一个 “Not Implemented” SQLException。 **注释** 如果已启用宽表且正在使用 Adaptive Server 12.5 或更高版本，则将忽略此属性设置，原因是服务器确实提供了有用的元数据。	False
GET_BY_NAME_USES_COLUMN_LABEL	提供与 jConnect 6.0 之前版本的向后兼容性。 使用 Adaptive Server 12.5， jConnect 跟以前相比可以访问更多的元数据。在 12.5 版本之前， column name 和 column alias 代表同一事物。现在在使用 Adaptive Server 12.5 或更高版本且启用宽表的情况下， jConnect 可以区分这两种说法。 若要保留向后兼容性，请将此属性设置为 “true”。 如果希望调用 getByte、 getInt、 get* (String columnName) 以查看列的真实名称 （按 JDBC 2.0 规范调用），请将此属性设置为 “false”。	True
GSSMANAGER_CLASS	指定 org.ietf.jgss.GSSManager 类的一种第三方实现。 可将此属性设置为字符串或 GSSManager 对象。 如果将此属性设置为字符串，其值应是第三方 GSSManager 实现的全限定类名。如果将此属性设置为对象，该对象必须扩展 org.ietf.jgss.GSSManager 类。有关详细信息，请参见第 3 章 “安全性”。	Null

属性	说明	缺省值
HOSTNAME	标识当前主机名。	无。 最大长度为 30 个字符，如果超过此限制，则会截断为 30 个字符。
HOSTPROC	标识主机上的应用程序进程。	无
IGNORE_DONE_IN_PROC	确定不返回中间更新结果 （像在存储过程中那样），而只返回最终结果集。	False
IS_CLOSED_TEST	允许指定在调用 Connection.isClosed 时向数据库发送何种查询（如果有）。有关详细信息，请参见第 96 页的“使用 Connection.isClosed 和 IS_CLOSED_TEST”。	Null
JCONNECT_VERSION	设置版本专有的特性。请参见第 10 页的“JCONNECT_VERSION 连接属性”。	6
LANGUAGE	设置从服务器返回的错误消息和 jConnect 消息的语言。该设置必须是 *syslanguages* 中的一种语言。	视版本而定。请参见第 5 页的“设置 jConnect 版本”。
LANGUAGE_CURSOR	确定 jConnect 使用 “语言游标”而不使用 “协议游标”。 请参见第 136 页的 “游标性能”。	False
LITERAL_PARAMS	如果设置为 “true”， setXXX 方法在 PreparedStatement 接口中设置的任何参数在执行 SQL 语句时都将以文字形式插入 SQL 语句中。 如果设置为 “false”，参数标记将留在 SQL 语句中，而参数值被单独发送给服务器。	False
USE_METADATA	在建立连接时创建并初始化 DatabaseMetaData 对象。连接到指定的数据库必须要使用 DatabaseMetaData 对象。 jConnect 将 DatabaseMetaData 用于某些功能，如分布式事务管理支持 (JTA/JTS) 和动态类装载 (DCL)。 如果收到错误 010SJ （表明应用程序需要元数据），请安装 jConnect 附带的用于返回元数据的存储过程。请参见 *jConnect for JDBC 安装指南* 第 3 章中的 “安装存储过程”。	True

属性	说明	缺省值
PACKETSIZE	标识网络包大小。如果您要使用 ASE 15.0，Sybase 建议您不要设置此属性，并让 jConnect 和 ASE 15.0 选用适合于您的环境的网络包大小。	512
PASSWORD	标识登录口令。 如果使用 getConnection(String, String, String) 方法，将自动设置；如果使用 getConnection(String, Props)，将显式设置。	无
PRELOAD_JARS	包含以逗号分隔的与指定 CLASS_LOADER 相关联的 *.jar* 文件名的列表。这些 *.jar* 文件在连接时装载，可由任何其它使用相同 jConnect 驱动程序的连接使用。有关详细信息，请参见第 84 页的 “预装载 .jar 文件”。	Null
PROTOCOL_CAPTURE	指定用于捕获应用程序和 Adaptive Server 间的 TDS 通信的文件。	Null
PROXY	指定网关地址。对于 HTTP 协议， URL 是 http://host:port。 若要使用支持加密的 HTTPS 协议， URL 应为 https://host:port/servlet_alias。	无
QUERY_TIMEOUT_CANCELS_ALL	强制 jConnect 在读取超时时取消连接上的所有语句。可以在客户端调用 execute() 但由于死锁 （例如，试图读取当前正在另一个事务中更新的表）而超时时使用此行为。缺省值为 *false*。此属性可能会与受 BE_AS_JDBC_COMPLIANT_AS_POSSIBLE 属性影响的属性值归并在一起，这将取决于与 Sun 进行磋商的结果。	False
REMOTEPWD	包含用于通过服务器到服务器远程过程调用进行访问的远程服务器口令。请参见第 47 页的 “执行服务器到服务器的远程过程调用”。	无
REPEAT_READ	确定驱动程序是否保留列和输出参数的副本，以便可以随机读取或重复读取列。请参见第 128 页的“REPEAT_READ 连接属性”。	True

属性	说明	缺省值
REQUEST_HA_SESSION	表明连接客户端是否希望启动一个与为故障切换配置的 Adaptive Server 12 或更高版本的高可用性 (HA) 故障切换会话。请参见第 42 页的 “实现高可用性故障切换支持”。 **注释** 此属性设置为 “true” 将导致 jConnect 尝试故障切换登录。如果没有正确设置此连接属性，即使已为故障切换配置了服务器，也不能启动故障切换会话。 连接建立后将不能重新设置此属性。 如果希望请求故障切换会话具有更高的灵活性，可以对客户端应用程序进行编码，使之在运行时设置 REQUEST_HA_SESSION。	False
REQUEST_KERBEROS_SESSION	确定 jConnect 是否使用 Kerberos 进行鉴定。如果此属性设置为 “true”，还必须指定 SERVICE_PRINCIPAL_NAME 属性的值。 可能还要为 GSSMANAGER_CLASS 属性提供值。有关详细信息，请参见第 3 章 “安全性”。	False
RMNAME	在使用分布式事务 (XA) 时设置资源管理器名称。此属性将覆盖可能在 LDAP 服务器条目中设置的资源管理器名称。有关详细信息，请参见第 91 页的 “分布式事务管理支持”。	Null
SECONDARY_SERVER_HOSTPORT	在客户端使用 HA 故障切换会话时设置辅助服务器的主机名和端口。此属性的值应采用下列格式：hostName:portNumber。除非将 REQUEST_HA_SESSION 也设置为 “true”，否则此属性将被忽略。有关详细信息，请参见第 42 页的 “实现高可用性故障切换支持”。	Null

属性	说明	缺省值
SELECT_OPENS_CURSOR	确定对 Statement.executeQuery 的调用在查询包含 FOR UPDATE 子句时是否自动生成游标。 如果您以前已在同一个语句上调用过 Statement.setFetchSize 或 Statement.setCursorName，则将 SELECT_OPENS_CURSOR 设置为 “true” 没有效果。 **注释** 将 SELECT_OPENS_CURSOR 设置为 “true” 时，系统性能将可能会下降。 有关在 jConnect 中使用游标的详细信息，请参见第 50 页的 “对结果集使用游标”。	False
SERIALIZE_REQUESTS	确定 jConnect 在发送其它请求之前是否等待服务器的响应。	False
SERVER_INITIATED_TRANSACTIONS	允许服务器控制事务。缺省情况下，该属性设置为 “true”，并且 jConnect 通过使用 Transact-SQL “set chained on” 来允许服务器启动并控制事务。如果设置为 “false”，则 jConnect 会通过使用 Transact SQL “begin tran” 来启动并控制事务。 Sybase 建议您允许服务器控制事务。	True
SERVICENAME	指示 DirectConnect 网关服务的后台数据库服务器的名称。还用于指示在连接到 Adaptive Server Anywhere 之后应使用的数据库。	无
SERVERTYPE	在连接到 OpenSwitch 时将此属性设置为 “OSW”。这使 jConnect 可以向 OpenSwitch 发送某些指令，允许 OpenSwitch 在移动到另一个服务器实例时记住初始的连接设置，例如隔离级别、textsize、带引号的标识符和 autocommit。	无
SERVICE_PRINCIPAL_NAME	在建立到 Adaptive Server Enterprise 的 Kerberos 连接时使用。此属性的值应与密钥分发中心 (KDC) 中的服务器条目和数据库在其下运行的服务器名称对应。 如果 REQUEST_KERBEROS_SESSION 属性设置为 “false”，将会忽略 SERVICE_PRINCIPAL_NAME 属性的值。有关详细信息，请参见第 3 章 “安全性”。	Null

属性	说明	缺省值
SESSION_ID	TDS 会话 ID。如果设置了此属性，jConnect 将认为应用程序正试图恢复由 TDS 贯通网关保持打开的现有 TDS 会话上的通信。jConnect 将跳过登录协商并将所有来自应用程序的请求转发到指定的会话 ID。	Null
SESSION_TIMEOUT	指定 HTTP 贯通会话（使用 jConnect TDS 贯通服务器小程序创建）在空闲时保持活动的时间量 （以秒为单位）。在达到指定时间后，连接会自动关闭。有关 TDS 贯通服务器小程序的详细信息，请参见第 149 页。	Null
SQLINITSTRING	定义要在连接打开时传递给数据库服务器的一组命令。这些命令必须是可以用 Statement.executeUpdate 方法执行的 SQL 命令。	Null
STREAM_CACHE_SIZE	指定用于高速缓存语句响应流的最大大小。	Null （无限制的高速缓存大小）
SYBSOCKET_FACTORY	使 jConnect 能够使用自定义套接字实现。 将 SYBSOCKET_FACTORY 设置为下列两项之一： • 实现 com.sybase.jdbcx.SybSocketFactory 的类的名称；或 • “DEFAULT”，这样将例示新的 java.net.Socket( ) 使用此属性建立到数据库的 SSL 连接。 请参见第 31 页的 “实现自定义套接字插件”。	Null
TEXTSIZE	允许设置 TEXTSIZE。缺省情况下，ASE 和 ASA 允许从图像或文本列中读取 32627 字节。如果已经安装了 jConnect mda 表，jConnect 会将该值更改为 2GB。不过，在连接至 OpenSwitch 的情况下设置此值，会使该连接在 OpenSwitch 移动到另一个服务器实例时记住设置。	2GB
USER	指定登录 ID。 如果使用 getConnection(String, String, String) 方法，将自动设置；如果使用 getConnection(String, Props)，将显式设置。	无
VERSIONSTRING	提供 JDBC 驱动程序的只读版本信息。	jConnect 驱动程序版本

以下代码是设置连接属性的一个示例。随 jConnect 提供的示例程序也包含设置这些属性的示例。

```
Properties props = new Properties();
 props.put("user", "userid");
 props.put("password", "user_password");
/*
 * If the program is an applet that wants to access
 * a server that is not on the same host as the
 * web server, then it uses a proxy gateway.
 */
 props.put("proxy", "localhost:port");
/*
 * Make sure you set connection properties before
 * attempting to make a connection. You can also
 * set the properties in the URL.
 */
Connection con = DriverManager.getConnection
  ("jdbc:sybase:Tds:host:port", props);
```

连接到 Adaptive Server

在 Java 应用程序中，定义一个使用 jConnect 驱动程序连接到 Adaptive Server 的 URL。此 URL 的基本格式如下：

```
jdbc:sybase:Tds:host:port
```

其中：

- jdbc:sybase 标识驱动程序。
- Tds 是 Adaptive Server 的 Sybase 通信协议。
- *host:port* 是 Adaptive Server 主机名和监听端口。有关数据库或 Open Server 应用程序使用的条目，请参见 *$SYBASE/interfaces* (UNIX) 或 *%SYBASE%\ini\sql.ini* (Windows)。从“查询”条目中获取 *host:port*。

可使用下面的格式连接到特定的数据库：

```
jdbc:sybase:Tds:host:port/database
```

注释　若要使用 Adaptive Server Anywhere 或 DirectConnect 连接到特定的数据库，请使用 SERVICENAME 连接属性指定数据库名而不使用“/database”。

示例

以下代码创建与对端口 3697 进行侦听的主机 “myserver”上的 Adaptive Server 的连接：

```
SysProps.put("user","userid");
SysProps.put("password","user_password");
String url = "jdbc:sybase:Tds:myserver:3697";
Connection_con =
   DriverManager.getConnection(url,SysProps);
```

URL 连接属性参数

可以在定义 URL 时指定 jConnect 驱动程序连接属性的值。

注释 在 URL 中设置的驱动程序连接属性不会替换任何在应用程序中使用 DriverManager.getConnection 方法设置的相应的连接属性。

若要在 URL 中设置连接属性，请将属性名及其值附加到 URL 定义中。使用语法：

```
jdbc:sybase:Tds:host:port/database?
   property_name=value
```

若要设置多个连接属性，请附加每个额外的连接属性及其值 （前面加上 “&”）。例如：

```
jdbc:sybase:Tds:myserver:1234/mydatabase?
   LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=myhost
```

如果其中一个连接属性的值包含 “&”，请在该连接属性值中的 “&” 之前加上一个反斜杠 (\)。例如，如果主机名为 “a&bhost”，请使用此语法：

```
jdbc:sybase:Tds:myserver:1234/mydatabase?
   LITERAL_PARAMS=true&PACKETSIZE=512&HOSTNAME=
   a\&bhost
```

请勿对连接属性值使用引号，即使这些值是字符串。例如，使用：

```
HOSTNAME=myhost
```

而不应使用：

```
HOSTNAME="myhost"
```

使用 JNDI 连接到服务器

在 jConnect 中，您可以使用 Java 命名和目录接口 (JNDI) 来提供连接信息，它可以提供：

- 一个集中位置，可以从中指定主机名和端口以连接到服务器。不需要在应用程序中添加特定的主机和端口号的代码。
- 一个集中位置，可以从中指定供所有应用程序使用的连接属性和缺省数据库。
- 用于处理不成功的连接尝试的 jConnect CONNECTION_FAILOVER 属性。CONNECTION_FAILOVER 设置为 “true” 时，jConnect 将尝试连接到 JNDI 名称空间中的主机 / 端口服务器地址序列，直到连接成功。

若要配合使用 jConnect 和 JNDI，需要确保在 JNDI 访问的任何目录服务中都能获得特定的信息，并且在 javax.naming.Context 类中设置了所需的信息。本节包含以下主题：

- 使用 JNDI 的连接 URL
- 所需的目录服务信息
- CONNECTION_FAILOVER 连接属性
- 提供 JNDI 环境信息

使用 JNDI 的连接 URL

若要指定 jConnect 使用 JNDI 来获取连接信息，请将 “jndi” 作为 URL 协议放置在 “sybase” 的后面：

```
jdbc:sybase:jndi:protocol-information-for-use-with-JNDI
```

URL 中 “jndi” 后面的任何内容都将通过 JNDI 进行处理。例如，若要使用 JNDI 处理轻量目录访问协议 (LDAP)，可以输入以下代码：

```
jdbc:sybase:jndi:ldap://LDAP_hostname:port_number/servername=
  Sybase11,o=MyCompany,c=US
```

此 URL 告知 JNDI 从 LDAP 服务器获取信息，提供了要使用的 LDAP 服务器的主机名和端口号，并且以特定 LDAP 形式提供了数据库服务器的名称。

所需的目录服务信息

当配合使用 JNDI 和 jConnect 时，JNDI 需要为目标数据库服务器返回以下信息：

- 要连接到的主机名和端口号
- 要使用的数据库的名称
- 不允许单个应用程序自己设置的任何连接属性

此信息需要按照固定格式存储在任何用于提供连接信息的目录服务中。要求的格式包括一个数值对象标识符 (OID)，它标识所提供的信息的类型（如目标数据库），随后是格式化信息（请参见第 23 页的“示例 1”）。

注释 可以使用别名代替 OID 引用属性。请参见第 24 页的 “示例 2”。

表 2-3 显示了要求的格式。

表 2-3：JNDI 的目录服务信息

属性说明	别名	OID (object_id)
LDAP 目录服务中的接口条目替换	sybaseServer	1.3.6.1.4.1.897.4.1.1
sybaseServer LDAP 属性的集合点	sybaseServer	1.3.6.1.4.1.897.4.2
版本属性	sybaseVersion	1.3.6.1.4.1.897.4.2.1
服务器名称属性	sybaseServer	1.3.6.1.4.1.897.4.2.2
服务属性	sybaseService	1.3.6.1.4.1.897.4.2.3
状态属性	sybaseStatus	1.3.6.1.4.1.897.4.2.4
地址属性	sybaseAddress	1.3.6.1.4.1.897.4.2.5
安全性机制属性	sybaseSecurity	1.3.6.1.4.1.897.4.2.6
重试次数属性	sybaseRetryCount	1.3.6.1.4.1.897.4.2.7
循环延迟属性	sybaseRetryDelay	1.3.6.1.4.1.897.4.2.8
jConnect 连接协议	sybaseJconnectProtocol	1.3.6.1.4.1.897.4.2.9
jConnect 连接属性	sybaseJconnectProperty	1.3.6.1.4.1.897.4.2.10
数据库名	sybaseDatabasename	1.3.6.1.4.1.897.4.2.11
高可用性故障切换服务器名称属性	sybaseHAservername	1.3.6.1.4.1.897.4.2.15
ResourceManager 名称	sybaseResourceManager Name	1.3.6.1.4.1.897.4.2.16
ResourceManager 类型	sybaseResourceManager Type	1.3.6.1.4.1.897.4.2.17
JDBCDataSource 接口	sybaseJdbcDataSource-接口	1.3.6.1.4.1.897.4.2.18
ServerType	sybaseServerType	1.3.6.1.4.1.897.4.2.19

注释　用斜体表示的属性是必需的。

以下示例显示为处于 LDAP 目录服务状态的数据库服务器“SYBASE11”输入的连接信息。示例 1 使用的是属性 OID。示例 2 使用的是属性别名，它不区分大小写。可以使用 OID 也可以使用别名。

示例 1

```
dn: servername=SYBASE11,o=MyCompany,c=US
 servername:SYBASE11
 1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1266
 1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1337
 1.3.6.1.4.1.897.4.2.5:TCP#1#standby1 4444
 1.3.6.1.4.1.897.4.2.10:REPEAT_READ=false&PACKETSIZE=1024
 1.3.6.1.4.1.897.4.2.10:CONNECTION_FAILOVER=true
 1.3.6.1.4.1.897.4.2.11:pubs2
 1.3.6.1.4.1.897.4.2.9:Tds
```

示例 2

```
dn: servername=SYBASE11,o=MyCompany,c=US
 servername:SYBASE11
 sybaseAddress:TCP#1#giotto 1266
 sybaseAddress:TCP#1#giotto 1337
 sybaseAddress:TCP#1#standby1 4444
 sybaseJconnectProperty:REPEAT_READ=false&PACKETSIZE=1024
 sybaseJconnectProperty:CONNECTION_FAILOVER=true
 sybaseDatabasename:pubs2
 sybaseJconnectProtocol:Tds
```

在这些示例中，可以通过主机 “giotto”上的端口 1266 或端口 1337 访问 SYBASE11，并且可以通过主机 “standby1”上的端口 4444 访问。REPEAT_READ 和 PACKETSIZE 两个连接属性在一个条目中设置。CONNECTION_FAILOVER 连接属性是作为单独条目设置的。连接到 SYBASE11 的应用程序开始时是与 pubs2 数据库连接。您无需指定连接协议，但如果指定的话，您输入的属性必须是 “`Tds`”，而不是 “`TDS`”。

CONNECTION_FAILOVER 连接属性

CONNECTION_FAILOVER 是布尔值连接属性，可以在 jConnect 使用 JNDI 获取连接信息时使用该属性。

如果 CONNECTION_FAILOVER 设置为 “true”， jConnect 将多次尝试连接到服务器。如果连接到与服务器关联的主机和端口号的尝试失败，jConnect 将使用 JNDI 获取与该服务器关联的下一个主机和端口号，并通过它们尝试连接。连接尝试将顺序使用与服务器关联的所有主机和端口。

例如，如果 CONNECTION_FAILOVER 设置为 “true”，并且数据库服务器与下列主机和端口号相关联 （与前面 LDAP 示例中的相同）：

```
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1266
1.3.6.1.4.1.897.4.2.5:TCP#1#giotto 1337
1.3.6.1.4.1.897.4.2.5:TCP#1#standby 4444
```

为了获取到服务器的连接， jConnect 会尝试通过端口 1266 连接到主机 “giotto”。如果失败， jConnect 会尝试 “giotto” 上的端口 1337。如果失败， jConnect 会尝试通过端口 4444 连接到主机 “standby1”。

CONNECTION_FAILOVER 的缺省值是 “true”。

如果 CONNECTION_FAILOVER 设置为 “false”， jConnect 将尝试连接到初始的主机和端口号。如果尝试失败，jConnect 将抛出一个 SQL 例外并不再重试。

提供 JNDI 环境信息

若要配合使用 jConnect 和 JNDI，应熟悉 Sun Microsystems 的 JNDI 规范，该规范可从以下网址获得：

http://java.sun.com/products/jndi

特别是当 JNDI 和 jConnect 一起使用时，需要确保在 javax.naming.directory.DirContext 中设置所需的初始化属性。这些属性可以在系统级或在运行期设置。

两个重要属性是：

- Context.INITIAL_CONTEXT_FACTORY

  此属性包含 JNDI 使用的初始环境工厂的全限定类名。这确定了在 Context.PROVIDER_URL 属性中指定的 URL 使用的 JNDI 驱动程序。

- Context.PROVIDER_URL

  此属性包含驱动程序 （如 LDAP 驱动程序）要访问的目录服务的 URL。 URL 应该为字符串，例如 “ldap://ldaphost:427”。

以下示例显示如何在运行期设置环境属性以及如何使用 JNDI 和 LDAP 获得连接。在该示例中，INITIAL_CONTEXT_FACTORY 环境属性被设置为调用 LDAP 服务提供程序的 Sun Microsystem 实现。PROVIDER_URL 环境属性被设置为位于端口 983 的主机“ldap_server1”上的 LDAP 目录服务的 URL。

```
Properties props = new Properties();

 /* We want to use LDAP, so INITIAL_CONTEXT_FACTORY is set to the
 * class name of an LDAP context factory. In this case, the
 * context factory is provided by Sun's implementation of a
 * driver for LDAP directory service.
 */
 props.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.ldap.LdapCtxFactory");

 /* Now, we set PROVIDER_URL to the URL of the LDAP server that
 * is to provide directory information for the connection.
 */
 props.put(Context.PROVIDER_URL, "ldap://ldap_server1:983");

 /* Set up additional context properties, as needed. */
 props.put("user", "xyz");
 props.put("password", "123");

 /* get the connection */
 Connection con = DriverManager.getConnection
   ("jdbc:sybase:jndi:ldap://ldap_server1:983" +
   "/servername=Sybase11,o=MyCompany,c=US",props);
```

传递给 getConnection 的连接字符串包含开发人员必须提供的特定 LDAP 信息。

在运行期设置完 JNDI 属性后 （如上例所示）， jConnect 将它们传递给要用于初始化服务器的 JNDI，如以下 jConnect 代码所示：

```
javax.naming.directory.DirContext ctx =
   new javax.naming.directory.InitialDirContext(props);
```

jConnect 然后会通过调用 DirContext.getAtributes 从 JNDI 获取所需要的连接信息，如以下示例所示，其中 *ctx* 是一个 DirContext 对象：

```
javax.naming.directory.Attributes attrs =
   ctx.getAttributes(ldap://ldap_server1:983/servername=
     Sybase11, SYBASE_SERVER_ATTRIBUTES);
```

此例中， SYBASE_SERVER_ATTRIBUTES 是在 jConnect 中定义的字符串数组。数组值是表 2-3 中列出的必需目录信息的 OID。

实现自定义套接字插件

本节讨论如何将自定义的套接字实现插入到应用程序中以自定义客户端和服务器之间的通信。 javax.net.ssl.SSLSocket 是一个套接字示例，您可以对其进行自定义以启用加密。

com.sybase.jdbcx.SybSocketFactory 是包含返回 java.net.Socket 的 createSocket(String, int, Properties) 方法的 Sybase 扩展接口。若要使 jConnect 4.1 或更高版本的驱动程序能够装载自定义套接字，应用程序必须：

- 实现此接口
- 定义 createSocket 方法

jConnect 使用新套接字完成其后续输入 / 输出操作。实现 SybSocketFactory 的类创建套接字并提供一般框架以添加公共套接字级功能，如下所示：

```
/**
 * Returns a socket connected to a ServerSocket on the named host,
 * at the given port.
 * @param host  the server host
 * @param port  the server port
 * @param props  Properties passed in through the connection
 * @returns Socket
 * @exception IOException, UnknownHostException
 */
 public java.net.Socket createSocket(String host, int port, Properties props)
throws IOException, UnknownHostException;
```

传入属性允许 SybSocketFactory 的实例使用连接属性实现智能套接字。

在实现 SybSocketFactory 以产生套接字时，通过向应用程序传递创建套接字的不同种类的工厂或伪工厂可以使相同的应用程序代码使用不同种类的套接字。

可以使用在套接字结构中使用的参数自定义工厂。例如，可以使用已配置的不同的网络超时或安全参数自定义工厂以返回套接字。返回到应用程序的套接字可以是 java.net.Socket 的子类，以直接公开一些功能 （如压缩、安全性、记录标记、统计信息收集或防火墙贯通）的新 API (javax.net.SocketFactory)。

注释　SybSocketFactory 是过度简化的 javax.net.SocketFactory，可以使应用程序从 java.net.* 过渡到 javax.net.*

❖ **在 jConnect 中使用自定义套接字**

1 提供一个实现 com.sybase.jdbcx.SybSocketFactory 的 Java 类 请参见第 32 页的 “创建和配置自定义套接字”。

2 设置 SYBSOCKET_FACTORY 连接属性以使 jConnect 可以使用您的实现获取套接字。

若要在 jConnect 中使用自定义套接字，请将 SYBSOCKET_FACTORY 连接属性设置为以下字符串之一：

- 实现 com.sybase.jdbcx.SybSocketFactory 的类的名称

  或者

- DEFAULT，这样将例示新的 java.net.Socket。

有关如何设置 SYBSOCKET_FACTORY 的指导，请参见第 12 页的 “设置连接属性”。

创建和配置自定义套接字

jConnect 获得自定义套接字后将使用该套接字连接到服务器。对套接字的任何配置都必须在 jConnect 获得该套接字之前完成。

本节介绍如何使用 jConnect 插入 SSL 套接字实现 （如 javax.net.ssl.SSLSocket）。

注释 目前，只有 Adaptive Server 12.5 和更高版本支持 SSL。

以下示例显示 SSL 的实现如何创建、配置和返回 SSLSocket 实例。在该示例中， MySSLSocketFactory 类实现 SybSocketFactory 并扩展 javax.net.ssl.SSLSocketFactory 以实现 SSL。它包含两个 createSocket 方法 — 一个用于 SSLSocketFactory，另一个用于 SybSocketFactory，功能如下：

- 创建 SSL 套接字
- 调用 SSLSocket.setEnableCipherSuites 以指定加密可用的密码成套程序
- 返回 jConnect 要使用的套接字

示例

```
public class MySSLSocketFactory extends SSLSocketFactory
   implements SybSocketFactory
 {
 /**
 * Create a socket, set the cipher suites it can use, return
 * the socket.
 * Demonstrates how cither suites could be hard-coded into the
 * implementation.
 *
 * See javax.net.SSLSocketFactory#createSocket
 */
public Socket createSocket(String host, int port)
   throws IOException, UnknownHostException
 {
   // Prepare an array containing the cipher suites that are to
   // be enabled.
   String enableThese[] =
   {
       "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA",
       "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5",
       "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA"
   }
   ;
   Socket s =
       SSLSocketFactory.getDefault().createSocket(host, port);
   ((SSLSocket)s).setEnabledCipherSuites(enableThese);
   return s;
 }
/**
 * Return an SSLSocket.
 * Demonstrates how to set cipher suites based on connection
 * properties like:
 * Properties _props = new Properties();
 * Set other url, password, etc. properties.
 * _props.put(("CIPHER_SUITES_1",
 *    "SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA");
 * _props.put("CIPHER_SUITES_2",
 *    "SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5");
 * _props.put("CIPHER_SUITES_3",
 *     "SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA");
 * _conn = _driver.getConnection(url, _props);
 *
 * See com.sybase.jdbcx.SybSocketFactory#createSocket
 */
public Socket createSocket(String host, int port,
```

```
  Properties props)
  throws IOException, UnknownHostException
{
  // check to see if cipher suites are set in the connection
  // properites
  Vector cipherSuites = new Vector();
  String cipherSuiteVal = null;
  int cipherIndex = 1;
  do
  {
      if((cipherSuiteVal = props.getProperty("CIPHER_SUITES_"
          + cipherIndex++)) == null)
      {
          if(cipherIndex <= 2)
          {
              // No cipher suites available
              // return what the object considers its default
              // SSLSocket, with cipher suites enabled.
              return createSocket(host, port);
          }
          else
          {
              // we have at least one cipher suite to enable
              // per request on the connection
              break;
          }
          else
          }
              // add to the cipher suit Vector, so that
              // we may enable them together
              cipherSuites.addElement(cipherSuiteVal);
          }
      }
      while(true);
     // lets you create a String[] out of the created vector
      String enableThese[] = new String[cipherSuites.size()];
      cipherSuites.copyInto(enableThese);
     // enable the cipher suites
      Socket s =
          SSLSocketFactory.getDefault().createSocket
            (host, port);
      ((SSLSocket)s).setEnabledCipherSuites(enableThese);
     // return the SSLSocket
      return s;
  }
 // other methods
}
```

由于 jConnect 不需要套接字的种类信息，因此必须在返回套接字之前完成所有配置。

有关详细信息，请参见：

- *EncryptASE.java* – 位于 jConnect 安装的 *sample2* 子目录中，此示例显示如何与 jConnect 应用程序一起使用 SybSocketFactory 接口。
- *MySSLSocketFactoryASE.java* – 也位于 jConnect 安装的 *sample2* 子目录中，这是 SybSocketFactory 接口实现的一个示例，您可以将其插入到应用程序中使用。

处理国际化和本地化

本节讨论与 jConnect 有关的国际化和本地化问题。

使用 jConnect 传递 Unicode 数据

在 Adaptive Server 12.5 和更高版本中，数据库客户端可以使用 unichar 和 univarchar 数据类型。这两种数据类型可以实现 Unicode 数据的有效存储和检索。

以下为从 Unicode 标准 （版本 2.0）中引用的一段内容：

“Unicode 标准是对字符和文本进行编码的具有固定宽度的统一的编码方案。这种用于信息处理的国际字符代码的指令系统包括当今世界上的主要脚本和常用的技术符号。 Unicode 字符编码将字母字符、表意字符和符号同等对待，这意味着它们可以混合使用，且使用的难易程度相同。Unicode 标准使用 ASCII 字符集的模型，但使用 16 位编码以完全支持多语言文本。”

这意味着用户可以指定数据库表列存储 Unicode 数据，而不用考虑服务器的缺省字符集。

注释　在 Adaptive Server 12.5 到 12.5.0.3 版本中，服务器必须具有缺省字符集 utf-8 才能使用 Unicode 数据类型。但在 Adaptive Server 12.5.1 和更高版本中，数据库用户无需考虑服务器的缺省字符集即可使用 unichar 和 univarchar 数据类型。

服务器接受 unichar 和 univarchar 数据时， jConnect 将执行以下操作：

- 对于客户端要发送到服务器的所有字符数据 – 例如，使用 PreparedStatement.setString (int column, String value) – jConnect 将确定字符串是否能转换为服务器的缺省字符集。
- 如果 jConnect 确定这些字符不能转换为服务器的字符集 （例如，有些字符无法用服务器的字符集表示），它会将数据以 unichar/univarchar 数据编码发送给服务器。

例如，如果客户端尝试向以 iso_1 作为缺省字符集的 Adaptive Server 12.5.1 发送 Unicode 日语字符， jConnect 将检测到该日语字符不能转换为 iso_1 字符。 jConnect 随后以 Unicode 数据发送字符串。

客户端向服务器发送 unichar/univarchar 数据会降低计算机的性能。这是因为 jConnect 必须对不能直接映射到服务器的缺省字符集的所有字符串和字符执行两次字符到字节的转换。

如果您使用的是 6.05 以前的 jConnect 版本，并想要使用 unichar 和 univarchar 数据类型，您必须执行以下任务：

1 设置 JCONNECT_VERSION = 6 或更高版本。有关详细信息，请参见第 5 页的 “设置 jConnect 版本”。

2 您需要将 DISABLE_UNICHAR_SENDING 连接属性设置为 false。从 jConnect 6.05 开始，此属性在缺省情况下设置为 false。有关详细信息，请参见第 12 页的 “设置连接属性”。

注释 有关 unichar 和 univarchar 数据类型支持的详细信息，请参见 Adaptive Server 12.5 或更高版本的手册。

Connect 字符集转换程序

jConnect 对所有字符集转换使用特殊类。通过选择字符集转换程序类，可指定 jConnect 处理单字节和多字节字符集转换的方式以及转换对应用程序性能的影响。

共有两个字符集转换类。 jConnect 使用的转换类基于 JCONNECT_VERSION、 CHARSET 和 CHARSET_CONVERTER_CLASS 连接属性。

- TruncationConverter 类只能用于使用 ASCII 字符的单字节字符集（如 iso_1 和 cp850）。它不能用于多字节字符集或使用非 ASCII 字符的单字节字符集。

  通过使用 TruncationConverter 类，jConnect 6.05 可以使用与 jConnect 2.2 版相同的方式处理字符集。 JCONNECT_VERSION = 2 时，TruncationConverter 类是缺省转换程序。

- PureConverter 类是纯 Java 类多字节字符集转换程序。 jConnect 在 JCONNECT_VERSION = 4 或更高版本时使用此类。如果 jConnect 检测到在 CHARSET 连接属性中指定的字符集与 TruncationConverter 类不兼容，则在 JCONNECT_VERSION = 2 时也使用此转换程序。

  虽然 PureConverter 类能实现多字节字符集转换，但也可能降低 jConnect 驱动程序的性能。如果需要考虑驱动程序性能，请参见第 38 页的 “提高字符集转换性能”。

选择字符集转换程序

jConnect 使用 JCONNECT_VERSION 来确定要使用的缺省字符集转换程序类。 JCONNECT_VERSION = 2 或更高版本时，缺省值为 PureConverter 和 TruncationConverter。JCONNECT_VERSION = 4 或更高版本时，缺省值为 PureConverter。

也可以通过设置 CHARSET_CONVERTER_CLASS 连接属性指定希望 jConnect 使用的字符集转换程序。如果希望使用 jConnect 版本的缺省字符集转换程序之外的字符集转换程序，此方法将很有用。

例如，如果您设置 JCONNECT_VERSION = 4 或更高版本，但要使用 TruncationConverter 类而不使用多字节的 PureConverter 类，则您可以设置 CHARSET_CONVERTER_CLASS：

```
...
 props.put("CHARSET_CONVERTER_CLASS",
   "com.sybase.jdbc3.utils.TruncationConverter")
```

设置 CHARSET 连接属性

可以通过设置 CHARSET 驱动程序属性指定要在应用程序中使用的字符集。如果没有设置 CHARSET 属性：

- JCONNECT_VERSION = 2 时， jConnect 将使用 iso_1 作为缺省字符集。
- JCONNECT_VERSION = 3 和更高版本时，jConnect 将使用数据库的缺省字符集并会在客户端自动调整以执行任何必要的转换。

- 对于从 6.05 开始的 jConnect 版本，如果 jConnect 无法将用户数据成功转换为协商的字符集，则在服务器支持 Unicode 字符时 （ASE 12.5 或更高版本），它会向服务器发送未经转换的 Unicode 字符，否则会抛出例外。

也可以使用 IsqlApp 应用程序的 -J *charset* 命令行选项指定字符集。

若要确定 Adaptive Server 上安装的是哪种字符集，请在服务器上发出以下 SQL 查询：

```
select name from syscharsets
 go
```

对于 PureConverter 类，如果客户端 Java 虚拟机 (VM) 不支持指定的 CHARSET，连接将失败并抛出 SQLException，指出必须将 CHARSET 设置为 Adaptive Server 和客户端都支持的字符集。

如果使用 TruncationConverter 类，则无论指定的 CHARSET 是否是 7 位 ASCII 码，都将进行字符截断。因此，如果您的应用程序需要处理非 ASCII 数据 （例如任何亚洲语言），则不应使用 TruncationConverter，因为这会导致数据损坏。

提高字符集转换性能

如果使用多字节字符集并需要提高驱动程序性能，可以使用 jConnect 示例提供的 SunIoConverter 类。有关详细信息，请参见第 129 页的“SunIoConverter 字符集转换”。

另外，如果您的应用程序仅处理 7 位 ASCII 数据，则可以使用 TruncationConverter 来提高性能。

支持的字符集

表 2-4 列出了 jConnect 支持的 Sybase 字符集。表中还列出了每个支持的字符集的对应的 JDK 字节转换程序。

虽然 jConnect 支持 UCS-2，但目前 Sybase 数据库或 Open Server 都不支持 UCS-2。

Adaptive Server 12.5 和更高版本支持一个 Unicode 版本 （称为 UTF-16 编码）。

表 2-4 列出了目前 Sybase 支持的字符集。

表 2-4：支持的 Sybase 字符集

SybCharset 名称	JDK 字节转换程序
ascii_7	ASCII
big5	Big5
big5hk （参见注释）	Big5_HKSCS
cp037	Cp037
cp437	Cp437
cp500	Cp500
cp850	Cp850
cp852	Cp852
cp855	Cp855
cp857	Cp857
cp860	Cp860
cp863	Cp863
cp864	Cp864
cp866	Cp866
cp869	Cp869
cp874	Cp874
cp932	MS932
cp936	GBK
cp950	Cp950
cp1250	Cp1250
cp1251	Cp1251
cp1252	Cp1252
cp1253	Cp1253
cp1254	Cp1254
cp1255	Cp1255
cp1256	Cp1256
cp1257	Cp1257
cp1258	Cp1258
deckanji	EUC_JP
eucgb	EUC_CN
eucjis	EUC_JP
eucksc	EUC_KR
GB18030	GB18030
ibm420	Cp420

SybCharset 名称	JDK 字节转换程序
ibm918	Cp918
iso_1	ISO8859_1
iso88592	ISO8859-2
is088595	ISO8859_5
iso88596	ISO8859_6
iso88597	ISO8859_7
iso88598	ISO8859_8
iso88599	ISO8859_9
iso15	ISO8859_15_FDIS
koi8	KOI8_R
mac	Macroman
mac_cyr	MacCyrillic
mac_ee	MacCentralEurope
macgreek	MacGreek
macturk	MacTurkish
sjis	MS932
tis620	MS874
utf8	UTF8

注释 只有在使用 JDK 1.3 或更高版本时才支持 big5hk 字符集。

欧洲货币符号支持

jConnect 4.1 版和更高版本支持使用新的欧洲货币符号 （即 “*euro*”），并支持该符号与 UCS-2 Unicode 之间的相互转换。

euro 已被添加到以下 Sybase 字符集中：cp1250、 cp1251、 cp1252、 cp1253、 cp1254、 cp1255、 cp1256、 cp1257、 cp1258、 cp874、 iso885915 和 utf8。

若要使用 *euro* 符号：

- 使用 PureConvertor 或 CheckPureConverter 类，它是一个纯 Java 多字节字符集转换程序。有关详细信息，请参见第 36 页的 “Connect 字符集转换程序”。
- 检验是否在服务器上安装了新的字符集。

  euro 符号仅在 Adaptive Server Enterprise 上受支持； Adaptive Server Anywhere 8.0 版和更高版本为 *euro* 符号提供支持。
- 在客户端选择合适的字符集。有关详细信息，请参见第 37 页的 “设置 CHARSET 连接属性”。

不支持的字符集

jConnect 不支持以下 Sybase 字符集，因为 JDK 字节转换程序与这些 Sybase 字符集都不类似：

- cp1047
- euccns
- greek8
- roman8
- turkish8

可以使用 TruncationConverter 类使用这些字符集，只要应用程序只使用这些字符的 7 位 ASCII 子集即可。

处理数据库

本节讨论与 jConnect 相关的数据库问题，包括以下主题：

- 实现高可用性故障切换支持
- 执行服务器到服务器的远程过程调用
- 使用 ASE 12.5 和更高版本的宽表支持
- 访问数据库元数据
- 对结果集使用游标
- 批处理更新支持
- 通过存储过程的结果集更新数据库
- 使用数据类型

实现高可用性故障切换支持

jConnect 6.0 版和更高版本支持 Adaptive Server 12.0 版和更高版本中具有的故障切换功能。

注释 高可用性系统中的 Sybase 故障切换与 “连接故障切换” 功能不同。如果希望同时使用这两个功能， Sybase 强烈建议您*仔细*阅读本节内容。

概述

Sybase 故障切换允许配置两台 Adaptive Server 12.0 或更高版本作为协同服务器。如果主协同服务器发生故障，该服务器的设备、数据库和连接可以由辅助协同服务器接管。

可以对称或非对称地配置高可用性系统：

- *非对称*配置包括两个 Adaptive Server，它们分别位于不同的计算机上，并且彼此相连以便当一台服务器出现故障时可以由另一台承担它的工作量。辅助 Adaptive Server 充当 “热备份”，它只有在出现故障切换时才工作。

- *对称*配置也包括运行在两台计算机上的 Adaptive Server。但当出现故障切换时，其中每一个 Adaptive Server 都可以充当另一个 Adaptive Server 的主协同服务器或辅助协同服务器。在此配置中，每一个 Adaptive Server 都具有完整的功能，都具有各自的系统设备、系统数据库、用户数据库和用户登录。

在上述两种设置中，两台计算机都被配置为双向访问，这样使两台计算机都可以看到并访问对方的磁盘。

可以在 jConnect 中启用故障切换，然后将客户端应用程序连接到进行过故障切换配置的 Adaptive Server。如果主服务器故障切换到辅助服务器，客户端应用程序也会自动切换到辅助服务器并重新建立网络连接。

注释　有关详细信息，请参见*使用 Adaptive Server 的高可用性系统中的 Sybase 故障切换手册*。

要求、依赖性和限制

- 必须有两个进行过故障切换配置的 Adaptive Server 12.0 或更高版本。
- 当客户端发生故障切换时，只保留在发生故障切换前提交给数据库的更改。
- 必须将 REQUEST_HA_SESSION jConnect 连接属性设置为 “true”（请参见第 12 页的 “设置连接属性”）。
- 发生故障切换时 jConnect 事件通知不能工作。请参见第 70 页的 “使用事件通知”。
- 请关闭所有不再使用的语句。 jConnect 会存储有关语句的信息以启用故障切换。未关闭的语句将导致内存泄漏。

在 jConnect 中实现故障切换

可使用以下两种方法在 jConnect 中实现故障切换支持：

- 使用两个连接属性 （REQUEST_HA_SESSION 和 SECONDARY_SERVER_HOSTPORT），并按以下方式设置：
  - 将 REQUEST_HA_SESSION 设置为 “true”。
  - 将 SECONDARY_SERVER_HOSTPORT 设置为辅助服务器监听的主机名和端口号。请参见第 12 页的 “设置连接属性”和 “SECONDARY_SERVER_HOSTPORT”连接属性。
- 使用 JNDI 连接到服务器。请参见“使用 JNDI 连接到服务器”。在 JNDI 所需的目录服务信息文件中加入一个主服务器条目和一个辅助服务器条目。主服务器条目有一个引用辅助服务器条目的属性 (HA OID)。

  使用 LDAP 作为 JNDI 的服务提供程序时，此 HA 属性可以有以下三种可能的形式：

  a *相对区分名 (RDN)* — 此形式假定搜索库 （通常由 java.naming.provider.url 属性提供）与此属性的值相结合足以标识辅助服务器。例如，假定主服务器位于 “hostname:4200”，辅助服务器位于 “hostname:4202”：

```
dn: servername=haprimary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary
objectclass: sybaseServer

dn: servername=hasecondary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4202
objectclass: sybaseServer
```

  b *区分名 (DN)* — 此形式假定 HA 属性的值唯一标识辅助服务器，因此可能复制也可能不复制在搜索库中找到的值。例如：

```
dn: servername=haprimary, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4200
1.3.6.1.4.1.897.4.2.15: servername=hasecondary,
          o=Sybase, c=US ou=Accounting
objectclass: sybaseServer

dn: servername=hasecondary, o=Sybase, c=US, ou=Accounting
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4202
objectclass: sybaseServer
```

  请注意， `hasecondary` 位于树的其它分支上 （请参见附加的 `ou=Accounting` 限定符）。

c　*完整的 LDAP URL* — 此形式对搜索库没有任何假定。 HA 属性应是用于标识辅助服务器的全限定 LDAP URL （甚至可以指向不同的 LDAP 服务器）。例如：

```
dn: servername=hafailover, o=Sybase, c=US
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4200
1.3.6.1.4.1.897.4.2.15:
ldap://ldapserver:      386/servername=secondary,
          o=Sybase, c=US ou=Accounting
objectclass: sybaseServer

dn: servername=secondary, o=Sybase, c=US, ou=Accounting
1.3.6.1.4.1.897.4.2.5: TCP#1#hostname 4202
objectclass: sybaseServer
```

d　在 JNDI 所需的目录服务信息文件中，将 REQUEST_HA_SESSION 连接属性设置为 “true”，以使每次建立连接时都启用故障切换会话。

使用 REQUEST_HA_SESSION 连接属性表明连接客户端希望与进行过故障切换配置的 Adaptive Server 12.0 或更高版本开始一个故障切换会话。此属性设置为 “true”将导致 jConnect 尝试故障切换登录。如果没有设置此连接属性，即使正确配置了服务器也不会启动故障切换会话。 REQUEST_HA_SESSION 的缺省值是 “false”。

像设置其它任何连接属性一样设置此连接属性。连接建立后将不能重新设置此属性。

如果希望请求故障切换会话具有更高的灵活性，可以对客户端应用程序进行编码，使之在运行时设置 REQUEST_HA_SESSION。

以下示例显示为 LDAP 目录服务下的数据库服务器 “SYBASE11” 输入的连接信息，其中 “tahiti”是主服务器，“moorea”是辅助协同服务器：

```
dn: servername=SYBASE11,o=MyCompany,c=US
1.3.6.1.4.1.897.4.2.5:TCP#1#tahiti 3456
1.3.6.1.4.1.897.4.2.10:REPEAT_READ=false&PACKETSIZE=1024
1.3.6.1.4.1.897.4.2.10:CONNECTION_FAILOVER=false
1.3.6.1.4.1.897.4.2.11:pubs2
1.3.6.1.4.1.897.4.2.9:Tds
1.3.6.1.4.1.897.4.2.15:servername=SECONDARY
1.3.6.1.4.1.897.4.2.10:REQUEST_HA_SESSION=true

dn:servername=SECONDARY, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1#moorea 6000
```

- 使用 JNDI 和 LDAP 请求连接：
  - jConnect 使用 LDAP 服务器的目录确定主服务器和辅助服务器的名称和位置：

```
/* get the connection */
Connection con = DriverManager.getConnection
          ("jdbc:sybase:jndi:ldap://ldap_server1:983" +
          "/servername=Sybase11,o=MyCompany,c=US",props);
```

或者

  - 指定搜索库：

```
props.put(Context.PROVIDER_URL,
          "ldap://ldap_server1:983/ o=MyCompany, c=US");

Connection con=DriverManager.getConnection
          ("jdbc:sybase:jndi:servername=Sybase11", props);
```

登录到主服务器

如果 Adaptive Server 没有配置故障切换或者不能批准故障切换会话，客户端将无法登录并显示以下警告：

```
'The server denied your request to use the high-
availability feature.

Please reconfigure your database, or do not request a
high-availability session.'
```

故障切换到辅助服务器

发生故障切换时将抛出 SQL 例外 JZ0F2：

```
'Sybase high-availability failover has occurred. The
current transaction is aborted, but the connection is
still usable. Retry your transaction.'
```

然后客户端自动使用 JNDI 重新连接到辅助数据库。

注意：

- 客户端连接到的数据库的标识和任何提交的事务都被保留下来。
- 部分读取的结果集、游标和存储过程调用将丢失。
- 发生故障切换时，应用程序可能需要重新启动过程或返回到上一个完成的事务或活动。

故障恢复到主服务器

有时客户端会从辅助服务器故障恢复到主服务器。何时发生故障恢复是由系统管理员决定的，他在辅助服务器上发出 sp_failback。之后，客户端即可期望在主服务器上发生同样的行为和结果，如第 46 页的 “故障切换到辅助服务器”所述。

执行服务器到服务器的远程过程调用

在一台服务器上运行的 Transact-SQL 语言命令和存储过程可以执行位于另一台服务器上的存储过程。应用程序已连接到的服务器登录到远程服务器，并执行服务器到服务器的远程过程调用。

应用程序可以指定一个 “通用”口令供服务器间通信使用，即用于所有服务器间连接的口令。连接打开后，服务器便可使用此口令登录到任何远程服务器。缺省情况下， jConnect 使用当前连接的口令作为服务器间通信的缺省口令。

但如果同一用户在两台服务器上的口令不同，且该用户打算执行服务器到服务器的远程过程调用，则应用程序必须为每个要使用的服务器显示定义口令。

jConnect 4.1 版和更高版本提供一个属性，可以用来设置通用的 “远程”口令或在不同的服务器上设置不同的口令。 jConnect 允许使用 SybDriver 类中的 setRemotePassword 方法设置并配置此属性：

```
Properties connectionProps = new Properties();

public final void setRemotePassword(String serverName,
    String password, Properties connectionProps
```

若要使用此方法，应用程序必须先导入 SybDriver 类，然后再调用方法：

```
import com.sybase.jdbcx.SybDriver;
SybDriver sybDriver = (SybDriver)
    Class.forName("com.sybase.jdbc3.jdbc.SybDriver").n
ewInstance();
sybDriver.setRemotePassword
    (serverName, password, connectionProps);
```

注释　若要为不同的服务器设置不同的远程口令，请为每个服务器重复上述调用 （适合您的 jConnect 版本）。

此调用将给定的服务器名 - 口令对添加到给定的 Properties 对象中，该对象可以由应用程序在 DriverManager.getConnection （*server_url, props*）中传递给 DriverManager。

如果 serverName 为 null，则将 password 设置为通用口令，供后续到所有服务器 （不包括由前面的 setRemotePassword 调用明确指定的服务器）的连接使用。

如果应用程序设置了 REMOTEPWD 属性， jConnect 将不再设置缺省的通用口令。

使用 ASE 12.5 和更高版本的宽表支持

Adaptive Server 12.5 和更高版本提供的上限和参数要大于以前版本的数据库服务器。例如：

- 表可以包含 1,024 列。
- Varchar 和 varbinary 列可以包含超过 255 字节的数据。
- 在调用存储过程或向表中插入数据时最多可以发送和检索 2,048 个参数。

若要确保 jConnect 从数据库请求宽表支持， jConnect 6.0 和更高版本用户必须确保 JCONNECT_VERSION = 6 是缺省设置。

注释 如果将版本设置为低于 JCONNECT_VERSION = 6， jConnect 仍可继续使用 Adaptive Server 12.5 版和更高版本。但如果试图从需要宽表支持才能检索完整数据的表中选择数据，将可能遇到意外的错误或数据截断。

从不支持宽表的 Sybase 服务器访问数据时，也可以设置 JCONNECT_VERSION = 6 或更高版本。在这种情况下，服务器只是忽略宽表支持请求。

除了提供更多数量的列和参数之外，支持宽表还能为 jConnect 用户带来额外的好处，即更多的 ResultSetMetaData。例如，在 jConnect 6.0 之前的版本中， ResultSetMetaData 方法 getCatalogName、 getSchemaName 和 getTableName 都返回 “Not Implemented” SQLException，因为服务器没有提供元数据。启用宽表支持后，服务器将发送回此信息，上述三个方法将返回有用的信息。

访问数据库元数据

为了支持 JDBC DatabaseMetaData 方法， Sybase 提供了一组存储过程，jConnect 可以调用这些存储过程以获取数据库的元数据。为使 JDBC 元数据方法能正常工作，必须在服务器上安装这些存储过程。

如果 Sybase 服务器上尚未安装用于提供元数据的存储过程，可以使用随 jConnect 提供的存储过程脚本进行安装：

- *sql_server12.5.sql* 在 Adaptive Server 12.5 和更高版本的数据库中安装存储过程。
- *sql_server15.0.sql* 安装 Adaptive Server Enterprise 15.x 或更高版本的存储过程。
- *sql_asa.sql* 在 SQL Anywhere 数据库和 Adaptive Server Anywhere 数据库上安装存储过程。

注释　这些脚本的最新版本与所有版本的 jConnect 都兼容。

有关安装存储过程的详细说明，请参见 *Sybase jConnect for JDBC 安装指南* 和 *发行公告*。

此外，若要使用元数据方法，在建立连接时必须将 USE_METADATA 连接属性设置为 “true”（缺省值）。

不能获取有关数据库中临时表的元数据。

注释　DatabaseMetaData.getPrimaryKeys 方法查找在表定义 (CREATE TABLE) 中声明或使用 alter table (ALTER TABLE ADD CONSTRAINT) 声明的主键。它不查找使用 sp_primarykey 定义的键。

服务器端元数据安装

元数据支持既可以在客户端 （ODBC、 JDBC）又可以在数据源 （服务器存储过程）中实现。 jConnect 在服务器中提供元数据支持，这种方式具有下列优点：

- jConnect 很小，这样可确保能够快速从 Internet 上下载该驱动程序。
- 数据源中预装载的存储过程提高了运行时效率。
- 提供灵活性：jConnect 可以连接到多种数据库。

对结果集使用游标

jConnect 实现许多 JDBC 2.0 游标和更新方法。这些方法使得使用游标以及根据结果集中的值更新表的数据行变得更加容易。

在 JDBC 2.0 中， ResultSet 的主要特点在于其类型和并发性。类型和并发值是 java.sql.ResultSet 接口的一部分，由该接口的 javadoc 描述。

表 2-5 描述了 jConnect 6.05 中具有的 java.sql.ResultSet 的特性。如果服务器为 ASE 15.0 或更高版本，则在请求时，jConnect 6.05 可以打开服务器端可滚动游标。

表 2-5：jConnect 6.05 中可用的 java.sql.ResultSet 选项

	类型		
并发	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
CONCUR_READ_ONLY	支持	支持	不可用
CONCUR_UPDATABLE	支持	不可用	不可用

本节包括以下主题：

- 创建游标
- 使用 JDBC 1.x 方法进行定位型更新和删除
- 使用 JDBC 2.0 方法进行定位型更新和删除
- 为 PreparedStatement 对象使用游标
- 在 jConnect 中使用 TYPE_SCROLL_INSENSITIVE 结果集

创建游标

使用 jConnect 创建游标有两种方法：

- SybStatement.setCursorName

  可使用 SybStatement.setCursorName 显式指定游标名称。SybStatement.setCursorName 的签名是：

  ```
  void setCursorName(String name) throws SQLException;
  ```

- SybStatement.setFetchSize

  可使用 SybStatement.setFetchSize 创建游标并指定每次读取操作从数据库返回的行数。 SybStatement.setFetchSize 的签名是：

  ```
  void setFetchSize(int rows) throws SQLException;
  ```

  使用 setFetchSize 创建游标时， jConnect 驱动程序会为游标命名。若要获取游标名称，请使用 ResultSet.getCursorName。

另一种创建游标的方法是在连接上使用如下 JDBC 方法指定希望语句返回的 ResultSet 的类型：

```
Statement createStatement(int resultSetType, int
resultSetConcurrency)throws SQL Exception
```

类型和并发与表 2-5 列出的 ResultSet 接口中找到的类型和并发相对应。如果请求的是不受支持的 ResultSet，将在连接上链接一个 SQL 警告。执行返回的 Statement 时，您将收到与所请求的类型最接近的 ResultSet 类型。有关此方法的行为的详细信息，请参见 JDBC 规范。

如果不使用 createStatement，则 ResultSet 的缺省类型是：

- 如果只调用 Statement.executeQuery，则返回的 ResultSet 是类型和并发分别为 TYPE_FORWARD_ONLY 和 CONCUR_READ_ONLY 的 SybResultSet。
- 如果调用 setFetchSize 或 setCursorName，则从 executeQuery 返回的 ResultSet 是类型和并发分别为 TYPE_FORWARD_ONLY 和 CONCUR_UPDATABLE 的 SybCursorResultSet。

若要检查 ResultSet 对象的类型是否是您所需要的类型，请使用下列两种 ResultSet 方法：

```
int getConcurrency() throws SQLException;
int getType() throws SQLException;
```

❖ **创建和使用游标**

1　使用 Statement.setCursorName 或 SybStatement.setFetchSize 创建游标。

2　调用 Statement.executeQuery 为语句打开游标并返回游标结果集。

3　调用 ResultSet.next 读取行并在结果集中定位游标。

下面的示例分别使用上述两种方法创建游标并返回结果集。还使用 ResultSet.getCursorName 获取通过 SybStatement.setFetchSize 创建的游标的名称。

```
// With conn as a Connection object, create a
// Statement object and assign it a cursor using
// Statement.setCursorName().
Statement stmt = conn.createStatement();
stmt.setCursorName("author_cursor");

// Use the statement to execute a query and return
// a cursor result set.
ResultSet rs = stmt.executeQuery("SELECT au_id,
      au_lname, au_fname FROM authors
      WHERE city = 'Oakland'");
```

```
while(rs.next())
{
...
}

// Create a second statement object and use
// SybStatement.setFetchSize()to create a cursor
// that returns 10 rows at a time.
SybStatement syb_stmt = conn.createStatement();
syb_stmt.setFetchSize(10);

// Use the syb_stmt to execute a query and return
// a cursor result set.
SybCursorResultSet rs2 =
      (SybCursorResultSet)syb_stmt.executeQuery
      ("SELECT au_id, au_lname, au_fname FROM
authors
       WHERE city = 'Pinole'");
while(rs2.next())
{
...
}

// Get the name of the cursor created through the
// setFetchSize() method.
String cursor_name = rs2.getCursorName();
 ...
// For jConnect 6.0, create a third statement
// object using the new method on Connection,
// and obtain a SCROLL_INSENSITIVE ResultSet.
// Note: you no longer have to downcast the
// Statement or the ResultSet.
Statement stmt = conn.createStatement(
                 ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
ResultSet rs3 = stmt.executeQuery
   ("SELECT ... [whatever]");
// Execute any of the JDBC 2.0 methods that
// are valid for read only ResultSets.
rs3.next();
rs3.previous();
rs3.relative(3);
rs3.afterLast();
...
```

使用 JDBC 1.x 方法进行定位型更新和删除

下面的示例显示如何使用 JDBC 1.x 中的方法进行定位型更新。该示例创建两个 Statement 对象，一个用于将行选择到游标结果集中，另一个用于通过结果集中的行更新数据库。

注释　虽然本手册提供了与 JDBC 1.0 和 2.0 方法相关的示例代码，但 Sybase 强烈建议使用 JDBC 2.0，因为它使用方便并且可以移植。

```
// Create two statement objects and create a cursor
// for the result set returned by the first
// statement, stmt1. Use stmt1 to execute a query
// and return a cursor result set.
Statement stmt1 = conn.createStatement();
Statement stmt2 = conn.createStatement();
stmt1.setCursorName("author_cursor");
ResultSet rs = stmt1.executeQuery("SELECT
    au_id,au_lname, au_fname
    FROM authors WHERE city = 'Oakland'
    FOR UPDATE OF au_lname");

// Get the name of the cursor created for stmt1 so
// that it can be used with stmt2.
String cursor = rs.getCursorName();

// Use stmt2 to update the database from the
// result set returned by stmt1.
String last_name = new String("Smith");
while(rs.next())
{
    if (rs.getString(1).equals("274-80-9391"))
     {
       stmt2.executeUpdate("UPDATE authors "+
       "SET au_lname = "+last_name +
       "WHERE CURRENT OF " + cursor);
    }
}
```

结果集中的删除

下面的示例使用从上述代码中获得的 Statement 对象 *stmt2* 执行定位型删除：

```
stmt2.executeUpdate("DELETE FROM authors
          WHERE CURRENT OF " + cursor);
```

使用 JDBC 2.0 方法进行定位型更新和删除

本节介绍一些 JDBC 2.0 方法，可用于更新当前游标行中的列以及通过结果集中的当前游标行更新数据库。每个方法后面都有一个示例。

在结果集中更新列

JDBC 2.0 提供了多个方法，用于在客户端更新内存中的结果集的列值。然后可使用更新的值对基础数据库执行更新、插入或删除操作。所有这些方法都在 SybCursorResultSet 类中实现。

下面是 jConnect 中可用的一些 JDBC 2.0 更新方法：

```
void updateAsciiStream(String columnName, java.io.InputStream x,
   int length) throws SQLException;
void updateBoolean(int columnIndex, boolean x) throws
   SQLException;
void updateFloat(int columnIndex, float x) throws SQLException;
void updateInt(String columnName, int x) throws SQLException;
void updateInt(int columnIndex, int x) throws SQLException;
void updateObject(String columnName, Object x) throws
   SQLException;
```

用于通过结果集更新数据库的方法

JDBC 2.0 指定了两个新方法，用于根据结果集中的当前值更新或删除数据库中的行。这些方法在形式上比 JDBC 1.x 中的 Statement.executeUpdate 简单，并且不需要使用游标名称。它们在 SybCursorResultSet 中实现：

```
void updateRow() throws SQLException;
void deleteRow() throws SQLException;
```

注释 结果集的并发必须是 CONCUR_UPDATABLE。否则上述方法将引发例外。对于 insertRow，必须指定所有要求非空条目的表列。

DatabaseMetaData 提供的方法决定这些更改何时可见。

示例

下面的示例将创建一个用于返回游标结果集的 Statement 对象。对于结果集中的每一行，列值在内存中被更新，然后数据库通过该行的新列值被更新。

```
// Create a Statement object and set fetch size to
// 25. This creates a cursor for the Statement
// object Use the statement to return a cursor
// result set.
SybStatement syb_stmt =
(SybStatement)conn.createStatement();
syb_stmt.setFetchSize(25);
SybCursorResultSet syb_rs =
(SybCursorResultSet)syb_stmt.executeQuery(
    "SELECT * from T1 WHERE ...")

// Update each row in the result set according to
// code in the following while loop. jConnect
// fetches 25 rows at a time, until fewer than 25
// rows are left. Its last fetch takes any
// remaining rows.
while(syb_rs.next())
{
  // Update columns 2 and 3 of each row, where
// column 2 is a varchar in the database and
// column 3 is an integer.
  syb_rs.updateString(2, "xyz");
syb_rs.updateInt(3,100);
//Now, update the row in the database.
  syb_rs.updateRow();
}
// Create a Statement object using the
// JDBC 2.0 method implemented in jConnect 6.0
Statement stmt = conn.createStatement
(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
// In jConnect 6.0, downcasting to SybCursorResultSet is not
// necessary. Update each row in the ResultSet in the same
// manner as above
while (rs.next())
{
rs.updateString(2, "xyz");
rs.updateInt(3,100);
  rs.updateRow();
// Use the Statement to return an updatable ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM T1 WHERE...");
}
```

从 ResultSet 中删除行

若要从游标结果集中删除某一行，可按如下方式使用 SybCursorResultSet.deleteRow：

```
while(syb_rs.next())
{
    int col3 = getInt(3);
    if (col3 >100)
    {
    syb_rs.deleteRow();
    }
}
```

在 ResultSet 中插入行

下面的示例说明如何使用 JDBC 2.0 API 执行插入。不需要使用 SybCursorResultSet。

```
// prepare to insert
rs.moveToInsertRow();
// populate new row with column values
rs.updateString(1, "New entry for col 1");
rs.updateInt(2, 42);
// insert new row into db
rs.insertRow();
// return to current row in result set
rs.moveToCurrentRow();
```

为 PreparedStatement 对象使用游标

PreparedStatement 对象在创建后可以多次使用，每次使用时可为其输入参数指定相同或者不同的值。如果为 PreparedStatement 对象使用游标，则每次使用完游标后必须将其关闭，下次使用时再将其重新打开。关闭游标的结果集 (ResultSet.close) 时也会关闭该游标。执行游标的预准备语句 (PreparedStatement.executeQuery) 时会打开该游标。

下面的示例显示如何创建 PreparedStatement 对象，如何为其指定游标，以及如何执行两次 PreparedStatement 对象 （关闭然后重新打开游标）。

```
// Create a prepared statement object with a
// parameterized query.
PreparedStatement prep_stmt =
conn.prepareStatement(
"SELECT au_id, au_lname, au_fname "+
"FROM authors WHERE city = ? "+
"FOR UPDATE OF au_lname");
```

```
//Create a cursor for the statement.
prep_stmt.setCursorName("author_cursor");

// Assign the parameter in the query a value.
// Execute the prepared statement to return a
// result set.
prep_stmt.setString(1, "Oakland");
ResultSet rs = prep_stmt.executeQuery();

//Do some processing on the result set.
while(rs.next())
{
    ...
}

// Close the result, which also closes the cursor.
rs.close();

// Execute the prepared statement again with a new
// parameter value.
prep_stmt.setString(1,"San Francisco");
rs = prep_stmt.executeQuery();
// reopens cursor
```

在 jConnect 中使用 TYPE_SCROLL_INSENSITIVE 结果集

jConnect 仅支持 TYPE_SCROLL_INSENSITIVE 结果集。

jConnect 使用 Tabular Data Stream (TDS) （即 Sybase 专有协议）与 Sybase 数据库服务器进行通信。 ASE 15.0 或更高版本支持 TDS 可滚动游标。对于不支持 TDS 可滚动游标的服务器，在每次调用 ResultSet.next 时， jConnect 都会在客户端按需高速缓存行数据。但到达结果集的末尾时，整个结果集将存储到客户端内存中。由于这可能导致性能降低，因此， Sybase 建议您仅在使用 ASE 15.0 或结果集相当小时才使用 TYPE_SCROLL_INSENSITIVE 结果集。

注释 在 jConnect 中使用 TYPE_SCROLL_INSENSITIVE ResultSets 时，如果服务器不支持 TDS 可滚动游标，则只能在读取完 ResultSet 的最后一行后才能调用 isLast 方法。在未达到最后一行时调用 isLast 会导致抛出 UnimplementedOperationException。

---

jConnect 在 *sample2* 目录中提供了 ExtendResultSet，此示例使用 JDBC 1.0 接口提供了一个受限制的 TYPE_SCROLL_INSENSITIVE ResultSet。

此实现使用标准 JDBC 1.0 方法生成无滚动功能的只读结果集，即基础数据的一个静态视图，它不会即时反映在结果集为打开状态时所进行的更改。 ExtendedResultSet 在客户端高速缓存所有 ResultSet 行。对较大的结果集使用此类时应谨慎。

sample.ScrollableResultSet 接口：

- 是 JDBC 1.0 java.sql.ResultSet 的扩展。
- 定义了与 JDBC 2.0 java.sql.ResultSet 具有相同签名的其它方法。
- *只* 包含部分 JDBC 2.0 方法。未包含其中用于修改 ResultSet 的方法。

来自 JDBC 2.0 API 的方法有：

```
boolean previous() throws SQLException;
boolean absolute(int row) throws SQLException;
boolean relative(int rows) throws SQLException;
boolean first() throws SQLException;
boolean last() throws SQLException;
void beforeFirst() throws SQLException;
void afterLast() throws SQLException;
boolean isFirst() throws SQLException;
boolean isLast() throws SQLException;
boolean isBeforeFirst() throws SQLException;
boolean isAfterLast() throws SQLException;
int getFetchSize() throws SQLException;
```

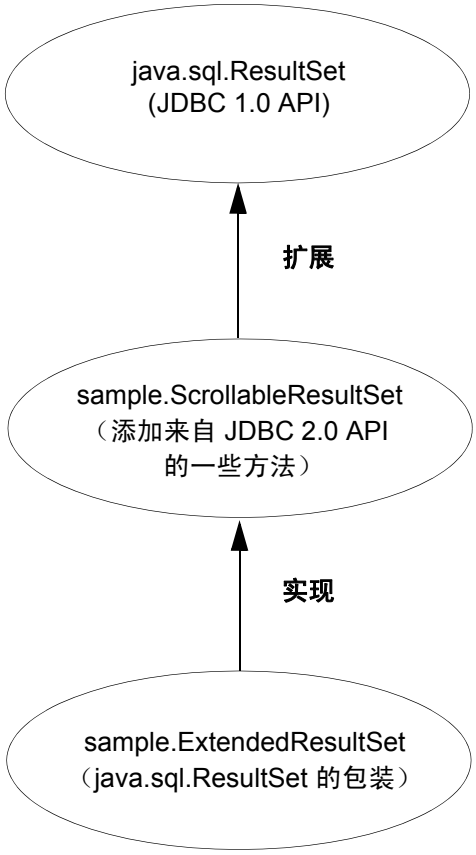
```
void setFetchSize(int rows) throws SQLException;
int getFetchDirection() throws SQLException;
void setFetchDirection(int direction) throws
SQLException;
int getType() throws SQLException;
int getConcurrency() throws SQLException;
int getRow() throws SQLException;
```

若要使用新的示例类，请使用任意 JDBC 1.0 java.sql.ResultSet 创建一个 ExtendedResultSet。以下为相关的代码段 （假定为 Java 1.1 环境）：

```
// import the sample files
import sample.*;
//import the JDBC 1.0 classes
import java.sql.*;
// connect to some db using some driver;
// create a statement and a query;
// Get a reference to a JDBC 1.0 ResultSet
ResultSet rs = stmt.executeQuery(_query);
// Create a ScrollableResultSet with it
ScrollableResultSet srs = new ExtendedResultSet(rs);
// invoke methods from the JDBC 2.0 API
srs.beforeFirst();
// or invoke methods from the JDBC 1.0 API
if (srs.next())
  String column1 = srs.getString(1);
```

图 2-1 是一个类框图，显示了新示例类和 JDBC API 的关系。

图 2-1：类框图



有关详细信息，请参见 JDBC 2.0 API，网址为：
http://java.sun.com/products/jdbc/jdbcse2.html。

批处理更新支持

批处理更新允许一个 Statement 对象向基础数据库提交多个 update 命令，这些命令作为一个单元进行处理。

注释　若要使用批处理更新，必须安装 jConnect 安装目录下 *sp* 目录中提供的最新元数据脚本。

有关对 Statement、PreparedStatement 和 CallableStatement 使用批处理更新的示例，请参见 *sample2* 子目录中的 *BatchUpdates.java*。

jConnect 还支持动态 PreparedStatements 的批处理。

实现说明

jConnect 按照 JDBC 2.0 API 中指定的方式实现批处理更新，但有下列例外：

- 如果实现 BatchUpdateException.getUpdateCounts 的 JDBC 2.0 标准将来放松了或发生了修改，jConnect 将继续执行原始标准，方法是让 BatchUpdateException.getUpdateCounts 返回一个小于 N 的 int[ ] 长度 M，表明批处理的前 M 个语句成功执行，第 M+1 个语句失败，第 M+2 到第 N 个语句没有执行。此处的 “N” 等于批处理中的语句总数。
- 若要以批处理 （非链式）模式调用存储过程，必须也以非链式模式创建存储过程。有关详细信息，请参见第 125 页的 “在非链式事务模式中执行存储过程”。
- 在 Adaptive Server 11.5.x 和更高版本中，如果服务器在批处理执行过程中遇到错误， BatchUpdateException.getUpdateCounts 只返回 int[ ] 长度值零。发生错误时整个事务回滚，因此成功操作的行数为零。

  注释　如果错误是重复的键行插入，则事务不回滚。

- 在 Adaptive Server 中，重复的键行插入不会导致批处理语句终止和回滚。服务器将继续处理批处理中的语句，除非遇到下列情况：发出 cancel 命令、批处理完成或发生不是重复键行插入的其它错误。由于 jConnect 在批处理过程中检测到任何异常后（包括重复键行插入）会向服务器发送 cancel 命令，因此无法确定在收到取消命令之前服务器已执行了批处理中的多少语句。因此， Sybase 强烈建议应按照 JDBC 规范操作，在事务中执行批处理时应将 autoCommit 设置为 “False”。这样可以回滚事务并使数据库返回到已知状态，然后再重新尝试批处理。
- Adaptive Server 11.0.1 为存储过程返回的受影响行数为 0 （零）。
- 在 SQL Anywhere 5.5.x 版中：
  - SQL Anywhere 5.5.x 不允许从包含插入的存储过程获得插入行计数。例如：

    ```
    create proc sp_A as insert tableA values (1,
    'hello A')
    create proc sp_B
    as
      insert tableA values (1, 'hello A')
    update tableA set col1=2
    create proc sp_C
    as
    update tableA set col1=2
    delete tableA
    ```

    对前面的存储过程运行 executeBatch 将分别产生下列结果：

    ```
    0 Rows Affected
    1 Rows Affected
    2 Rows Affected
    ```

  - 不支持动态 PreparedStatements 的批处理。
  - 由于按照 JDBC 2.0 规范 SQL Anywhere 5.5.x 不提供对批处理更新的内在支持，因此批处理更新由 executeUpdate 循环完成。
- 不支持批处理更新的数据库中的批处理更新：即使数据库不支持批处理更新， jConnect 仍可在 executeUpdate 循环中执行批处理更新。这样无论指向哪个数据库，都可以使用相同的批处理代码。

有关批处理更新的详细信息，请参见 *Sun Microsystems, Inc. JDBC 2.0 API*。

通过存储过程的结果集更新数据库

jConnect 提供 update 方法和 delete 方法，用于在由存储过程返回的结果集中获取游标。然后可使用游标的位置更新或删除提供结果集的基础表中的行。这些方法位于 SybCursorResultSet 中：

void updateRow(String *tableName*) throws SQLException;

void deleteRow(String *tableName*) throws SQLException;

tableName 参数标识提供结果集的数据库表。

若要获取存储过程返回的结果集中的游标，需要先使用 SybCallableStatement.setCursorName 或 SybCallableStatement.setFetchSize，然后再执行包含该过程的可调用语句。下面的示例显示如何在存储过程的结果集中创建游标，更新结果集的值，然后使用 SybCursorResultSet.update 方法更新基础表：

```
// Create a CallableStatement object for executing the stored
// procedure.
CallableStatement sproc_stmt =
   conn.prepareCall("{call update_titles}");

// Set the number of rows to be returned from the database with
// each fetch. This creates a cursor on the result set.
(SybCallableStatement)sproc_stmt.setFetchSize(10);

//Execute the stored procedure and get a result set from it.
SybCursorResultSet sproc_result = (SybCursorResultSet)
   sproc_stmt.executeQuery();

// Move through the result set row by row, updating values in the
// cursor's current row and updating the underlying titles table
// with the modified row values.
while(sproc_result.next())
{
   sproc_result.updateString(...);
   sproc_result.updateInt(...);
   ...
   sproc_result.updateRow(titles);
}
```

使用数据类型

本节介绍 numeric、 image、 text、 date、 time 和 char 数据的使用。

发送数值数据

jConnect 添加了 SybPreparedStatement 扩展以支持 Adaptive Server 处理 NUMERIC 数据类型的方式，可为该数据类型指定精度 （总位数）和标度 （小数点后的位数）。

Java 中与此对应的数据类型 （java.math.BigDecimal）稍有不同，当 jConnect 应用程序使用 setBigDecimal 方法控制输入 / 输出参数的值时，这些差异会引发问题。具体地说，有时参数 （无论是存储过程参数还是列）的精度和标度必须与对应的 SQL 对象的精度和标度完全一致。

为了加强 jConnect 应用程序对 setBigDecimal 方法的控制，在此方法中添加了 SybPreparedStatement 扩展：

```
public void setBigDecimal (int parameterIndex, BigDecimal X, int scale,
  int precision) throws SQLException
```

有关详细信息，请参见 jConnect 安装目录下 */sample2* 子目录中的 *SybPrepExtension.java* 示例。

更新数据库中的图像数据

jConnect 的 TextPointer 类包含 sendData 方法，可用于更新 Adaptive Server 或 Adaptive Server Anywhere 数据库中的 image 列。在 jConnect 的早期版本中，需要在 java.sql.PreparedStatement 中使用 setBinaryStream 方法发送图像数据。在当前版本中， TextPointer.sendData 方法使用 java.io.InputStream 将图像数据发送到 Adaptive Server 数据库中，并且极大地提高了性能。

警告！ jConnect 已不再支持 TextPointer 类，即在 jConnect 的后续版本中不推荐使用，并且可能会取消该类。

如果使用的数据服务器是 Adaptive Server 12.5 版或更高版本或者 Adaptive Server Anywhere 6.05 版或更高版本，请使用标准 JDBC 格式发送 image 数据：

```
PreparedStatement.setBinaryStream(int paramIndex,
  InputStream image)
```

若要获取 TextPointer 类的实例，可以在 SybResultSet 中使用两个 getTextPtr 方法之一：

```
public TextPointer getTextPtr(String columnName)
public TextPointer getTextPtr(int columnIndex)
```

TextPointer 类中的公共方法

com.sybase.jdbcx 包中包含 TextPointer 类。它的公共方法接口是：

```
public void sendData(InputStream is, boolean log)
  throws SQLException
public void sendData(InputStream is, int length,
  boolean log) throws SQLException
public void sendData(InputStream is, int offset,
  int length, boolean log) throws SQLException
public void sendData(byte[] byteInput, int offset,
  int length, boolean log) throws SQLEXception
```

其中：

- sendData(InputStream *is*, boolean *log)* 用指定的输入流中的数据更新 image 列。
- sendData(InputStream *is*, int *length*, boolean *log)* 用指定的输入流中的数据更新 image 列。 *length* 是要发送的字节数。
- sendData(InputStream *is*, int *offset*, int *length*, boolean *log)* 用指定的输入流中的数据更新 image 列，从 *offset* 参数给定的字节偏移处开始传送，一直传送完 *length* 参数指定的字节数。
- sendData(byte[ ] *byteInput*, int *offset*, int *length*, boolean *log)* 用 *byteInput* 参数指定的字节数组中所包含的图像数据更新列。更新从 *offset* 参数给定的字节偏移处开始，一直持续到读取完 *length* 参数指定的字节数为止。
- *log* 是每个方法中都有的参数，用于指定是否将 image 数据完全记录到数据库事务日志中。如果 *log* 参数设置为 “true”，则整个二进制图像将被写入到事务日志中。如果 *log* 参数设置为 “false”，则将记录更新操作，但图像本身并不写入日志。

❖ **使用 *TextPointer.sendData* 更新 *image* 列**

使用图像数据更新列：

1 为要更新的行和列获取一个 TextPointer 对象。

2 使用 TextPointer.sendData 执行更新操作。

以下两节通过示例说明了这些更新步骤。该示例发送 *Anne_Ringer.gif* 文件中的 image 数据，用以更新 pubs2 数据库中 au_pix 表的 pic 列。更新是针对 author ID 为 899-46-2035 的行进行的。

获取 TextPointer 对象

text 和 image 列包含 timestamp 和页位置信息，它们与列的文本和图像数据分开存放。在从 text 或 image 列中选取数据时，此额外信息作为结果集的一部分 “隐藏”起来。

用于更新 image 列的 TextPointer 对象需要此隐含信息，但不需要列数据的图像部分。为了获取此信息，需将这一列选取到 ResultSet 对象中，然后使用 SybResultSet.getTextPtr 提取文本指针信息，忽略图像数据，并创建 TextPointer 对象。请参见以下示例代码。

当一列中包含大量的图像数据时，为一行或多行选取列并等待获取所有数据很可能效率很低，因为不需要使用这些数据。为了缩短这一过程，请使用 set textsize 命令最小化数据包中返回的数据量。为达到这一目的，下面的代码示例在获取 TextPointer 对象时使用了 set textsize。

```
/*
 * Define a string for selecting pic column data for author ID
 * 899-46-2035.
 */
String getColumnData = "select pic from au_pix where au_id = '899-46-2035'";

 /*
 * Use set textsize to return only a single byte of column data
 * to a Statement object. The packet with the column data will
 * contain the "hidden" information necessary for creating a
 * TextPointer object.
 */
 Statement stmt= connection.createStatement();
 stmt.executeUpdate("set textsize 1");

 /*
 * Select the column data into a ResultSet object—cast the
 * ResultSet to SybResultSet because the getTextPtr method is
 * in SybResultSet, which extends ResultSet.
 */
 SybResultSet rs = (SybResultSet)stmt.executeQuery(getColumnData);
```

```
/*
* Position the result set cursor on the returned column data
* and create the desired TextPointer object.
*/
rs.next();
TextPointer tp = rs.getTextPtr("pic");

/*
* Now, assuming we are only updating one row, and won 諚 need
* the minimum textsize set for the next return from the server,
* we reset textsize to its default value.
*/
stmt.executeUpdate("set textsize 0");
```

使用 *TextPointer.sendData* 执行更新

以下代码利用了上一节中的 TextPointer 对象，使用 *Anne_Ringer.gif* 文件中的图像数据来更新 pic 列。

```
/*
 *First, define an input stream for the file.
 */
 FileInputStream in = new FileInputStream("Anne_Ringer.gif");

 /*
 * Prepare to send the input stream without logging the image data
 * in the transaction log.
 */
 boolean log = false;

 /*
 * Send the image data in Anne_Ringer.gif to update the pic
 * column for author ID 899-46-2035.
 */
 tp.sendData(in, log);
```

有关详细信息，请参见 jConnect 安装目录下 *sample2* 子目录中的 *TextPointers.java* 示例。

使用 *text* 数据

在早期版本中， jConnect 使用 TextPointer 类的 sendData 方法更新 Adaptive Server 或 Adaptive Server Anywhere 数据库中的 text 列。

Java 已不再支持 TextPointer 类，即在 Java 的后续版本中不推荐使用，并且可能会取消该类。

如果使用的数据服务器是 Adaptive Server 12.5 或更高版本或者 Adaptive Server Anywhere 6.05 版或更高版本，请使用标准 JDBC 格式发送文本数据：

```
PreparedStatement.setAsciiStream(int paramIndex,
  InputStream text, int length)
```

或者

```
PreparedStatement.setUnicodeStream(int paramIndex,
  InputStream text, int length)
```

或者

```
PreparedStatement.setCharacterStream(int paramIndex,
  Reader reader, int length)
```

使用 *date* 和 *time* 数据类型

Adaptive Server 12.5.1 和更高版本支持 SQL date 和 time 数据类型。以前，Adaptive Server 仅提供对 datetime 和 smalldatetime 数据类型的支持。这些数据类型曾由于以下原因而受到限制：

- 无法分开使用 time 和 date 数据类型。
- 早于 1753 年 1 月 1 日的日期是非法的。datetime 值只能采用 1753 年 1 月 1 日至 9999 年 12 月 31 日之间的日期。
- 当 JDBC 客户端使用 setTime 和 setDate 方法插入 java.sql.Time 或 java.sql.Date 方法时，这些值在服务器中被转换成 datetime 数据类型。这一转换导致将缺省的 date 或 time 添加到客户端提供的插入值中。

添加 date 和 time 数据类型具有以下优点：

- 现在，日期值在 0001 年 1 月 1 日至 9999 年 12 月 31 日之间都有效，这与 java.sql.Date 中的允许值完全一致。
- 现在 java.sql.Date 与 date 数据类型以及 java.sql.Time 与 time 数据类型之间存在直接的对应关系。

若要在 jConnect 6.0 中使用 date 和 time 数据类型，请确保将 JCONNECT_VERSION 属性设置为 “6”或更高版本。

实现说明

- 如果从包含 date 或 time 列的表中选择数据，并且尚未 （通过设置 jConnect 版本）在 jConnect 中启用 date/time 支持，则服务器会在返回数据之前将 date 或 time 转换为 datetime 值。如果返回的日期早于 1753 年 1 月 1 日，可能会导致出现错误。在这种情况下，会出现转换错误，同时数据库会发出错误通知。
- Adaptive Server Anywhere 支持 date 和 time 数据类型，但 date 和 time 数据类型并不直接与 Adaptive Server 12.5.1 和更高版本中的数据类型相兼容。在使用 jConnect 与 Adaptive Server Anywhere 进行通信时，应当继续使用 datetime 和 smalldatetime 数据类型。
- 在 Adaptive Server Anywhere 中， datetime 列的最大值是 1-1-7911 00:00:00。
- 使用 jConnect 时，如果试图将早于 1753 年 1 月 1 日的日期插入到 datetime 列或参数中，则会收到转换错误。
- 有关 date 和 time 数据类型的详细信息，请参见 Adaptive Server 手册；请特别注意其中有关可执行的隐式转换的一节。
- 如果对 Adaptive Server date、time 或 datetime 列使用 getObject，则返回的值分别为 java.sql.Date、 java.sql.Time 和 java.sql.Timestamp 数据类型。

使用 *char*/*varchar*/*text* 数据类型和 *getByte*

如果数据不是十六进制、八进制或十进制数据，请勿对 char、univarchar、 unichar、 varchar 或 text 字段使用 rs.getByte。

实现高级功能

本节介绍如何使用 jConnect 高级功能，包含以下主题：

- 使用事件通知
- 处理错误消息
- 将 Java 对象作为列数据存储在表中
- 使用动态类装载
- JDBC 2.0 可选软件包扩展支持

使用事件通知

可以使用 jConnect 事件通知功能，让应用程序在执行 Open Server 过程时获得通知。

若要使用这一功能，必须使用 SybConnection 类，该类扩展了 Connection 接口。 SybConnection 包含 regWatch 方法和 regNoWatch 方法，分别用于打开事件通知和关闭事件通知。

应用程序还必须实现 SybEventHandler 接口。该接口包含一个公共方法 void event(String proc_name, ResultSet params)，在发生指定事件时将调用该方法。事件的参数被传递给 event，后者告知应用程序如何进行响应。

若要在应用程序中使用事件通知，请调用 SybConnection.regWatch( ) 将应用程序注册到已注册过程的通知列表中。使用语法：

SybConnection.regWatch(*proc_name*,*eventHdlr*,*option*)

其中：

- *proc_name* 是一个字符串，是用于生成通知的注册过程的名称。
- *eventHdler* 是实现的 SybEventHandler 类的实例。
- *option* 是 NOTIFY_ONCE 或 NOTIFY_ALWAYS。如果希望应用程序仅在过程首次执行时得到通知，请使用 NOTIFY_ONCE。如果希望应用程序在过程每次执行时均得到通知，请使用 NOTIFY_ALWAYS。

每当 Open Server 上发生具有指定 *proc_name* 的事件时，jConnect 都会从独立的线程中调用 eventHdlr.event。 eventHdlr.event 在执行时会接收传递来的事件参数。因为这是一个独立的线程，所以事件通知不会阻止应用程序的执行。

如果 *proc_name* 不是已注册过程，或者如果 Open Server 无法将该客户端添加到通知列表中，则调用 regWatch 会抛出 SQL 例外。

若要关闭事件通知，请使用以下调用：

```
SybConnection.regNoWatch(proc_name)
```

警告！ 使用 Sybase 事件通知扩展时，应用程序需要对连接调用 close 方法，以移除首次调用 regWatch 时创建的子线程。否则，当退出应用程序时可能会导致虚拟机挂起。

事件通知示例

下面的示例说明在建立连接后如何实现事件处理程序，然后将事件注册到事件处理程序的一个实例：

```
public class MyEventHandler implements SybEventHandler
{
  // Declare fields and constructors, as needed.
  ...
  public MyEventHandler(String eventname)
  {
    ...
  }

  // Implement SybEventHandler.event.
  public void event(String eventName, ResultSet params)
  {
    try
    {
      // Check for error messages received prior to event
      // notification.
      SQLWarning sqlw = params.getWarnings();
      if sqlw != null
      {
        // process errors, if any
        ...
      }
      // process params as you would any result set with
      // one row.
```

```
      ResultSetMetaData rsmd = params.getMetaData();
      int numColumns = rsmd.getColumnCount();
      while (params.next())          // optional
      {
        for (int i = 1; i <= numColumns; i++)
        {
          System.out.println(rsmd.getColumnName(i) + " =
            " + params.getString(i));
        }
        // Take appropriate action on the event. For example,
        // perhaps notify application thread.
        ...
      }
    }
    catch (SQLException sqe)
    {
      // process errors, if any
      ...
    }
  }
}

public class MyProgram
{
  ...
  // Get a connection and register an event with an instance
  // of MyEventHandler.
  Connection conn = DriverManager.getConnection(...);
  MyEventHandler myHdlr = new  MyEventHandler("MY_EVENT");

  // Register your event handler.
  ((SybConnection)conn).regWatch("MY_EVENT", myHdlr,
    SybEventHandler.NOTIFY_ALWAYS);
  ...
 conn.regNoWatch("MY_EVENT");
  conn.close();
}
```

处理错误消息

jConnect 提供两个用于返回 Sybase 特定的错误信息的类：SybSQLException 和 SybSQLWarning，并提供了 SybMessageHandler 接口，用于自定义 jConnect 处理来自服务器的错误消息的方式。

检索 Sybase 特定的错误信息

jConnect 提供 EedInfo 接口来指定获取 Sybase 特定的错误信息的方法。EedInfo 接口在 SybSQLException 和 SybSQLWarning 中实现，它们是 SQLException 和 SQLWarning 的扩展类。

SybSQLException 和 SybSQLWarning 包含以下方法：

- public ResultSet getEedParams，返回包含附带了错误消息的所有参数值的单行结果集。
- public int getStatus，如果消息中有参数值，它将返回 “1”；如果没有，则返回 “0”。
- public int getLineNumber，返回引发了错误消息的存储过程或查询的行号。
- public String getProcedureName，返回引发了错误消息的过程的名称。
- public String getServerName，返回生成消息的服务器的名称。
- public int getSeverity，返回错误消息的严重性。
- public int getState，返回有关服务器中错误消息的内部源的信息。（仅用于 Sybase 技术支持部门。）
- public int getTranState，返回以下事务状态之一：
  - 0　连接当前处于扩展事务中。
  - 1　前一事务成功提交。
  - 3　前一事务中止。

有些错误消息可能是 SQLException 或 SQLWarning 消息，但不是 SybSQLException 或 SybSQLWarning 消息。应用程序应先检查其正在处理的例外类型，然后下转到 SybSQLException 或 SybSQLWarning。

自定义错误消息处理

可以使用 SybMessageHandler 接口来自定义 jConnect 处理服务器生成的错误消息的方式。通过在自己的类中实现 SybMessageHandler 来处理错误消息有如下好处：

- “通用”错误处理

  错误处理逻辑可放在错误消息处理程序中，而不需要在整个应用程序中重复。

- “通用”错误记录

  错误消息处理程序可包含处理所有错误记录的逻辑。

- 根据应用程序要求重新映射错误消息严重性

  错误消息处理程序可以包含识别特定错误消息的逻辑，并根据应用程序的要求而不是根据服务器的严重性级别来降低或提高错误的严重级。例如，在进行删除旧行的清除操作期间，可能想降低消息“一行不存在”的严重级。然而，在其它环境中又可能需要提高严重级。

注释 实现 SybMessageHandler 接口的错误消息处理程序仅接收服务器生成的消息。它们不处理由 jConnect 生成的消息。

---

jConnect 接收到错误消息时，它会检查是否已注册 SybMessageHandler 类来处理该消息。如果已注册， jConnect 将调用 messageHandler 方法，该方法接受 SQL 例外作为其参数。然后，jConnect 根据 messageHandler 返回的值处理该消息。错误消息处理程序可以：

- 依原样返回 SQL 例外。
- 返回 null。 jConnect 将忽略此消息。
- 根据 SQL 例外创建 SQL 警告，并将其返回。警告结果被添加到警告消息链中。
- 如果起始消息是 SQL 警告，messageHandler 可将此 SQL 警告评估为紧急，并创建和返回一个 SQL 例外，一旦 jConnect 获得控制就会抛出此例外。

安装错误消息处理程序

通过从 SybDriver、SybConnection 或 SybStatement 调用 setMessageHandler 方法可安装实现 SybMessageHandler 的错误消息处理程序。如果从 SybDriver 安装错误消息处理程序，则所有后续 SybConnection 对象都会继承此处理程序。如果从 SybConnection 对象安装错误消息处理程序，则由该 SybConnection 对象创建的所有 SybStatement 对象都会继承此处理程序。

此继承关系仅从安装了错误消息处理程序对象后才起作用。例如，如果创建一个名为 “myConnection”的 SybConnection 对象，然后调用 SybDriver.setMessageHandler 来安装错误消息处理程序对象，则 “myConnection”无法使用此对象。

若要返回当前错误消息处理程序对象，请使用 getMessageHandler。

错误消息处理程序示例

```
import java.io.*;
import java.sql.*;
import com.sybase.jdbcx.SybMessageHandler;
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybStatement;
import java.util.*;

public class MyApp
{
  static SybConnection conn = null;
  static SybStatement stmt = null
  static ResultSet rs = null;
  static String user = "guest";
  static String password = "sybase";
  static String server = "jdbc:sybase:Tds:192.138.151.39:4444";
  static final int AVOID_SQLE = 20001;

  public MyApp()
  {
     try
     {
             Class.forName("com.sybase.jdbc3.jdbc.SybDriver").newInstance();
             Properties props = new Properties();
             props.put("user", user);
             props.put("password", password);
             conn = (SybConnection)
             DriverManager.getConnection(server, props);
             conn.setMessageHandler(new NoResultSetHandler());
```

```
            stmt =(SybStatement) conn.createStatement();
            stmt.executeUpdate("raiserror 20001 'your error'");

            for (SQLWarning sqw = _stmt.getWarnings();
            sqw != null;
            sqw = sqw.getNextWarning());
            {
               if (sqw.getErrorCode() == AVOID_SQLE);
               {
                System.out.println("Error" +  sqw.getErrorCode()+
                " was found in the Statement's warning list.");
                break;
               }
            }
            stmt.close();
            conn.close();
  }
  catch(Exception e)
  {
    System.out.println(e.getMessage());
    e.printStackTrace();
  }
}

class NoResultSetHandler implements SybMessageHandler
{
  public SQLException messageHandler(SQLException sqe)
  {
    int code = sqe.getErrorCode();
    if (code == AVOID_SQLE)
    {
      System.out.println("User " + _user + " downgrading " +
        AVOID_SQLE + " to a warning");
      sqe = new SQLWarning(sqe.getMessage(),
        sqe.getSQLState(),sqe.getErrorCode());
    }
    return sqe;
  }
}

public static void main(String args[])
{
  new MyApp();
}
```

将 Java 对象作为列数据存储在表中

某些数据库产品允许直接将 Java 对象作为列数据存储在数据库中。在这样的数据库中，Java 类被当作数据类型，可以声明以 Java 类作为其数据类型的列。

通过实现定义于 PreparedStatement 接口中的 setObject 方法以及定义于 CallableStatement 和 ResultSet 接口中的 getObject 方法， jConnect 可以将 Java 对象存储在数据库中。这样对使用本地 JDBC 类和方法的应用程序使用 jConnect 就可以直接将 Java 对象作为列数据进行存储和检索。

注释　若要使用 getObject 和 setObject，请将 jConnect 版本设置为 com.sybase.jdbcx.SybDriver.VERSION_4 或更高版本。请参见第 5 页的 “设置 jConnect 版本”。

下面几节介绍了使用 JDBC 与 jConnect 在表中存储和检索对象的要求和过程：

- 将 java 对象作为列数据存储的前提条件
- 将 Java 对象发送到数据库
- 接收来自数据库的 Java 对象

注释　Adaptive Server 12.0 及更高版本和 Adaptive Server Anywhere 6.0.x 及更高版本可在表中存储 Java 对象，但有一些限制。有关详细信息，请参见 *jConnect for JDBC 发行公告*。

将 java 对象作为列数据存储的前提条件

若要将属于用户定义的 Java 类的 Java 对象存储在列中，必须满足三个要求：

- 该类必须实现 java.io.Serializable 接口。这是因为 jConnect 使用本地 Java 序列和非序列将对象发送到数据库，并接收从数据库返回的对象。
- 类定义必须安装在目标数据库中，*或者* 您必须使用 DynamicClassLoader (DCL) 直接从 Adaptive Server Anywhere 或 Adaptive Server 服务器装载类，并在使用类时视其位于本地 CLASSPATH 中。有关详细信息，请参见第 81 页的 “使用动态类装载”。
- 客户端系统必须在 *.class* 文件中包含类定义，该文件可通过本地 CLASSPATH 环境变量访问。

将 Java 对象发送到数据库

若要将用户定义类的实例作为列数据发送，请使用下面的 setObject 方法之一（根据 PreparedStatement 接口中的指定）：

```
void setObject(int parameterIndex, Object x, int targetSqlType,
   int scale) throws SQLException;
void setObject(int parameterIndex, Object x, int targetSqlType)
   throws SQLException;
void setObject(int parameterIndex, Object x) throws SQLException;
```

在 jConnect 6.0 中，若要发送 Java 对象，可使用 java.sql.Types.JAVA_OBJECT 目标 sql.Type，或者使用 java.sql.Types.OTHER。

下面的示例定义了一个 Address 类，显示了 Friends 表（该表包含数据类型为 Address 类的 Address 列）的定义，并向表中插入了一行。

```
public class Address implements Serializable
{
  public String   streetNumber;
   public String   street;
   public String   apartmentNumber;
   public String   city;
   public int   zipCode;
  //Methods
  ...
}

/* This code assumes a table with the following structure
**  Create table Friends:
**  (firstname varchar(30)  ,
**   lastname varchar(30),
**  address Address,
**  phone varchar(15))
*/

// Connect to the database containing the Friends table.
Connection conn =
   DriverManager.getConnection("jdbc:sybase:Tds:localhost:5000",
     "username", "password");

// Create a Prepared Statement object with an insert statement
//for updating the Friends table.
PreparedStatement ps = conn.prepareStatement("INSERT INTO
   Friends values (?,?,?,?)");

// Now, set the values in the prepared statement object, ps.
// set firstname to "Joan."
```

```
ps.setString(1, "Joan");

// Set last name to "Smith."
ps.setString(2, "Smith");

// Assuming that we already have "Joan_address" as an instance
// of Address, use setObject(int parameterIndex, Object x) to
// set the address column to "Joan_address."
ps.setObject(3, Joan_address);

// Set the phone column to Joan's phone number.
ps.setString(4, "123-456-7890");

// Perform the insert.
ps.executeUpdate();
```

接收来自数据库的 Java 对象

客户端 JDBC 应用程序可将接收到的来自数据库的 Java 对象放到结果集中，或作为从存储过程返回的输出参数的值。

如果结果集包含作为列数据的 Java 对象，请在 ResultSet 接口中使用以下 getObject 方法之一检索该对象：

```
Object getObject(int columnIndex) throws SQLException;
Object getObject(String columnName) throws SQLException;
```

如果从存储过程返回的输出参数包含 Java 对象，请在 CallableStatement 接口中使用下面的 getObject 方法检索该对象：

```
Object getObject(int parameterIndex) throws SQLException;
```

下面的示例说明如何使用 ResultSet.getObject(int parameterIndex) 将在结果集中接收的对象指派给类变量。本示例使用前一节中所用的 Address 类和 Friends 表，提供了一个可在信封上打印姓名与地址的简单应用程序。

```
/*
 ** This application takes a first and last name, gets the
 ** specified person's address from the Friends table in the
 ** database, and addresses an envelope using the name and
 ** retrieved address.
 */
 public class Envelope
 {
   Connection conn = null;
   String firstName = null;
   String lastName = null;
   String street = null;
```

```
String city = null;
String zip = null;

public static void main(String[] args)
{
  if (args.length < 2)
  {
  System.out.println("Usage: Envelope <firstName>
    <lastName>");
  System.exit(1);
  }
  // create a 4" x 10" envelope
  Envelope e = new Envelope(4, 10);
  try
  {
    // connect to the database with the Friends table.
    conn = DriverManager.getConnection(
      "jdbc:sybase:Tds:localhost:5000", "username",
        "password");
    // look up the address of the specified person
    firstName = args[0];
    lastName = args[1];
    PreparedStatement ps = conn.prepareStatement(
      "SELECT address FROM friends WHERE " +
        "firstname = ? AND lastname = ?");
    ps.setString(1, firstName);
    ps.setString(2, lastName);
    ResultSet rs = ps.executeQuery();
    if (rs.next())
    {
      Address a = (Address) rs.getObject(1);
      // set the destination address on the envelope
      e.setAddress(firstName, lastName, a);
    }
    conn.close();


  }
  catch (SQLException sqe)
  {
    sqe.printStackTrace();
    System.exit(2);
  }
  // if everything was successful, print the envelope
  e.print();
}
private void setAddress(String fname, String lname, Address a)
```

```
  {
    street = a.streetNumber + " " + a.street + " " +
      a.apartmentNumber;
    city = a.city;
    zip = "" + a.zipCode;
  }
  private void print()
  {
    // Print the name and address on the envelope.
    ...
  }
}
```

可在 jConnect 安装目录下的 *sample2* 子目录中找到更多的 HandleObject.java 详细示例。

使用动态类装载

Adaptive Server Anywhere 6.0 和 Adaptive Server 12.0 及更高版本允许将 Java 类指定为：

- SQL 列的数据类型
- Transact-SQL 变量的数据类型
- SQL 列的缺省值

在早期版本中，只有出现在 jConnect CLASSPATH 中的类才是可访问的，也就是说，如果 jConnect 应用程序试图访问不在本地 CLASSPATH 中的类的实例，就将产生 java.lang.ClassNotFound 例外。

jConnect 6.05 及更高版本通过实现 DynamicClassLoader (DCL) 可直接从 Adaptive Server Anywhere 或 Adaptive Server 服务器装载类，并在使用类时视其位于本地 CLASSPATH 中。

会继承超类中的所有安全性功能。 Java 2 中实现的装载程序委托模型仍然存在 — jConnect 首先尝试从 CLASSPATH 中装载请求的类；如果失败， jConnect 就尝试 DynamicClassLoader。

有关使用 Java 和 Adaptive Server 的详细信息，请参见 *Adaptive Server 中的 Java*。

使用 *DynamicClassLoader*

若要使用 DCL 功能：

1 创建并配置类装载程序。 jConnect 应用程序的代码应该类似如下所示：

```
Properties props = new Properties();

// URL of the server where the classes live.
String classesUrl = "jdbc:sybase:Tds:myase:1200";

// Connection properties for connecting to above server.
props.put("user", "grinch");
props.put("password", "meanone");
...

// Ask the SybDriver for a new class loader.
DynamicClassLoader loader = driver.getClassLoader(classesUrl, props);
```

2 使用 CLASS_LOADER 连接属性，使得新的类装载程序可用于执行查询的语句。创建类装载程序后，即可将其传递给后续连接，如下所示 （接着第 1 步中的代码示例）：

```
// Stash the class loader so that other connection(s)
// can know about it.
props.put("CLASS_LOADER", loader);

// Additional connection properties
props.put("user", "joeuser");
props.put("password", "joespassword");

// URL of the server we now want to connect to.
String url = "jdbc:sybase:Tds:jdbc.sybase.com:4446";

// Make a connection and go.
Connection conn = DriverManager.getConnection(url, props);
```

假定 Java 类定义如下所示：

```
class Addr {
      String street;
      String city;
      String state;
}
```

假定 SQL 表定义如下所示：

```
create table employee (char(100) name, int empid, Addr address)
```

3　如果客户端应用程序 CLASSPATH 中缺少一个 Addr 类，请使用下面的客户端代码：

```
Statement stmnt = conn.createStatement();
// Retrieve some rows from the table that has a Java class
// as one of its fields.
ResultSet rs = stmnt.executeQuery(
      "select * from employee where empid = '19'");
if (rs.next() {
      // Even though the class is not in our class path,
      // we should be able to access its instance.
      Object obj = rs.getObject("address");
      // The class has been loaded from the server,
      // so let's take a look.
      Class c = obj.getClass();

    // Some Java Reflection can be done here
      // to access the fields of obj.
      ...
}
```

CLASS_LOADER 连接属性提供了一个方便的机制，可在许多连接中共享一个类装载程序。

应确保在许多连接中共享一个类装载程序不会导致类冲突。例如，假定 org.foo.Bar 类的两个不同且不兼容的实例存在于两个不同的数据库中，当使用同一装载程序访问这两个类时，就可能引发问题。在检查来自第一个连接的结果集时装载第一个类。当检查来自第二个连接的结果集时，已经装载了这个类。因此，将不会装载第二个类，而 jConnect 不能检测此情况。

不过， Java 具有一种内置机制，可确保类的版本与非序列化对象的版本信息匹配。 Java 至少会检测并报告上述情况。

类及其实例不需要驻留在同一数据库或服务器中，但装载程序和后续连接没有理由不引用同一数据库或服务器。

使用非序列化

下面的示例说明了如何非序列化本地文件中的对象。该序列化对象是一个驻留在服务器但不存在于 CLASSPATH 中的类的实例。

SybResultSet.getObject( ) 使用 DynamicObjectInputStream （装载来自 DynamicClassLoader 的类定义的 ObjectInputStream 的子类），而非缺省系统 （“boot”）类装载程序。

```
// Make a stream on the file containing the
//serialized object.
FileInputStream fileStream = new FileInputStream("serFile");
// Make a "deserializer" on it. Notice that, apart
//from the additional parameter, this is the same
//as ObjectInputStreamDynamicObjectInputStream
stream = new DynamicObjectInputStream(fileStream, loader);
// As the object is deserialized, its class is
//retrieved through the loader from our server.
Object obj = stream.readObject();stream.close();
```

预装载 *.jar* 文件

jConnect 6.05 版本有一个称为 PRELOAD_JARS 的连接属性。将 *.jar* 文件定义为以逗号分隔的 *.jar* 文件名的列表时，这些文件会被全部装载。在这种情况下，“JAR”会引用服务器使用的 “保留的 JARname”。这就是安装 Java 程序中指定的 *.jar* 文件名，例如：

```
install java new jar 'myJarName' from file '/tmp/mystuff.jar'
```

如果设置 PRELOAD_JARS， *.jar* 文件就与类装载程序关联，因此不需要为每个连接预装载这些文件。您只需为一个连接指定 PRELOAD_JARS。以后再试图预装载同样的 *.jar* 文件会影响性能，因为不必要从服务器检索 *.jar* 文件数据。

注释 Adaptive Server Anywhere 6.x 及更高版本不能将 *.jar* 文件作为一个实体返回，因此 jConnect 会依次迭代检索每个类。不过， Adaptive Server 12.x 及更高版本可检索整个 *.jar* 文件，并装载其包含的所有类。

高级功能

DynamicClassLoader 中有许多公共方法。有关详细信息，请参见 *JDBC_HOME/docs/en/javadocs* 中的 javadocs 信息。

附加功能包括能够在期望进行一系列类装载时使装载程序的数据库连接保持 “活动”状态，并能按照类名称显式装载单个类。

还可使用继承自 java.lang.ClassLoader 的公共方法。java.lang.Class 中处理装载类的方法也是可用的；不过，使用这些方法时要小心，因为其中某些方法会对使用哪些类装载程序进行假定。特别是应该使用 Class.forName 的 3 参数版本，否则使用系统 （“boot”）类装载程序。请参见第 73 页的 “处理错误消息”。

JDBC 2.0 可选软件包扩展支持

JDBC 2.0 可选软件包（旧称 *JDBC 2.0 标准扩展 API*）定义了多个 JDBC 2.0 驱动程序可实现的功能。 jConnect 6.05 版及更高版本已实现了以下可选软件包扩展功能：

- 用于命名数据库的 JNDI
  （使用 jConnect 支持的任何 Sybase DBMS）
- 连接归集
  （使用 jConnect 支持的任何 Sybase DBMS）
- 分布式事务管理支持
  （仅使用 Adaptive Server 12.0 版及更高版本，或使用 XA-Server™ 的 11.x 版）

以上功能要求具有标准 JDK 1.2.x 版本中不具有的类和 / 或接口。如果使用 JDK 1.2.x 或 JRE 安装，则必须下载并执行 javax.sql.* 和 javax.naming.*。不过，如果使用的是 JDK 1.3.x 或更高版本，则不需要进行额外下载，因为这些类是标准 Java 安装数据库和连接归集的一部分；若要实现分布式事务管理支持，则必须下载 javax.transaction.xa.*。

注释　Sybase 建议使用 JNDI 1.2，它可与 Java 1.1.6 及更高版本兼容。

用于命名数据库的 JNDI

参考

JDBC 2.0 可选软件包（旧称 *JDBC 2.0 标准扩展 API*）的第 5 章“JNDI 与 JDBC API”。

相关接口

- javax.sql.DataSource
- javax.naming.Referenceable
- javax.naming.spi.ObjectFactory

此功能为 JDBC 客户端提供了按标准方法获取数据库连接的替代方法。客户端可使用逻辑名访问 JNDI 命名服务器，然后检索 javax.sql.DataSource 对象，而不是调用 Class.forName（“com.sybase.jdbc3.jdbc.SybDriver”），然后将 JDBC URL 传递到 DriverManager 的 getConnection( ) 方法。此对象负责装载驱动程序，并与它代表的物理数据库建立连接。客户端代码更简单并且是可重用的，因为特定于供应商的信息已放入 DataSource 对象中。

DataSource 对象的 Sybase 实现是 com.sybase.jdbcx.SybDataSource （有关详细信息，请参见 javadocs）。此实现通过使用 JavaBean 组件的设计模式支持以下标准属性：

- databaseName
- dataSourceName
- description
- networkProtocol
- password
- portNumber
- serverName
- user

注释 roleName 不受支持。

jConnect 提供 javax.naming.spi.ObjectFactory 接口的实现，因此 DataSource 对象可根据命名服务器条目的属性构造。给定 javax.naming.Reference，或 javax.naming.Name 和 javax.naming.DirContext 时，此 factory 可构造 com.sybase.jdbcx.SybDataSource 对象。若要使用此 factory，请将 java.naming.object.factory 系统属性设置为包括 com.sybase.jdbc3.SybObjectFactory。

用法

可以在不同的数据库中，以不同的方式使用 DataSource。下面的几小节中介绍所有选项并提供一些代码示例，以指导您完成该过程。有关详细信息，请参见 *JDBC 2.0 可选软件包*（旧称 *JDBC 2.0 标准扩展 API*）以及 Sun Web 站点上的 JNDI 文档。

1a. 管理员进行的配置：LDAP

jConnect 自 4.0 版本开始已经支持 LDAP 连接。因此，建议的方法（不要求自定义软件）是使用 LDAP 数据交换格式 (LDIF) 将 DataSources 配置为 LDAP 条目。例如：

```
dn:servername:myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
```

1b. 由客户端访问

这是典型的 JDBC 客户端应用程序。唯一的区别是获取对 DataSource 对象的引用的方式是通过访问命名服务器，而不是通过访问 DriverManager 并提供 JDBC URL。获取连接后，客户端代码就和所有其它 JDBC 客户端代码相同了。代码非常通用，且设置对象 factory 属性（可设置为环境的一部分）时仅引用 Sybase。

jConnect 安装包含了示例程序 *sample2/SimpleDataSource.java* 以说明 DataSource 的使用。此示例仅用于参考，也就是说，除非适当地配置环境并编辑此示例，否则不能运行它。 *SimpleDataSource.java* 包含如下重要代码：

```
import javax.naming.*;
import javax.sql.*;
import java.sql.*;

// set necessary JNDI properties for your environment (same as above)
Properties jndiProps = new Properties();

// used by JNDI to build the SybDataSource
jndiProps.put(Context.OBJECT_FACTORIES,
    "com.sybase.jdbc3.jdbc.SybObjectFactory");
```

```
// nameserver that JNDI should talk to
jndiProps.put(Context.PROVIDER_URL,   "ldap:
//some_ldap_server:238/o=MyCompany,c=Us");

// used by JNDI to establish the naming context
jndiProps.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");

// obtain a connection to your name server
Context ctx = new InitialContext(jndiProps);
DataSource ds = (DataSource) ctx.lookup("servername=myASE");

// obtains a connection to the server as configured earlier.
// in this case, the default username and password will be used
Connection conn = ds.getConnection();

// do standard JDBC methods
...
```

如果 Properties 已经在虚拟机中定义，则不必将这些属性显式传递给 InitialContext 构造函数，也就是说，要么在将 Java 设置为浏览器属性的一部分时传递，要么使用下面的方法传递：

```
java -Djava.naming.object.factory=com.sybase.jdbc3.jdbc.SybObjectFactory
```

有关设置环境属性的详细信息，请参见 Java VM 文档。

2a. 管理员进行的配置：自定义

此阶段通常由为其公司执行数据库系统管理或应用程序集成的人进行。目的是定义一个数据源，然后以一个逻辑名部署到命名服务器。如果需要重新配置服务器（例如，移到其它计算机、端口等等），管理员运行此配置实用程序（概述如下），并将此逻辑名重新分配给新的数据源配置。因此，客户端代码不会更改，因为它只认识此逻辑名。

```
import javax.sql.*;
import com.sybase.jdbcx.*;
.....

// create a SybDataSource, and configure it
SybDataSource ds = new com.sybase.jdbc3.jdbc.SybDataSource();
ds.setUser("my_username");
ds.setPassword("my_password");
ds.setDatabaseName("my_favorite_db");
ds.setServerName("db_machine");
ds.setPortNumber(4000);
ds.setDescription("This DataSource represents the Adaptive Server
    Enterprise server running on db_machine at port 2638.  The default
    username and password have been set to 'me' and 'mine' respectively.
    Upon connection, the user will access the my_favorite_db database on
```

```
    this server.");
Properties props = newProperties()
props.put("REPEAT_READ","false");
props.put("REQUEST_HA_SESSION","true");
ds.setConnectionProperties(props);
// store the DataSource object. Typically this is
// done by setting JNDI properties specific to the
// type of JNDI service provider you are using.
// Then, initialize the context and bind the object.
Context ctx = new InitialContext();
ctx.bind("jcbc/myASE", ds);
```

设置 DataSource 后，需要决定信息的存储位置和存储方式。SybDataSource 提供了 java.io.Serializable 和 javax.naming.Referenceable 来帮助您作出决定，但仍需管理员根据使用的 JNDI 服务提供程序来决定如何存储数据。

2b. 由客户端访问

客户端以与部署 DataSource 相同的方法设置 DataSource 对象的 JNDI 属性，从而检索该对象。客户端需要一个可转换该对象的可用对象 factory，因为它是存储 （例如，序列化）到 Java 对象的。

```
Context ctx = new InitialContext();
DataSource ds = (DataSource ctx.lookup("jcbc/myASE");
```

连接归集

参考

JDBC 2.0 可选软件包（旧称 *JDBC 2.0 标准扩展 API*）的第 6 章 “连接归集”。

相关接口

- javax.sql.ConnectionPoolDataSource
- javax.sql.PooledConnection

概述

传统数据库应用程序可与用于应用程序的每次会话的数据库创建一个连接。不过，使用此应用程序时，基于 Web 的数据库应用程序可能需要多次打开和关闭新连接。

处理基于 Web 的数据库连接的一个有效方法是使用连接归集，它可维护打开的数据库连接并管理在不同用户请求间共享的连接，从而维护性能并减少空闲连接的数目。对于每个连接请求，连接归集首先确定归集中是否有空闲连接。如果有，连接归集会返回空闲连接，而不是与数据库建立新连接。

连接归集功能由 ConnectionPoolDataSource 提供。如果使用此接口，则可归集连接。如果使用 DataSource 接口，则不能归集连接。

使用 ConnectionPoolDataSource 时，归集实现会监听 PooledConnection。当用户关闭连接或用户有错误破坏了连接时，会通知该实现。此时，归集实现会决定如何处理 PooledConnection。

如果没有连接归集，事务会：

1 创建与数据库的连接。

2 向数据库发送查询。

3 获得结果集。

4 显示结果集。

5 破坏连接。

有连接归集时，序列大致如下：

1 查看连接 “池”中是否存在未用连接。

2 如果有，则使用此连接；否则创建新连接。

3 向数据库发送查询。

4 获得结果集。

5 显示结果集。

6 将连接返回到 “池”中。用户仍然调用 “close( )”，但连接保持打开，且池会得到 close 请求的通知。

与每次客户端需要建立与数据库的连接时都创建一个新的连接相比，重新使用连接的开销要少。

为允许第三方使用连接池，jConnect 实现让 ConnectionPoolDataSource 接口产生 PooledConnections，类似于 DataSource 接口产生 Connections 的方法。

池实现使用 ConnectionPoolDataSource 的 getPooledConnection( ) 方法创建“真正”的数据库连接。然后，池实现将自己注册为针对 PooledConnection 的监听器。

目前，当客户端请求连接时，池实现就会调用可用 PooledConnection 上的 getConnection( )。当客户端完成连接并调用 close 时，池实现就会通过 ConnectionEventListener 接口得到通知，告知连接空闲，可以重用。

如果客户端因为某种原因破坏了数据库连接，池实现也会通过 ConnectionEventListener 接口得到通知，这样池实现就会将连接从池中删除。

有关详细信息，请参见 *JDBC 2.0 可选软件包*（旧称 *JDBC 2.0 标准扩展 API*）中的附录 B。

管理员进行的配置：LDAP

此方法与“用于命名数据库的 JNDI”中描述的“1a. 管理员进行的配置：LDAP”相似，只是向 LDIF 条目添加了一个附加的行。在下面的示例中，添加的代码行以粗体显示以供参考。

```
dn:servername=myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.18:ConnectionPoolDataSource
```

中间层客户端的访问

此过程初始化三个属性 （如第 78 页所示的 INITIAL_CONTEXT_FACTORY、 PROVIDER_URL 和 OBJECT_FACTORIES），并检索 ConnectionPoolDataSource 对象。有关更完整的代码示例，请参见 *sample2/SimpleConnectionPool.java*。基本区别是：

```
...
ConnectionPoolDatabase cpds = (ConnectionPoolDataSource)
    ctx.lookup("servername=myASE");
PooledConnection pconn = cpds.getPooledConnection();
```

分布式事务管理支持

此功能为 Adaptive Server 12.x 或其带 XA-Server 的 11.x 版本提供了标准 Java API，来执行分布式事务。

注释　此功能设计用于大的多层环境。

参考

请参见 *JDBC 2.0 可选软件包*（旧称 *JDBC 2.0 标准扩展 API*）的第 7 章“分布式事务”。

相关接口

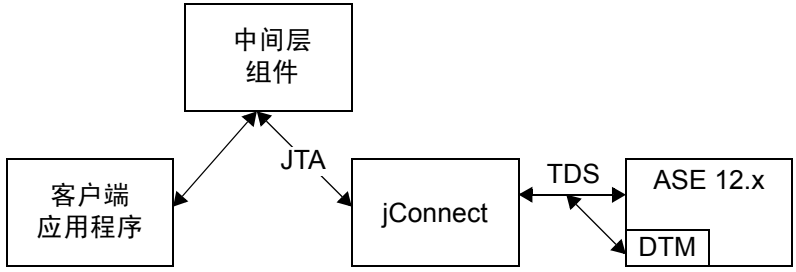
- javax.sql.XADataSource
- javax.sql.XAConnection
- javax.transaction.xa.XAResource

背景和系统要求

对于 Adaptive Server 12.0 及更高版本

- 因为在 Sybase Adaptive Server 12.0 和更高版本中 jConnect 是直接与资源管理器通信的，因此安装必须具有分布式事务管理支持。
- 任何想参与分布式事务的用户都必须具有 “dtm_tm_role”授权，否则事务将失败。
- 若要使用分布式事务，必须在 */sp* 目录中安装存储过程。请参见 *jConnect for JDBC 安装指南* 的第 1 章中的 “安装存储过程”。

图 2-2：12.x 版本的分布式事务管理支持

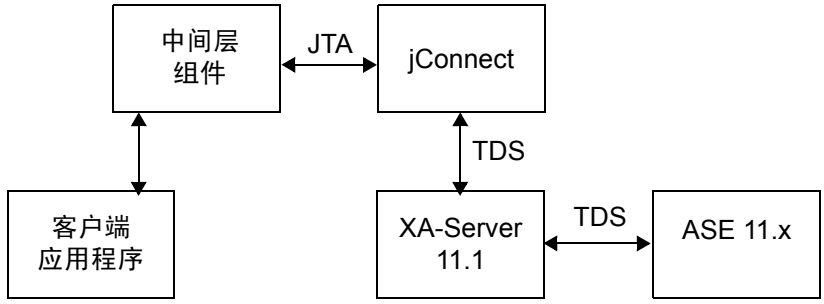


对于 Adaptive Server 11.x

jConnect 还提供标准 Java API 来以 Adaptive Server 11.x 作为数据库服务器执行分布式事务。

- 此实现仅用于 Sybase Adaptive Server 11.x 和 XA-Server 11.1。

图 2-3：11.x 版本的分布式事务管理支持



- 选择的登录不能以 master、model 或 sybsystemdb 作为缺省登录数据库。这是因为仅在用户的工作与分布式事务相关时才连接 XA-Server，而这些数据库上不允许有分布式事务。
- 不能访问元数据。虽然这样限制了客户端，但却无法限制在分布式事务边界使用的 API。

Adaptive Server 12.x 使用

管理员进行的配置：LDAP

此方法与第 86 页的 “用于命名数据库的 JNDI”中描述的 “1a. 管理员进行的配置：LDAP”相似，只是向 LDIF 条目添加了附加行。在下面的示例中，添加的代码行是以粗体显示的。

```
dn:servername:myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.18:XADataSource
```

中间层客户端的访问

此过程初始化三个属性 （INITIAL_CONTEXT_FACTORY、PROVIDER_URL 和 OBJECT_FACTORIES），并检索 XADataSource 对象。例如：

```
...
XADataSource xads = (XADatasource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection();
```

或覆盖用户名和口令的缺省设置：

```
...
XADataSource xads = (XADatasource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection("my_username","my_password");
```

Adaptive Server 11.x 使用

管理员进行的配置：LDAP

此方法与第 86 页的 “用于命名数据库的 JNDI”中描述的 “1a. 管理员进行的配置：LDAP”相似，只是向 LDIF 条目添加了三个行。

在下面的示例中，附加的代码行是以粗体显示的：

```
dn:servername:myASE, o=MyCompany, c=US
1.3.6.1.4.1.897.4.2.5:TCP#1# mymachine 4000
1.3.6.1.4.1.897.4.2.10:PACKETSIZE=1024&user=me&password=secret
1.3.6.1.4.1.897.4.2.11:userdb
1.3.6.1.4.1.897.4.2.16:userconnection
1.3.6.1.4.1.897:4.2.17:1
1.3.6.1.4.1.897.4.2.18:XADataSource
```

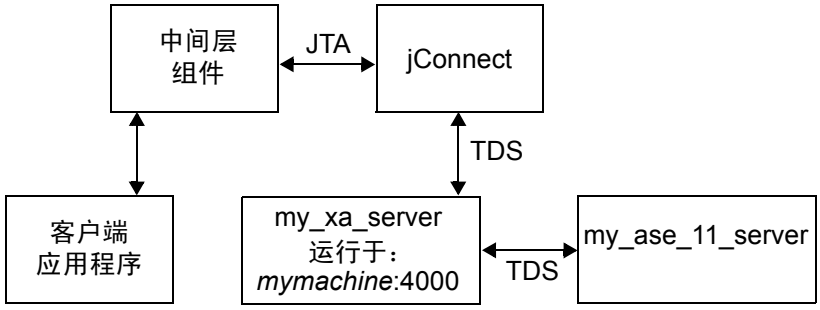
其中：

- `. . .4.2.17:1` 表示 jConnect 将与 XA-Server 连接。
- `userconnection` 对应于要使用的逻辑资源管理器 (LRM)。

XA-Server 包含一个 *xa_config* 文件，此文件包含以下条目：

```
[xa]
lrm=userconnection
server=my_ase_11_server
XAServer=my_xa_server
```

图 2-4：分布式事务管理支持示例配置



有关如何编写 *xa_config* 文件的详细信息，请参见 XA-Server 文档。

中间层客户端的访问

此过程初始化三个属性 （INITIAL_CONTEXT_FACTORY、PROVIDER_URL 和 OBJECT_FACTORIES），并检索 XADataSource 对象。例如：

```
...
XADataSource xads = (XADatasource) ctx.lookup("servername=myASE");
XAConnection xaconn = xads.getXAConnection();
```

对于 Adaptive Server 11.x，*不能* 替换缺省用户名和口令，也就是说不能调用下面的语句：

```
xads.getXAConnection("my_username","my_password");
```

因为 *lrm* 是与特定用户名和口令关联的。

对 JDBC 标准的限制与解释

本节讨论 JDBC 的 jConnect 实现是怎样偏离 JDBC 1.x,、 2.x 和 3.0 标准的。涉及以下主题：

- 使用 JDBC 3.0 方法占位模块
- 使用 Connection.isClosed 和 IS_CLOSED_TEST
- 对未处理的结果使用 Statement.close
- 调整多线程
- 使用 ResultSet.getCursorName
- 使用具有较大参数值的 setLong
- 使用 COMPUTE 语句
- 执行存储过程

使用 JDBC 3.0 方法占位模块

jConnect 6.x 是使用 JDK 1.4 （包括 JDBC 3.0 标准及所有相关方法和接口）编译的。jConnect 6.05 尚未实现任何 JDBC 3.0 方法。此功能将在未来的 EBF 版本中逐步实现。若要使用 jConnect 6.05 调用 JDBC 3.0 方法，则会产生表示方法尚未实现的 SQLException。有关最新 EBF 和软件维护的信息，请浏览 http://www.sybase.com/support 上的 “Sybase Support Page”（Sybase 技术支持页）。

使用 Connection.isClosed 和 IS_CLOSED_TEST

根据 JDBC 2.1 规范的第 11.1 节：

“仅能确保 Connection.isClosed 方法在调用 Connection.close 后返回“ture”。通常情况下不能调用 Connection.isClosed 来确定数据库连接有效还是无效。典型客户端可通过捕获在试图进行操作时抛出的例外来确定连接无效。”

jConnect 对 isClosed 方法提供了一种缺省的解释，这与规范中定义的行为不同。调用 Connection.isClosed 时，jConnect 首先验证是否已对此连接调用了 Connection.close。如果 close 已被调用，则 jConnect 为 isClosed 返回 “true”。

但是，如果 Connection.close 未被调用，则 jConnect 接下来将尝试在数据库上执行 sp_mda 存储过程。 sp_mda 存储过程是 jConnect 用户在数据库上使用 jConnect 时必须安装的元数据的一部分。

调用 sp_mda 的目的是使 jConnect 可执行已知（或至少是期望）驻留在数据库服务器上的过程。如果存储过程正常执行，jConnect 则为 isClosed 返回“false”，因为我们已经验证过数据库连接是有效的，且正在工作。但是，如果对 sp_mda 的调用导致抛出 SQLException，则 jConnect 会捕获例外并为 isClosed 返回“true”，因为连接似乎有问题。

如果希望强制 jConnect 更严格地遵循 isClosed() 的标准 JDBC 行为，可通过将 IS_CLOSED_TEST 连接属性设置为特殊值“INTERNAL”达到这个目的。INTERNAL 设置表示仅当 Connection.close 已被调用，或 jConnect 已检测到 IOException 禁用了连接时才为 isClosed 返回“true”。

也可指定在 isClosed 被调用时使用除 sp_mda 外的其它查询。例如，如果想让 jConnect 在 isClosed 被调用时尝试 select 1，可将 IS_CLOSED_TEST 连接属性设置为 select 1。

对未处理的结果使用 Statement.close

关于首先调用 Statement.execute，然后调用同一语句对象的 close，而不处理 Statement 返回的任何结果 （更新计数和 ResultSets）时驱动程序的行为， JDBC 规范的表述很模糊。

例如，假定数据库上有一个进行七次行插入的存储过程。应用程序于是使用 Statement.execute 执行该存储过程。在这种情况下， Sybase 数据库向应用程序返回七个更新计数 （为每个插入的行返回一个计数）。根据常规 JDBC 应用程序逻辑，应该使用 getMoreResults、 getResultSet 和 getUpdateCount 方法在循环中处理这些更新计数。这些在 java.sql.* 软件包的 javadocs 中的 *java.sun.com* Web 站点解释得很清楚。

不过，应用程序程序员可能会在通览返回的所有更新计数前错误地选择调用 Statement.close。在这种情况下， jConnect 会向数据库发送 cancel，而这会造成无法预料的和不希望的负面影响。

在这个特殊的示例中，如果应用程序在数据库完成插入前调用 Statement.close，则数据库可能不会执行所有插入。例如，它可能会在插入 5 行后就停止，因为存储过程尚未结束就在数据库上取消了。

这种情况下就不会向您报告丢失的插入。在仍然具有未处理的结果时试图关闭 Statement 时， jConnect 的更高版本会抛出 SQLException，但在这之前，仍然强烈建议 jConnect 程序员遵循以下指南：

- 调用 Statement.close 时，如果您未处理完所有结果 （更新计数和 ResultSets）时向服务器发送了 cancel。在只执行 select 语句的情况下，这完全可以。但是，在执行 insert/update/delete 操作的情况下，这会导致某些操作不能如期望的那样完成。
- 因此，执行除 select 之外的其它操作时，切勿在还有未处理结果时调用 close。
- 因此，如果调用 Statement.execute，请用 getUpdateCount、getMoreResults 和 getResultSet 方法确认处理了所有结果。

调整多线程

如果多个线程同时调用同一 Statement 实例 （CallableStatement 或 PreparedStatement）上的方法 （Sybase 不建议使用），您必须手动同步 Statement 上的对这些方法的调用； jConnect 不能自动进行此同步。

例如，如果有两个线程在同一 Statement 实例上操作 （一个线程发送查询，另一个线程处理警告），就必须同步 Statement 上的对这些方法的调用，否则会发生冲突。

使用 *ResultSet.getCursorName*

有些 JDBC 驱动程序可为任何 SQL 查询生成游标名，这样总是可以返回一个字符串。不过，调用了 ResultSet.getCursorName 时 jConnect 不会返回名称，除非您：

- 调用相应的 Statement 上的 setFetchSize 或 setCursorName，或者
- 将 SELECT_OPENS_CURSOR 连接属性设置为 “true”，且查询的格式为 SELECT... FOR UPDATE。例如：

```
select au_id from authors for update
```

如果不调用相应的 Statement 上的 setFetchSize 或 setCursorName，或不将 SELECT_OPENS_CURSOR 连接属性设置为 “true”，则会返回 null。

根据 JDBC 2.0 API （请参见第 11 章 “说明”），所有其它 SQL 语句都不需要打开游标及返回名称。

有关如何在 jConnect 中使用游标的详细信息，请参见第 50 页的 “对结果集使用游标”。

使用具有较大参数值的 *setLong*

PreparedStatement.setLong 方法的实现可将参数值设置为 SQL BIGINT 数据类型。大多数 Adaptive Server 数据库都不具有 8 字节 BIGINT 数据类型。如果一个参数值需要超过 4 字节的 BIGINT，使用 setLong 可导致溢出例外。

支持的新数据类型

jConnect 支持下列新数据类型：

- bigint — 一种精确数值数据类型，供在现有 int 类型的范围不足时使用。
- unsigned int — 无符号形式的精确数值整数数据类型：unsignedsmallint、 unsignedint 和 unsignedbigint。
- unitext — 可变长度数据类型，用于 Unicode 字符。

B*igint* 数据类型

Sybase 支持 bigint，它是一种 64 位整数数据类型，作为本机 ASE 数据类型受到支持。在 Java 中，此数据类型映射到 Java 数据类型 long。若要作为参数使用此数据类型，可以调用 PreparedStatement.setLong(int index, long value)，jConnect 即会将数据作为 bigint 发送给 ASE。在从 bigint 列进行检索时，可以使用 ResultSet.getLong(int index) 方法。

Unitext 数据类型

在使用 unitext 数据类型方面， jConnect 中并没有发生 API 更改。使用 unitext 列时， jConnect 可以在内部处理来自 ASE 的数据的存储和检索。

Unsigned int 数据类型

在此版本中， ASE 引入了 *unsigned bigint*、 *unsigned int* 和 *unsigned smallint*，作为本机 ASE 数据类型。由于在 Java 中并没有与之相对应的无符号数据类型，因此，如果要正确处理数据，必须用 set 和 get 语句设置和获取下一个较大整数。例如，如果要从 *unsigned int* 中检索数据，使用 Java 数据类型 int 则会太小，不能包含大的正值，因此，ResultSet.getInt (int index) 可能会返回不正确的数据或抛出例外。若要正确处理数据，应该用 get 语句获取下一个较大整数值 ResultSet.getLong()。可以按照下表用 set 或 get 语句设置或获取数据。

ASE 数据类型	Java 数据类型
unsigned smallint	setInt(), getInt()
unsigned int	setLong(), getLong()
unsigned bigint	setBigDecimal(), getBigDecimal()

使用 COMPUTE 语句

jConnect 不支持计算行。实际上，如果查询包含计算行，结果会被自动取消。例如，以下语句会被拒绝：

```
SELECT name FROM sysobjects
WHERE type="S" COMPUTE COUNT(name)
```

若要避免此问题，请用下面的代码替代：

```
SELECT name from sysobjects WHERE type="S"
SELECT COUNT(name) from sysobjects WHERE type="S"
```

执行存储过程

- 如果执行用问号代表参数值的 CallableStatement 对象中一个存储过程，会比对参数既使用问号又使用实际值得到更好的性能。而且，如果混合使用实际值和问号，就不能对存储过程使用输出参数。

  下面的示例将 *sp_stmt* 创建为 CallableStatement 对象，以执行存储过程 MyProc：

  ```
  CallableStatement sp_stmt = conn.prepareCall(
     "{call MyProc(?,?)}");
  ```

  MyProc 中的两个参数用问号表示。可以使用 CallableStatement 接口中的 registerOutParameter 方法将其中的一个或所有两个参数注册为输出参数。

  在下面的示例中，*sp_stmt2* 是 CallableStatement 对象，用于执行存储过程 MyProc2。

  ```
  CallableStatement sp_stmt2 = conn.prepareCall(
     {"call MyProc2(?,'javelin')}");
  ```

  在 *sp_stmt2* 中，一个参数值为实际值，另一个为问号。不能将两个参数中的任一个注册为输出参数。

- 若要使用参数的名称绑定来通过 RPC 命令执行存储过程，请使用以下两个过程之一：
  - 使用语言命令，用 PreparedStatement 类将输入参数直接从 Java 变量传递给语言命令。如以下代码段所示：

```
// Prepare the statement
System.out.println("Preparing the statement...");
String stmtString = "exec " + procname + " @p3=?, @p1=?";
PreparedStatement pstmt = con.preparedStatement(stmtString);

// Set the values
pstmt.setString(1, "xyz");
pstmt.setInt(2, 123);

// Send the query
System.out.println("Executing the query...");
ResultSet rs = pstmt.executeQuery();
```

  - 对于 jConnect 6.05 及更高版本，请使用 com.sybase.jdbcx.SybCallableStatement 接口，如下面的示例所示：

```
import com.sybase.jdbcx.*;
....
// prepare the call for the stored procedure to execute as an RPC
String execRPC = "{call " + procName + " (?, ?)}";
SybCallableStatement scs = (SybCallableStatement)
con.prepareCall(execRPC);

// set the values and name the parameters
// also (optional) register for any output parameters
scs.setString(1, "xyz");
scs.setParameterName(1, "@p3");
scs.setInt(2, 123);
scs.setParameterName(2, "@p1");

// execute the RPC
// may also process the results using getResultSet()
// and getMoreResults()

// see the samples for more information on processing results
ResultSet rs = scs.executeQuery();
```

第 3 章

安全性

本章描述 jConnect 的安全性问题。

主题	页码
概述	103
SSL	104
Kerberos	104

概述

jConnect 版本 6.x 提供了以下选项来保护客户端 / 服务器通信：

- *SSL* — 使用 SSL 加密客户端和服务器应用程序之间包括登录交换在内的通信。
- *Kerberos* — 使用 Kerberos 为 Adaptive Server Enterprise 鉴定 Java 应用程序或 Java 应用程序的用户，而不需要通过网络发送用户名或口令。还可以使用 Kerberos 设置单点登录 (SSO) 环境，并提供 Java 应用程序的数字标识和 Adaptive Server Enterprise 的数字标识之间的相互鉴定。

  注释 Kerberos 可用于加密通信并提供数据完整性检查，但对 jConnect 6.0 尚未实现此功能。

Kerberos 和 SSL 也可一起使用，这样可以提供 SSO 以及客户端和服务器应用程序之间传输的数据加密的优点。

限制

Kerberos 和 SSL 仅能用于 Adaptive Server Enterprise 版本 12.0 及更高版本。 Adaptive Server Anywhere 当前既不支持 SSL 安全性，也不支持 Kerberos 安全性。

Sybase 建议在 jConnect 中使用 SSL 或 Kerberos 前先阅读其相关文档。本章中的信息假定要使用的服务器已经进行配置，可正常使用 SSL、Kerberos 或上述两者。

有关 Kerberos、SSL 和配置 Adaptive Server Enterprise 的详细信息，请参见第 115 页的 “相关文档”。还请参见有关设置 Kerberos 的白皮书。该文档的 URL 可在 *jConnect for JDBC 发行公告* 中找到。

SSL

有关在 jConnect 中使用 SSL 的详细信息，请参见第 31 页的 “实现自定义套接字插件”。

Kerberos

Kerberos 是一种网络鉴定协议，对客户端 / 服务器应用程序的鉴定使用了加密。 Kerberos 为用户和系统管理员带来了以下优点：

- Kerberos 数据库可用作用户的集中仓库。
- Kerberos 便于建立 SSO 环境，在此环境中用户系统登录可提供访问数据库所必需的认证。
- Kerberos 是一种 IETF 标准。 Kerberos 的不同实现间的互操作是可能的。

为 Kerberos 配置 jConnect 应用程序

为 jConnect 6.0 配置 Kerberos 前，应确保拥有以下条件：

- JDK 1.4 或更高版本 （推荐使用 JDK 1.4.2 或更高版本）
- Java 通用安全服务 (GSS) Manager：
  a　缺省 Sun GSS Manager （ JDK 的一部分），或
  b　Wedgetail JCSI Kerberos 2.6 或更高版本，或
  c　CyberSafe TrustBroker Application Security Runtime Library 3.1.0 或更高版本，或
  d　来自其它供应商的 GSS Manager 实现。
- 在服务器端得到支持，可与 GSS 互操作，同时在客户端也得到支持，可与 GSSManager 互操作的 KDC。

若要启用 jConnect 的 Kerberos 登录，请使用以下步骤。

❖ **为 jConnect 配置 Kerberos**

1　将 REQUEST_KERBEROS_SESSION 属性设置为 “ true”。

2　将 SERVICE_PRINCIPAL_NAME 属性设置为正在运行的 Adaptive Server Enterprise 的名称。通常就是服务器启动时用 -s 选项设置的名称。服务的主管名也必须用 KDC 注册。如果不为 SERVICE_PRINCIPAL_NAME 属性设置任何值，缺省情况下 jConnect 会使用客户端计算机的主机名。

3　也可以选择设置 GSSMANAGER_CLASS 属性。

有关 REQUEST_KERBEROS_SESSION 和 SERVICE_PRINCIPAL_NAME 属性的详细信息，请参见第 2 章“编程信息”。关 GSSMANAGER_CLASS 属性的详细信息，请参见 “GSSMANAGER_CLASS 连接属性”。

GSSMANAGER_CLASS 连接属性

使用 Kerberos 时，jConnect 依赖于实现通用安全服务 (GSS) API 的几个 Java 类。这个功能的大部分都是由 org.ietf.jgss.GSSManager 类提供的。

供应商实现

Java 允许供应商提供自己的 GSSManager 类实现。如 Wedgetail Communications 和 CyberSafe Limited 提供的实现就是供应商提供的 GSSManager 实现。用户可配置供应商编写的 GSSManager 类，以使其在特定的 Kerberos 环境中工作。供应商提供的 GSSManager 类提供的与 Windows 的互操作性可能比标准的 Java GSSManager 类提供的还要多。

在使用供应商提供的 GSSManager 实现前，一定要阅读供应商文档。供应商使用属性设置而不是用于 Kerberos 的标准 Java 系统属性，而且可能会定位领域名和密钥分发中心 (KDC) 条目而不使用配置文件。

设置 GSSMANAGER_CLASS

通过设置 GSSMANAGER_CLASS 连接属性可以在 jConnect 中使用供应商提供的 GSSManager 实现。有两种设置此属性的方法：

- 创建一个 GSSManager 实例，并将此实例设置为 GSSMANAGER_CLASS 属性的值。
- 将 GSSMANAGER_CLASS 属性的值设置为指定 GSSManager 对象的全限定类名的字符串。 jConnect 使用该字符串调用 `Class.forName().newInstance()`，并将返回的对象转换为 GSSManager 类。

在任一情况下，应用程序 CLASSPATH 变量必须包含供应商实现的类和 *.jar* 文件的位置。

注释 如果不设置 GSSMANAGER_CLASS 连接属性，jConnect 将使用 org.ietf.jgss.GSSManager.getInstance 方法装载缺省 Java GSSManager 实现。

使用 GSSMANAGER_CLASS 连接属性传递全限定类名时， jConnect 将调用 GSSManager 的不带参数的构造方法。此处例示了供应商实现的一个处于缺省配置的 GSSManager，因此您无法控制 GSSManager 对象的精确配置。如果创建自己的 GSSManager 实例，就可以使用构造方法参数来设置配置选项。

jConnect 使用 GSSMANAGER_CLASS 的方式

首先， jConnect 检查要用于 Kerberos 鉴定的 GSSManager 类对象的 GSSMANAGER_CLASS 的值。

如果已将 GSSMANAGER_CLASS 的值设置为字符串而不是类对象，则 jConnect 将使用该字符串创建指定类的实例，并在 Kerberos 鉴定中使用新实例。

如果已将 GSSMANAGER_CLASS 的值设置为 GSSManager 类对象和字符串以外的其它内容，或如果 jConnect 遇到 ClassCastException，则 jConnect 将抛出指明问题的 SQLException。

示例

以下示例说明了将 GSSMANAGER_CLASS 连接属性设置为全限定类名时，如何创建自己的 GSSManager 实例，以及如何让 jConnect 创建 GSSManager 对象。这两个示例都使用了 Wedgetail GSSManager。

- **示例：创建自己的 GSSManager 实例**

  1　在自己的应用程序代码中例示 GSSManager。例如：

```
GSSManager gssMan =  new com.dstc.security.kerberos.gssapi.GSSManager();
```

  此示例使用了不带参数的缺省构造方法。也可以使用允许设置各种配置选项的其它供应商提供的构造方法。

  2　将新 GSSManager 实例传递给 GSSMANAGER_CLASS 连接属性。例如：

```
Properties props = new Properties();
props.put("GSSMANAGER_CLASS", gssMan);
```

  3　在连接中使用这些连接属性 （包括 GSSMANAGER_CLASS）。例如：

```
Connection conn = DriverManager.getConnection (url, props);
```

❖ **示例：将字符串传递给 GSSMANAGER_CLASS**

1 在应用程序代码中，创建一个指定 GSSManager 对象的全限定类名的字符串。例如：

```
String gssManClass = "com.dstc.security.kerberos.gssapi.GSSManager";
```

2 将该字符串传递给 GSSMANAGER_CLASS 连接属性。例如：

```
Properties props = new Properties();
props.put("GSSMANAGER_CLASS", gssManClass);
```

3 在连接中使用这些连接属性 （包括 GSSMANAGER_CLASS）。例如，

```
Connection conn = DriverManager.getConnection (url, props);
```

设置 Kerberos 环境

本节提供了一些设置环境的建议，来用三个不同的 Kerberos 实现使用 jConnect 6.05：

- CyberSafe
- MIT
- Microsoft Active Directory

注释 阅读本节前，请参见有关设置 Kerberos 的白皮书。该文档的 URL 可在 *jConnect for JDBC 发行公告* 中找到。

CyberSafe

加密密钥

创建 CyberSafe KDC 中要由 Java 使用的主管时应指定数据加密标准 (DES) 密钥。 Java 参考实现不支持三倍数据加密标准 (3DES) 密钥。

注释 如果是在 CyberSafe KDC 中使用 CyberSafe 且设置了 GSSMANAGER_CLASS 属性，就可以使用 3DES 密钥。

地址映射和领域信息

CyberSafe Kerberos 不使用 *krb5.conf* 配置文件。缺省情况下， CyberSafe 使用 DNS 记录来定位 KDC 地址映射和领域信息。或者， CyberSafe 分别在 *krb.conf* 和 *krb.realms* 文件中定位 KDC 地址映射和领域信息。有关详细信息，请参见 CyberSafe 文档。

如果使用的是标准 Java GSSManager 实现，还必须创建 *krb5.conf* 文件以供 Java 使用。 CyberSafe *krb.conf* 文件的格式与 *krb5.conf* 文件的格式不同。根据 Sun 手册页或 MIT 文档中的指定创建 *krb5.conf* 文件。如果正在使用 CyberSafe GSSManager，就不需要 *krb5.conf* 文件。

有关 *krb5.conf* 文件的示例，请参见有关设置 Kerberos 的白皮书。该文档的 URL 可在 *jConnect for JDBC 发行公告* 中找到。

Solaris

在 Solaris 上使用 CyberSafe 客户端库时，应确保库搜索路径在任何其它 Kerberos 库前包含 CyberSafe 库。

MIT

加密密钥

创建 MIT KDC 中要由 Java 使用的主管时应指定 DES 密钥。Java 参考实现不支持 3DES 密钥。

如果计划只使用标准 Java GSSManager 实现，应指定以下类型的加密密钥：`des-cbc-crc` 或者 `des-cbc-md5`. 按如下方法指定加密类型：

```
des-cbc-crc:normal
```

这里 `normal` 是密钥 salt 的类型。也可以使用其它 salt 类型。

注释　如果使用的是 Wedgetail GSSManager，就可以在 MIT KDC 中创建以下类型的主管：`des3-cbc-sha1-kd`.

Microsoft Active Directory

SSO

使用 Java 参考实现的 SSO 仅在 Windows 2000 和 Windows XP 客户端可用，不可用于使用 Microsoft Active Directory 的 Windows NT 客户端。不过， SSO 可以用于使用供应商提供的 GSSManager （例如由 CyberSafe 提供）的 NT 客户端。

用户帐号和服务主管

确保已在 Active Directory 中为用户主管 （用户）和服务主管 （代表数据库服务器的帐号）设置了帐号。用户主管和服务主管都应在 Active Directory 中创建为 “Users”。

加密

如果要使用 Java 参考 GSS Manager 实现，就必须对用户和服务主管使用 DES 加密。

❖ **设置 DES 加密**

1 在 Active Directory “用户”列表中右击特定用户主管或服务主管的名称。

2 选择 “属性”。

3 单击 “帐号”选项卡。出现 “帐号选项”列表。

4 为用户主管和服务主管指定应使用的 DES 加密类型。

客户端计算机

如果计划使用 Java 参考实现在 Windows 2000 客户端设置 SSO 环境，可能需要根据下面的 URL 中指定的指导修改 Windows 注册表：

http://support.microsoft.com/default.aspx?scid=kb;en-us;308339。

配置文件

在 Windows 上， Kerberos 配置文件称为 *krb5.ini*。缺省情况下， Java 会在 *C:\WINNT\krb5.ini* 中查找 *krb5.ini*。 Java 允许指定该文件的位置。*krb5.ini* 的格式与 *krb5.conf* 的格式相同。

有关 *krb5.conf* 文件的示例，请参见有关设置 Kerberos 的白皮书。该文档的 URL 可在 *jConnect for JDBC 发行公告* 中找到。

有关 Microsoft Active Directory 的 Kerberos 的详细信息，请参见下面的文档：

www.microsoft.com/windows2000/techinfo/planning/security/kerbsteps.asp

示例应用程序

在 *jConnect-6_0/sample2* 目录中提供了以下两个注释的代码示例，说明了如何建立与 Adaptive Server Enterprise 的 Kerberos 连接：

- *ConnectKerberos.java* — 到 Adaptive Server Enterprise 的简单的 Kerberos 登录
- *ConnectKerberosJAAS.java* — 更详细的示例，显示了如何在应用程序 / 服务器代码中实现 Kerberos 登录

ConnectKerberos.java

若要运行 *ConnectKerberos.java* 示例应用程序，请使用以下步骤。

❖ **运行 *ConnectKerberos.java***

1　确保计算机具有有效 Kerberos 认证。该任务根据计算机和环境的不同而不同。

Windows — 通过使用 Kerberos 鉴定成功登录，可以为运行在 Active Directory 环境中的 Windows 2000 或 XP 计算机建立 Kerberos 认证。

UNIX 或 Linux — 可以使用 Kerberos 客户端的 kinit 实用程序为 UNIX 或 Linux 计算机建立 Kerberos 认证。如果未使用 kinit 获取初始认证，试图运行示例应用程序时，会得到输入用户名和口令的提示。

注释　Sun JDK 只能使用 DES_CBC_MD5 和 DES_CBC_CRC 加密类型。通过使用第三方软件和设置 GSSMANAGER_CLASS 可能可以使用其它加密类型。

2　确定计算机认证的位置。

Windows — 对于运行在 Active Directory 环境中的 Windows 2000 或 XP 计算机， Kerberos 认证存储在内存中的票据高速缓存中。

UNIX 或 Linux — 对于使用 Sun Java、CyberSafe、Solaris 或 Kerberos 的 MIT 实现的 UNIX 或 Linux 计算机，缺省情况下 kinit 将认证放在 */tmp/krb5cc_{user_id_number}* 中，其中 *{user_id_number}* 对于您的用户名来说是唯一的。

如果认证位于其它位置，必须通过设置 ticketCache 属性在 *sample2/exampleLogin.conf* 文件中指定该位置。

3 向 Java 参考实现指定 KDC 计算机的缺省领域和主机名。 Java 可从 *krb5.conf* 或 *krb5.ini* 配置文件或从 Java System 属性获取该信息。如果使用供应商提供的 GSS Manager 实现，则此实现可能从 DNS SRV 记录获取主机及领域信息。

Sybase 推荐使用 Kerberos 配置文件，它允许对 Kerberos 环境进行更多控制，包括向 Java 指定鉴定期间请求加密的类型的能力。

注释 在 Linux 上， Java 参考实现在 */etc/krb5.conf* 中查找 Kerberos 配置文件。

如果不使用 Kerberos 配置文件，且未将 Kerberos 配置设置为使用 DNS SRV 记录，就可以使用 java.security.krb5.realm 和 java.security.krb5.kdc 系统属性指定领域和 KDC。

4 编辑 *ConnectKerberos.java* 以使连接 URL 指向您的数据库。

5 编译 *ConnectKerberos.java*。
确保使用 JDK 版本 1.4 或更高版本。Sybase 建议使用 JDK 1.4.2 或更高版本。通览源代码注释，并确保已在 CLASSPATH 环境变量中指定了 jConnect 安装的 *jconn3.jar*。

6 执行 ConnectKerberos.class：

```
java ConnectKerberos
```

确保使用 1.4.2 java 可执行代码。示例应用程序输出说明建立了成功的连接并执行下面的 SQL：

```
select 1
```

- *若要执行示例而不使用 Kerberos 配置文件*，请使用以下命令：

  ```
  java -Djava.security.krb5.realm=your_realm
  -Djava.security.krb5.kdc=your_kdc
  ConnectKerberos
  ```

  其中， *your_realm* 是缺省领域，而 *your_kdc* 是 KDC。

- *如有必要，可在调试模式下运行示例应用程序，*来查看 Java Kerberos 层的调试结果：

  ```
  java -Dsun.security.krb5.debug=true
  ConnectKerberos
  ```

也可以使用位于 *jConnect-6_0/classes* 目录的 IsqlApp （isql 的 Java 版本）来建立 Kerberos 连接：

```
java IsqlApp -S jdbc:sybase:Tds:hostName:portNum
-K service_principal_name
-F path_to_JAAS_login_module_config_file
```

有关使用 IsqlApp 的详细信息，请参见第 B 章“jConnect 示例程序”。

互操作性

表 3-1 显示了 KDC、GSS 库和平台的组合，Sybase 在这些平台上成功地建立了到 Adaptive Server Enterprise 的连接。缺少任何特定组合并不表示不能与该组合建立连接。互操作性测试一直在进行，可以在 jConnect for JDBC Web 站点找到最新的状态：

http://www.sybase.com/products/middleware/jconnectforjdbc

表 3-1：互操作性组合

客户端平台	KDC	GSSManager	GSS C 库 [a]	ASE 平台
Solaris 8[b]	CyberSafe	Java GSS	CyberSafe	Solaris 8
Solaris 8	Active Directory[c]	Java GSS	CyberSafe	Solaris 8
Solaris 8	MIT	Java GSS	CyberSafe	Solaris 8
Solaris 8	MIT	Wedgetail GSS[d]	MIT	Solaris 8
Solaris 8	CyberSafe	Wedgetail GSS[e]	CyberSafe	Solaris 8
Windows 2000	Active Directory	Java GSS	CyberSafe	Solaris 8
Windows XP	Active Directory	Java GSS[f]	CyberSafe	Solaris 8

a. 这些是 Adaptive Server Enterprise 用于提供 GSS 功能的库。
b. 表中所有 Solaris 8 平台均为 32 位。
c. 表中所有 Active Directory 条目均指运行于 Windows 2000 上的 Active Directory 服务器。若要获得 Kerberos 互操作性， Active Directory 用户必须设置为 “为此帐号使用 DES 加密类型”。
d. 使用 Wedgetail JCSI Kerberos 2.6。加密类型为 3DES。
e. 使用 Wedgetail JCSI Kerberos 2.6。加密类型为 DES。
f. Java 1.4.x 有一个错误，它要求客户端使用 `System.setProperty("os.name", "Windows 2000");` 来确保 Java 可在 Windows XP 客户端找到内存中的认证。

Sybase 建议使用这些库的最新版本。如果想要使用旧版本，或者非 Sybase 产品有问题，请联系供应商。

加密类型

Sun 提供的标准 Java GSS 实现仅支持 DES 加密。若要使用 3DES、RC4-HMAC、AES-256 或 AES-128 加密标准，就必须使用 CyberSafe 或 Wedgetail GSSManager。

有关 Wedgetail 和 CyberSafe 的详细信息，请参见各自的文档。

故障排除

本节提供进行 Kerberos 安全性故障排除时要考虑的问题的文档资料。

Kerberos

进行 Kerberos 安全性问题故障排除时要考虑以下内容：

- Java 参考实现仅支持 DES 加密类型。必须配置 Active Directory 和 KDC 主管使用 DES 加密。
- SERVICE_PRINCIPAL_NAME 属性的值必须设置为启动数据服务器时用 -s 选项指定的同一名称。
- 检查 *krb5.conf* 和 *krb5.ini* 文件。对于 CyberSafe 客户端，检查 *krb.conf* 和 *krb.realms* 文件或 DNS SRV 记录。
- 可在 JAAS 登录配置文件中将 debug 属性设置为 “true”。
- 可在命令行将 debug 属性设置为 “true”：

  ```
  -Dsun.security.krb5.debug=true
  ```

- JAAS 登录配置文件提供了多个可进行设置的选项，以满足各种特殊需要。有关此配置文件的信息，请参见以下链接：

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/LoginConfigFile.html

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html

有关 JAAS 和 Java GSS API 故障排除的详细信息，请参见下面的链接：

http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/Troubleshooting.html

相关文档

以下文档提供了有关 Kerberos 安全性的详细信息。

- 有关 JAAS 和 Java GSS API 的 Java 教程：

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/index.html

- MIT Kerberos 文档和下载站点：

  http://web.mit.edu/kerberos/www/index.html

- CyberSafe Limited：

  http://www.cybersafe.ltd.uk

- 有关 Windows/Kerberos 互操作性的 CyberSafe Limited 文档：

  www.cybersafe.ltd.uk/docs_cybersafe/Kerberos%20Interoperability%20-%20Microsoft%20W2k%20&%20ActiveTRUST.pdf

- Wedgetail Communications Kerberos 常见问题解答：

  http://www.wedgetail.com/jcsi/kerberos/FAQ.html

- 有关 Windows 如何实现鉴定的说明，包括关于 Active Directory Kerberos 的信息：

  http://www.windowsitlibrary.com/Content/617/06/1.html

- Windows Kerberos 的说明：

  http://www.microsoft.com/windows2000/techinfo/howitworks/security/kerberos.asp

- Kerberos RFC 1510：

  http://www.linuxdig.com/rfc/individual/1510.php

第 4 章

故障排除

本章描述了对 jConnect 使用过程中可能出现问题的解决方案。

主题	页码
使用 jConnect 进行调试	117
捕获 TDS 通信	121
解决连接错误	122
管理 jConnect 应用程序所使用的内存	124
解决存储过程错误	125
解决自定义套接字执行错误	126

使用 jConnect 进行调试

jConnect 包括一个 Debug 类，该类包含一组调试函数。 Debug 方法包括多个断言函数、跟踪函数和计时器函数，用于定义调试过程的范围以及调试结果的输出位置。

jConnect 安装程序还包括一组完整的具有调试功能的类。这些类位于 jConnect 安装目录的 *devclasses* 子目录下。为了进行调试，必须重定向 CLASSPATH 环境变量以引用调试模式运行期类 (*devclasses/jconn3d.jar*) 目录，而不是引用标准的 jConnect *classes* 目录。也可以在运行 Java 程序时，通过将 -classpath 参数显式提供给 java 命令来实现。

获取 Debug 类的一个实例

若要使用 jConnect 调试功能，应用程序必须导入 Debug 接口并通过调用 SybDriver 类的 getDebug 方法获取 Debug 类的一个实例。

```
import com.sybase.jdbcx.Debug
 import.com.sybase.jdbcx.SybDebug
 //
 ...
 SybDriver sybDriver = (SybDriver)

Class.forName("com.sybase.jdbc3.jdbc.SybDriver").newIn
stance();
Debug sybdebug = sybDriver.getDebug();
 ...
```

在应用程序中打开调试程序

若要使用 Debug 对象的 debug 方法打开应用程序中的调试程序，请添加以下调用：

```
sybdebug.debug(true, [classes], [printstream]);
```

classes 参数是一个字符串，其中列出了要调试的特定类 （以冒号分隔各个类）。例如：

```
sybdebug.debug(true,"MyClass")
```

和

```
sybdebug.debug(true,"MyClass:YourClass")
```

在类字符串中使用 “STATIC”为 jConnect 中的所有静态方法以及指定的类打开调试程序。例如：

```
sybdebug.debug(true,"STATIC:MyClass")
```

可以指定 “ALL”为所有类打开调试程序。例如：

```
sybdebug.debug(true,"ALL");
```

printstream 参数是可选的。如果未指定 printstream 参数，调试结果将传送到 DriverManager.setLogStream 所指定的输出文件中。

在应用程序中关闭调试程序

若要关闭调试程序，请添加以下调用：

```
sybdebug.debug(false);
```

为调试程序设置 CLASSPATH

在运行启用调试功能的应用程序之前，需重新定义 CLASSPATH 环境变量，使其指向 jConnect 安装目录下的 */devclasses* 子目录：

- 在 UNIX 系统中，使用 *$JDBC_HOME/devclasses/jconn3d.jar* 替换 *$JDBC_HOME/classes/jconn3.jar*。
- 在 Windows 系统中，使用 *%JDBC_HOME%\devclasses\jconn3d.jar* 替换 *%JDBC_HOME%\classes\jconn3.jar*。

使用 Debug 方法

若要自定义调试过程，需添加到其它 Debug 方法的调用。

在这些方法中，第一个 （对象）参数通常是 this，用以指定调用对象。如果这些方法都是静态的，对象参数需使用 null。

- println

  如果已启用调试程序并且对象包含在要调试的类列表中，请使用该方法定义要在输出日志中输出的消息。调试输出传送到 *sybdebug*.debug 所指定的文件中。

  语法为：

  ```
  sybdebug.println(object,message string);
  ```

  例如：

  ```
  sybdebug.println(this,"Query: "+ query);
  ```

  在输出日志中产生如下类似的消息：

  ```
  myApp(thread[x,y,z]): Query: select * from authors
  ```

- assert

  使用该方法声明条件，并当该条件不满足时抛出一个运行期例外。如果条件不满足，也可以定义消息在输出日志中输出。语法为：

  ```
  sybdebug.assert(object,boolean condition,message
     string);
  ```

  例如：

  ```
  sybdebug.assert(this,amount<=buf.length,amount+"
     too big!");
  ```

  如果 “amount”超出了 buf.length 的值，就会在输出日志中产生如下类似的消息：

  ```
  java.lang.RuntimeException:myApp(thread[x,y,z]):
  Assertion failed: 513 too big!
  at jdbc.sybase.utils.sybdebug.assert(
  sybdebug.java:338)
  at myApp.myCall(myApp.java:xxx)
  at .... more stack:
  ```

- startTimer
  stopTimer

  使用这些方法启动和终止计时器，计时器用于测量事件所占用的时间 （以毫秒计）。该方法为每个对象保留一个计时器，并为所有静态方法保留一个计时器。启动计时器的语法为：

  ```
  sybdebug.startTimer(object);
  ```

  终止计时器的语法为：

  ```
  sybdebug.stopTimer(object,message string);
  ```

  例如：

  ```
  sybdebug.startTimer(this);
  stmt.executeQuery(query);
  sybdebug.stopTimer(this,"executeQuery");
  ```

  在输出日志中产生如下类似的消息：

  ```
  myApp(thread[x,y,z]):executeQuery elapsed time =
    25ms
  ```

捕获 TDS 通信

Tabular Data Stream (TDS) 是用于处理客户端应用程序和 Adaptive Server 之间通信的 Sybase 专有协议。 jConnect 包含 PROTOCOL_CAPTURE 连接属性，该属性可以将多个原始 TDS 包捕获到一个文件中。

如果应用程序出现故障，而且无法在应用程序或服务器内部加以解决，则可使用 PROTOCOL_CAPTURE 捕获客户端和服务器之间的通信，并将其存放到一个文件中。该文件包含二进制数据，不能被直接解释。您可以将该文件发送到 Sybase 技术支持部门进行分析。

注释　也可使用 Ribo 实用程序来捕获、解释并显示客户端和服务器之间的协议流。有关如何获取和使用 Ribo 的详细信息，请访问位于 http://www.sybase.com/detail/1.6904.1009793.00.html 的 jConnect 实用程序 Web 页面。

PROTOCOL_CAPTURE 连接属性

使用 PROTOCOL_CAPTURE 连接属性指定一个文件，用于接收应用程序和 Adaptive Server 之间交换的 TDS 包。 PROTOCOL_CAPTURE 会立即生效，这样在建立连接过程中交换的 TDS 包就会被写入到指定的文件中。所有的 TDS 包将继续被写入到该文件中，直到执行 Capture.pause 或关闭该会话为止。

以下示例显示如何使用 PROTOCOL_CAPTURE 将 TDS 数据发送到文件 *tds_data* 中：

```
...
 props.put("PROTOCOL_CAPTURE", "tds_data")Connection
conn = DriverManager.getConnection(url, props);
```

其中，*url* 是连接的 URL，*props* 是用于指定连接属性的 Properties 对象。

Capture 类中的 *pause* 和 *resume* 方法

Capture 类包含在 com.sybase.jdbcx 包中。该类包含两个公共方法：

- public void pause
- public void resume

Capture.pause 停止将原始 TDS 包捕获到文件中； Capture.resume 重新开始捕获。

整个会话的 TDS 捕获文件可能会变得很大。如果要限制捕获文件的大小，并且知道要捕获的 TDS 数据在应用程序中的位置，则可执行如下操作。

❖ **若要限制捕获文件的大小**

1 在建立连接后，立即获取用于该连接的 Capture 对象，并使用 pause 方法停止捕获 TDS 数据：

```
Capture cap = ((SybConnection)conn).getCapture();
 cap.pause();
```

2 在要开始捕获 TDS 数据的位置前放置 cap.resume。

3 在要停止捕获 TDS 数据的位置后放置 cap.pause。

解决连接错误

本节介绍如何解决在试图建立连接或启用网关时出现的问题。

网关连接被拒绝

```
Gateway connection refused:
HTTP/1.0 502 Bad Gateway|Restart Connection
```

该错误消息表明，使用 *hostname* 或 *port#* 连接到 Adaptive Server 会出错。检查 *$SYBASE/interfaces* (UNIX) 或 *%SYBASE%\ini\sql.ini* (Windows) 中的 [query] 项。

如果在检验了 *hostname* 和 *port#* 之后问题仍然存在，请使用 “verbose” 系统属性启动 HTTP 服务器以进一步了解相关信息。

在 Windows 系统中，转至 DOS 提示符并输入：

```
httpd -Dverbose=1 > filename
```

在 UNIX 系统中，输入：

```
sh httpd.sh -Dverbose=1 > filename &
```

其中， *filename* 是调试消息输出文件。

您的 Web 服务器可能不支持 connect 方法。小程序仅能够连接到可下载这些小程序的主机。

HTTP 网关和 Web 服务器必须在同一主机上运行。在这种情况下，小程序可以通过 HTTP 网关 （HTTP 网关能够将请求路由到相关数据库）控制的端口连接到同一主机。

若要查看该过程是如何实现的，请查阅位于 jConnect 安装目录下的 *sample2* 子目录中的 *Isql.java* 和 *gateway.html* 源文件。搜索 “proxy”。

无法连接到 4.9.2 SQL Server

jConnect 使用 TDS 5.0（ Sybase 传输协议）。SQL Server 4.9.x 使用 TDS 4.6 （与 TDS 5.0 不兼容）。

而且， SQL Server 10.0.2 及更高版本均要求与 jConnect 一同使用。

管理 jConnect 应用程序所使用的内存

以下情形及其解决方案可能有助于解决 jConnect 应用程序使用越来越多的内存的问题。

- 在 jConnect 应用程序中，应当显式关闭最近一次使用的所有 Statement 对象以及子类 （例如， PreparedStatement、CallableStatement）以阻止语句累积在内存中。仅关闭 ResultSet 是不够的。

  例如，下面的语句会导致出现问题：

  ```
  ResultSet rs = _conn.prepareCall(_query).execute();
  ...
  rs.close();
  ```

  而应当使用如下语句：

  ```
  PreparedStatement ps = _conn.prepareCall(_query);
  ResultSet rs = ps.execute();
  ...
  ps.close();
  rs.close();
  ```

- jConnect 使用 TDS（Sybase 专有协议）与 Sybase 数据库服务器进行通信。在 jConnect 6.0 中， TDS 不支持可滚动游标。若要支持可滚动游标，则需 jConnect 每次调用 ResultSet.next 时，在客户端按需缓冲行数据。但到达结果集的末尾时，整个结果集将存储到客户端内存中。由于这可能会导致性能降低，因此， Sybase 建议仅当结果集相当小的时候，才使用 TYPE_SCROLL_INSENSITIVE 结果集。在此版本中， jConnect 会确定 ASE 连接是否支持本机可滚动游标功能，并使用该功能代替客户端高速缓存。因此，大多数应用程序都可以在访问无序行的过程中获得显著性能提高并可降低客户端内存要求。

解决存储过程错误

本节介绍如何解决在试图使用 jConnect 和存储过程时出现的问题。

RPC 返回比已注册参数更少的输出参数

```
SQLState: JZ0SG - An RPC did not return as many output
parameters as the application had registered for it.
```

如果通过调用 CallableStatement.registerOutParam 注册的参数多于在存储过程中声明为 “OUTPUT”的参数，则会出现此错误。确保已将所有相关参数声明为 “OUTPUT”参数。查看以下代码行：

```
create procedure yourproc (@p1 int OUTPUT, ...
```

注释　如果在使用 Adaptive Server Anywhere（以前称为 SQL Anywhere）的过程中接收到该错误，请升级到 Adaptive Server Anywhere 5.5.04 版或更高版本。

在返回输出参数时出现读取 / 状态错误

如果查询没有返回行数据，则应该使用 CallableStatement.executeUpdate 或 execute 方法而不应使用 executeQuery 方法。

根据 JDBC 标准的要求，如果 executeQuery 没有结果集， jConnect 会抛出一个 SQL 例外。

在非链式事务模式中执行存储过程

```
Sybase Error 7713 - Stored Procedure can only be
executed in unchained transaction mode.
```

在 JDBC 试图将连接置于 autocommit(true) 模式时会出现该错误。应用程序可使用 Connection.setAutoCommit(false) 或通过使用 “set chained on”语言命令将连接更改到链式模式。如果存储过程不是在兼容模式中创建的，则会出现该错误。

若要修复该问题，请使用：

sp_procxmode *procedure_name,"*anymode"

解决自定义套接字执行错误

当调用 sun.security.ssl.SSLSocketImpl.setEnabledCipherSuites 时如果试图设置 SSL 套接字，则可能会接收到如下类似的例外：

```
java.lang.IllegalArgumentException:
 SSL_SH_anon_EXPORT_WITH_RC4_40_MDS
```

检验 SSL 库是否使用系统库路径。

第 5 章

性能和调优

本章描述如何在使用 jConnect 时调优和改善性能。本章涉及以下主题：

主题	页码
改善 jConnect 性能	127
对动态 SQL 中的预准备语句的性能调优	129
游标性能	136

改善 jConnect 性能

以下多种方法用于优化使用 jConnect 的应用程序的性能：

- 使用 TextPointer.sendData 方法将文本和图像发送到 Adaptive Server 数据库。请参见第 64 页的“更新数据库中的图像数据”。
- 创建在会话过程中重复使用的动态 SQL 语句的预编译 PreparedStatement 对象。请参见第 129 页的 “对动态 SQL 中的预准备语句的性能调优”。
- 使用批处理更新通过减少网络通信量来改善性能；具体地说，就是所有查询被发送到一个组的服务器，而且返回到客户端的所有响应被发送到一个组中。请参见第 61 页的 “批处理更新支持”。
- 对于可能会移动图像数据、大的行集以及过长的文本数据的会话，使用 PACKETSIZE 连接属性设置最大可用包大小。
- 对于 TDS-tunneled 的 HTTP，设置最大 TDS 包大小并配置 Web 服务器以支持 HTTP1.1 Keep-Alive 功能。此外，将 *SkipDoneProc* 服务器小程序参数设置为 “true”。
- 使用协议游标（LANGUAGE_CURSOR 连接属性的缺省设置）。有关详细信息，请参见第 136 页的“LANGUAGE_CURSOR 连接属性”。

- 如果使用 TYPE_SCROLL_INSENSITIVE 结果集，应只在结果集很小时才使用它们。有关详细信息，请参见第 58 页的 “在 jConnect 中使用 TYPE_SCROLL_INSENSITIVE 结果集”。

在后续章节中将介绍改善性能的其它需要考虑的事项。

BigDecimal 范围重设

JDBC 1.0 规范要求 getBigDecimal 有一个范围因子。然后，当从服务器返回 BigDecimal 对象时，必须使用 getBigDecimal 已使用的原范围因子对其重新进行范围设置。

若要减少范围重设所需的时间，请使用 JDBC 2.0 getBigDecimal 方法（jConnect 在 SybResultSet 类中实现，且不需要 *scale* 值）：

```
public BigDecimal getBigDecimal(int columnIndex)
  throws SQLException
```

例如：

```
SybResultSet rs =
   (SybResultSet)stmt.executeQuery("SELECT
   numeric_column from T1");
 while (rs.next())
 {
   BigDecimal bd rs.getBigDecimal(
     "numeric_column");
      ...
 }
```

REPEAT_READ 连接属性

如果将 REPEAT_READ 连接属性设置为 “false”，可改善从数据库中检索结果集的性能。但是，当 REPEAT_READ 为 “false”时：

- 必须按照列索引顺序读取列值。如果要按名称而不是按列编号访问列将是很困难的。
- 不能多次读取行中的列值。

SunIoConverter 字符集转换

如果使用多字节字符集并需要改善驱动程序性能，可以使用 jConnect 示例提供的 SunIoConverter 类。该转换程序基于 Sun Microsystems, Inc. 公司 Java 软件部门提供的 sun.io 类。

SunIoConverter 类不是字符集转换程序功能的纯 Java 实现，而且因此未集成到标准 jConnect 产品中。不过，Sybase 已提供了该转换程序以供参考之用，并且可与 jConnect 驱动程序一起使用来改善字符集转换性能。

注释　根据 Sybase 的测试，SunIoConverter 类在所测试的所有虚拟机上都改善了性能。不过，Sun Microsystems, Inc. 公司 Java 软件部门保留在 JDK 未来的版本中删除或更改 sun.io 类的权利。因此，此 SunIoConverter 类可能会与 JDK 的更高版本不兼容。

若要使用 SunIoConverter 类，必须先安装 jConnect 示例应用程序。安装完毕后，设置 CHARSET_CONVERTER_CLASS 连接属性，使其指向 jConnect 安装目录的 *sample2* 子目录中的 SunIoConverter 类。有关安装 jConnect 及其组件（包括示例应用程序）的完整指导，请参见 *Sybase jConnect for JDBC 安装指南*。

若正在使用缺省字符集为 iso_1 的数据库或仅前 7 位 ASCII，则通过使用 TruncationConverter 可获得显著的性能优势。请参见第 36 页的“Connect 字符集转换程序”。

对动态 SQL 中的预准备语句的性能调优

在 Embedded SQL™ 中，动态语句是需要在运行期编译的 SQL 语句，而不是静态编译的 SQL 语句。通常，动态语句包含输入参数，但这不是必需的。在 SQL 中，prepare 命令用于预编译动态语句并将其保存，以使其在会话期间不必重新编译便可重复执行。

如果语句在一个会话中使用多次，预编译将比每次使用时将其发送到数据库并进行编译提供更好的性能。语句越复杂，性能优势就越显著。

如果可能仅使用几次语句，预编译可能会降低效率，因为在数据库中的预编译、保存以及随后的释放都会增加开销。

预编译要执行的动态 SQL 语句并将其保存到内存中会耗用时间和资源。如果在会话中不太可能多次使用同一个语句，执行数据库 prepare 的开销可能会大大超过其性能优势。另外需要考虑的是，只要数据库中预准备了动态 SQL 语句，它就非常类似于存储过程。在某些情况下，创建存储过程并使其驻留在服务器中可能会比在应用程序中定义预准备语句更可取。这将在第 130 页的 “选择预准备语句和存储过程”中讨论。

可以使用 jConnect 优化 Sybase 数据库中动态 SQL 语句的性能，方法如下：

- 在同一语句在会话中可能多次执行的情况下，创建包含预编译语句的 PreparedStatement 对象。
- 在同一语句在会话中很少使用的情况下，创建包含未编译 SQL 语句的 PreparedStatement 对象。

如以下各节所述，设置 DYNAMIC_PREPARE 连接属性并创建 PreparedStatement 对象的最佳方法可取决于应用程序是否需要跨 JDBC 驱动程序移植，或者所编写的应用程序是否允许到 JDBC 的特定 jConnect 扩展。

jConnect 4.1 以及更高版本提供了动态 SQL 语句的性能调优功能。

选择预准备语句和存储过程

如果创建包含预编译动态 SQL 语句的 PreparedStatement 对象，一旦该语句在数据库中编译，它实际上就变成了存储过程，驻留在内存中，并附加到与会话相关的数据结构中。在决定是否维护数据库中的存储过程或在应用程序中创建包含已编译 SQL 语句的 PreparedStatement 对象时，资源需求以及数据库和应用程序维护都是需要考虑的重要因素：

- 存储过程一旦被编译，就跨所有的连接在全局都可用。相反，PreparedStatement 对象中的动态 SQL 语句在每个使用它的会话中都需要进行编译和释放。
- 如果应用程序访问多个数据库，使用存储过程意味着相同的存储过程需要在所有的目标数据库上都可用。这样便产生了数据库维护问题。如果对动态 SQL 语句使用 PreparedStatement 对象，就可避免出现这一问题。
- 如果应用程序为调用存储过程创建了 CallableStatement 对象，则可在该存储过程中封装 SQL 代码和表引用。然后可修改基础数据库或 SQL 代码而无需更改应用程序。

可移植应用程序中的预准备语句

如果在来自不同供应商的数据库上运行应用程序，而且想要一些 PreparedStatement 对象包含预编译语句，而其它对象包含未编译语句，进行如下处理：

- 在访问 Sybase 数据库时，确保已将 DYNAMIC_PREPARE 连接属性设置为 “ true”。
- 若要返回包含预编译语句的 PreparedStatement 对象，请以标准方式使用 Connection.prepareStatement：

  ```
  PreparedStatement ps_precomp =
     Connection.prepareStatement(sql_string);
  ```

- 若要返回包含未编译语句的 PreparedStatement 对象，请使用 Connection.prepareCall。

  Connection.prepareCall 返回 CallableStatement 对象，但因为 CallableStatement 是 PreparedStatement 的一个子类，则可向上转换 CallableStatement 对象到 PreparedStatement 对象，如下所示：

  ```
  PreparedStatement ps_uncomp =
     Connection.prepareCall(sql_string);
  ```

  确保 PreparedStatement 对象 *ps_uncomp* 包含未编译语句，因为仅执行 Connection.prepareStatement 返回包含预编译语句的 PreparedStatement 对象。

具有 jConnect 扩展的预准备语句

如果不考虑跨驱动程序的可移植性，可编写使用 SybConnection.prepareStatement 的代码，以指定 PreparedStatement 对象是否包含预编译或未编译语句。在这种情况下，如何编码预准备语句可取决于应用程序中的大多数动态语句在会话中可能执行多次还是仅执行几次。

如果大多数动态语句不常执行

对于大多数动态 SQL 语句在会话中可能仅执行一两次的应用程序：

- 将连接属性 DYNAMIC_PREPARE 设置为 “ false”。
- 若要返回包含未编译语句的 PreparedStatement 对象，请以标准方式使用 Connection.prepareStatement：

```
PreparedStatement ps_uncomp =
   Connection.prepareStatement(sql_string);
```

- 若要返回包含预编译语句的 PreparedStatement 对象，请使用 SybConnection.prepareStatement 并将 *dynamic* 设置为 “true”，如下所示：

```
PreparedStatement ps_precomp =
   (SybConnection)conn.prepareStatement(sql_string, true);
```

如果大多数动态语句在会话中执行多次

如果应用程序中的大多数动态语句在会话中可能执行多次，进行如下处理：

- 将连接属性 DYNAMIC_PREPARE 设置为 “ true”。
- 若要返回包含预编译语句的 PreparedStatement 对象，请以标准方式使用 Connection.prepareStatement：

```
PreparedStatement ps_precomp =
   Connection.prepareStatement(sql_string);
```

- 若要返回包含未编译语句的 PreparedStatement 对象，则可使用 Connection.prepareCall （请参见可移植应用程序中的预准备语句的第三个符号项）或 SybConnection.prepareStatement 并将 *dynamic* 设置为 “false”：

```
PreparedStatement ps_uncomp =
   (SybConnection)conn.prepareStatement(sql_string,
false);

PreparedStatement ps_uncomp =
   Connection.prepareCall(sql_string);
```

Connection.prepareStatement

jConnect 执行 Connection.prepareStatement，因此可对其进行设置，以返回 PreparedStatement 对象中的预编译 SQL 语句或未编译 SQL 语句。如果设置 Connection.prepareStatement 返回 PreparedStatement 对象中的预编译 SQL 语句，它会将动态 SQL 语句发送到数据库中进行预编译，并且如同直接执行 prepare 命令时一样，被准确地保存下来。如果设置 Connection.prepareStatement 返回未编译 SQL 语句，它将返回 PreparedStatement 对象中的未编译 SQL 语句，而不将其发送到数据库中。

Connection.prepareStatement 返回的 SQL 语句的类型由连接属性 DYNAMIC_PREPARE 确定，而且在整个会话中适用。

对于特定 Sybase 应用程序，jConnect 6.05 提供了 jConnect SybConnection 类下的 prepareStatement 方法。 SybConnection.prepareStatement 允许指定是否对单个动态 SQL 语句进行预编译，而与 DYNAMIC_PREPARE 连接属性的会话级设置无关。

DYNAMIC_PREPARE 连接属性

DYNAMIC_PREPARE 是用于启用动态 SQL 预准备语句的布尔值连接属性：

- 如果将 DYNAMIC_PREPARE 设置为 “true”，则在会话期间每次调用 Connection.prepareStatement 都将试图返回 PreparedStatement 对象中的预编译语句。

  在这种情况下，PreparedStatement 执行时，它所包含的语句已在数据库中进行了预编译，拥有动态赋值的占位符，而且仅需执行该语句。

- 如果将连接的 DYNAMIC_PREPARE 设置为 “false”，则由 Connection.prepareStatement 返回的 PreparedStatement 对象不包含预编译语句。

  在这种情况下，每次执行 PreparedStatement 时，它所包含的动态 SQL 语句必须发送到数据库中进行编译和执行。

DYNAMIC_PREPARE 的缺省值为 “ false”。

在以下示例中， DYNAMIC_PREPARE 设置为 “true”以启用动态 SQL 语句的预编译。在该示例中， props 是指定连接属性的 Properties 对象。

```
...
 props.put("DYNAMIC_PREPARE", "true")
 Connection conn = DriverManager.getConnection(url,
props);
```

将 DYNAMIC_PREPARE 设置为 “true”时，应注意：

- 并非所有的动态语句均可在 prepare 命令下预编译。 SQL-92 标准对可用于 prepare 命令的语句做了一些限制，而且每个数据库供应商可能会有各自不同的约束。
- 如果数据库因为不能预编译和不能保存通过 Connection.prepareStatement 发送到该数据库的语句而产生错误，jConnect 会捕获该错误，并返回包含未编译动态 SQL 语句的 PreparedStatement 对象。每次执行 PreparedStatement 对象时，该语句都会重新发送到数据库进行编译和执行。
- 在会话结束或显式关闭预编译语句的 PreparedStatement 对象之前，预编译语句将一直驻留在数据库的内存中。 PreparedStatement 对象的碎片收集不能从数据库中删除预准备语句。

作为一般规则，应当在最后一次使用 PreparedStatement 对象之后显式关闭它，以避免预准备语句在会话期间累积在服务器的内存中并降低性能。

SybConnection.prepareStatement

如果应用程序允许对 JDBC 的特定 jConnect 扩展，则可使用 SybConnection.prepareStatement 扩展方法返回 PreparedStatement 对象中的动态 SQL 语句：

```
PreparedStatement SybConnection.prepareStatement
(String sql_stmt,
   boolean dynamic) throws SQLException
```

SybConnection.prepareStatement 会根据 *dynamic* 参数的设置，返回包含预编译或未编译 SQL 语句的 PreparedStatement 对象。如果 *dynamic* 为 “true”，则 SybConnection.prepareStatement 返回具有预编译 SQL 语句的 PreparedStatement 对象。如果 *dynamic* 为 “false”，则它会返回具有未编译 SQL 语句的 PreparedStatement 对象。

以下示例显示如何使用 SybConnection.prepareStatement 返回包含预编译语句的 PreparedStatement 对象：

```
PreparedStatement precomp_stmt =
   ((SybConnection) conn).prepareStatement( "SELECT *
FROM
   authors WHERE au_fname LIKE ?", true);
```

在该示例中，连接对象 *conn* 转换为 SybConnection 对象，以允许使用 SybConnection.prepareStatement。传递给 SybConnection.prepareStatement 的 SQL 字符串在数据库中预编译，即使连接属性 DYNAMIC_PREPARE 设置为 “false”。

如果数据库因为不能预编译通过 SybConnection.prepareStatement 发送到该数据库的语句而产生错误，则 jConnect 会抛出 SQLException，并且调用无法返回 PreparedStatement 对象。这与 Connection.prepareStatement 不同，后者会捕获 SQL 错误，并且如果产生上述错误，它会返回包含未编译语句的 PreparedStatement 对象。

ESCAPE_PROCESSING_DEFAULT 连接属性

缺省情况下，jConnect 会分析提交到数据库的所有 SQL 语句，以查找有效的 JDBC 函数转义。如果应用程序不在其 SQL 调用中使用 JDBC 函数转义，可将此连接属性设置为 “ false”以回避此分析过程。这样做可以稍微改善性能。

游标性能

当在 SybCursorResultSet 类中使用 Statement.setCursorName 方法或 setFetchSize( ) 方法时， jConnect 会在数据库中创建游标。使用其它方法可以使 jConnect 打开、读取和更新游标。

低于 jConnect 4.0 的版本只能通过使用显式游标命令将 SQL 语句发送到数据库进行分析和编译来创建和操纵游标。

jConnect 4.0 及更高版本则可通过将 SQL 语句发送到数据库或通过将游标命令编码为 TDS 通信协议内部的标识来创建和操纵游标。第一种类型的游标是 “语言游标”，第二种类型的游标是 “协议游标”。

协议游标较语言游标能够提供更好的性能。另外，并非所有的数据库都支持语言游标。例如， Adaptive Server Anywhere 数据库就不支持语言游标。

在 jConnect 中，缺省条件是所有游标都是协议游标。不过，LANGUAGE_CURSOR 连接属性提供了通过数据库中的语言命令创建和操纵游标的选项。

LANGUAGE_CURSOR 连接属性

LANGUAGE_CURSOR 在 jConnect 中是一个布尔值连接属性，用于确定将游标创建为协议游标还是语言游标：

- 如果 LANGUAGE_CURSOR 设置为 “ false”，则在会话期间创建的所有游标均为能够提供更好性能的协议游标。 jConnect 通过将游标命令作为 TDS 协议中的标识发送来创建和操纵游标。

  缺省情况下， LANGUAGE_CURSOR 设置为 “ false”。

- 如果 LANGUAGE_CURSOR 设置为 “ true”，则在会话期间创建的所有游标均为语言游标。 jConnect 通过将 SQL 语句发送到数据库进行分析和编译来创建和操纵游标。

  将 LANGUAGE_CURSOR 设置为 “ true”无任何已知优点，但如果将其设置为 “ false”时应用程序显示无法预料的结果时，在这种情况下可将其设置为 “ true”。

第 6 章

迁移 jConnect 应用程序

本章说明如何从 jConnect 4.x 和 5.x 向 jConnect 6.0 中迁移应用程序。本章涉及以下主题：

主题	页码
向 jConnect 6.x 迁移应用程序	137
更改 Sybase 扩展	138

向 jConnect 6.x 迁移应用程序

使用以下过程升级到 jConnect 6.x。

❖ **迁移到 jConnect 6.0**

1 如果代码使用 Sybase 扩展，或者如果在代码中显式导入任何 jConnect 类，则可根据需要更改软件包导入语句。

例如，将导入语句

```
import com.sybase.jdbc.*
```

和

```
import com.sybase.jdbc2.jdbc.*
```

更改为

```
import com.sybase.jdbcx.*
```

有关使用 Sybase 扩展 API 的信息，请参见第 138 页的 “更改 Sybase 扩展”。

2 将 JDBC_HOME 设置为 jConnect 驱动程序的顶层安装目录：

```
JDBC_HOME=jConnect-6_0
```

有关设置 JDBC_HOME 的详细信息，请参见 *jConnect for JDBC 安装指南* 的第 1 章中的 “设置环境变量”。

3 更改 CLASSPATH 环境变量以反映新的安装。对于 jConnect 6.0，类路径必须包括下面的内容：

```
JDBC_HOME/classes/jconn3.jar
```

4 更改用于装载驱动程序的源代码，并重新编译该应用程序以使用新的 jConnect 6.05 驱动程序：

```
Class.forName("com.sybase.jdbc3.jdbc.SybDriver");
```

5 检验新的 jConnect 6.05 驱动程序 （在 *JDBC_HOME/classes/jconn3.jar* 中）是否为 CLASSPATH 环境变量中指定的第一个 jConnect 驱动程序。

更改 Sybase 扩展

jConnect 版本 4.1 及更高版本包括软件包 com.sybase.jdbcx，该软件包含 JDBC 的所有 Sybase 扩展。在 jConnect 4.1 以前的版本中，可在 com.sybase.jdbc 和 com.sybase.utils 软件包中找到这些扩展。

com.sybase.jdbcx 软件包为不同版本的 jConnect 提供一致的接口。所有 Sybase 扩展都定义为 Java 接口，从而可以在不影响使用这些接口建立的应用程序的情况下更改底层实现。

当开发使用 Sybase 扩展的新应用程序时，可使用 com.sybase.jdbcx。该软件包中的接口允许以最小的更改将应用程序升级到 jConnect 4.0 以后的版本。

某些 Sybase 扩展已被更改为容纳新的 com.sybase.jdbcx 接口。

扩展更改示例

如果应用程序使用 SybMessageHandler，代码的区别有：

- **jConnect 4.0** 代码：

```
import com.sybase.jdbc.SybConnection;
import com.sybase.jdbc.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setMessageHandler(new ConnectionMsgHandler());
```

- **jConnect 6.0** 代码：

```
import com.sybase.jdbcx.SybConnection;
import com.sybase.jdbcx.SybMessageHandler;
.
.
Connection con = DriverManager.getConnection(url, props);
SybConnection sybCon = (SybConnection) con;
sybCon.setSybMessageHandler(new ConnectionMsgHandler());
```

有关如何使用 Sybase 扩展的更多示例，请参见与 jConnect 一起提供的示例。

方法名称

表 6-1 列出了方法在新接口中的新名称。

表 6-1：方法名称更改

类	原有名称	新名称
SybConnection	getCapture( )	createCapture( )
SybConnection	setMessageHandler( )	setSybMessageHandler( )
SybConnection	getMessageHandler( )	getSybMessageHandler( )
SybStatement	setMessageHandler( )	setSybMessageHandler( )
SybStatement	getMessageHandler( )	getSybMessageHandler( )

Debug 类

不再支持对 Debug 类的直接静态引用，但在 com.sybase.utils 软件包中存在此不受支持的引用。若要使用 jConnect 调试功能，请使用 SybDriver 类的 getDebug 方法来获得对 Debug 类的引用。例如：

```
import com.sybase.jdbcx.SybDriver;
import com.sybase.jdbcx.Debug;
.
.
.
SybDriver sybDriver =
    SybDriver)Class.forName
    ("com.sybase.jdbc3.jdbc.SybDriver") newInstance();
Debug sybDebug = sybDriver.getDebug();
sybDebug.debug(true, "ALL", System.out);
```

在 jConnect javadoc 文档中有 Sybase 扩展的完整列表，该文档位于 jConnect 安装目录的 *docs/* 目录中。

第 7 章

Web 服务器网关

本章介绍 Web 服务器网关以及如何利用 jConnect 来使用这些网关。本章涉及以下主题：

主题	页码
关于 Web 服务器网关	141
使用要求	147
使用 TDS 贯通服务器小程序	149

关于 Web 服务器网关

如果数据库服务器与 Web 服务器运行在不同的主机上，或者如果正在开发的 Internet 应用程序必须通过防火墙连接到安全的数据库服务器，则需要一个网关充当代理，以提供到数据库服务器的路径。

为使用安全套接字层 (SSL) 协议连接到服务器， jConnect 提供了一个 Java 服务器小程序，该小程序可安装在支持 javax.servlet 接口的任何 Web 服务器上。该服务器小程序启用 jConnect 支持加密并将 Web 服务器用作网关。

注释 jConnect 支持客户端系统上的 SSL。有关详细信息，请参见第 31 页的 “实现自定义套接字插件”。

使用 TDS 贯通

jConnect 使用 TDS 与数据库服务器通信。HTTP 贯通 TDS 可用于转发请求。从客户端到后端数据库的请求经过了整个网关，并在请求的正文中包含 TDS。请求的标头指示请求包中包含的 TDS 的长度。

TDS 是一种面向连接的协议，但 HTTP 不是。为支持安全性功能 （如为 Internet 应用程序加密）， jConnect 使用 TDS 贯通服务器小程序来维护各 HTTP 请求间的逻辑连接。服务器小程序在初始登录请求的过程中生成一个会话 ID，并且每个后续请求的标头中都包含此会话 ID。使用会话 ID 可以标识活动会话甚至恢复会话，只要服务器小程序拥有一个使用该特定会话 ID 的开放式连接。

TDS 贯通服务器小程序提供的逻辑连接启用 jConnect 以支持两个系统间的加密通信，例如，jConnect 客户端将 CONNECT_PROTOCOL 连接属性设置为“https”后可连接到运行 TDS 贯通服务器小程序的 Web 服务器。

配置 jConnect 和网关

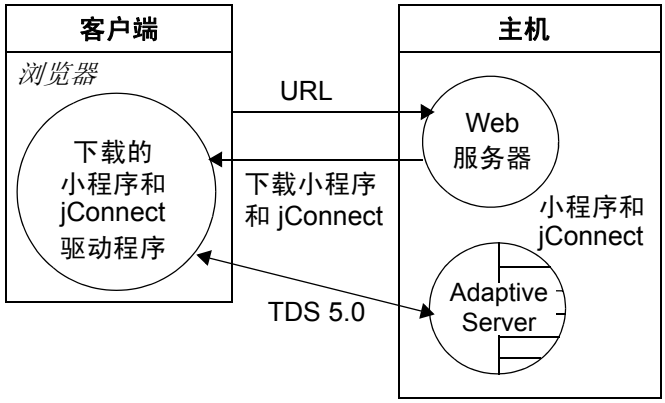
有几个用于设置 Web 服务器和 Adaptive Server 的选项。下面是关于四种常见配置的示例，显示了在何处安装 jConnect 驱动程序以及何时使用带有 TDS 贯通服务器小程序的网关。

Web 服务器和 Adaptive Server 在同一主机上

在两层配置中， Web 服务器和 Adaptive Server 安装在同一主机上：

- 在 Web 服务器主机上安装 jConnect。
- 不需要任何网关。

图 7-1：Web 服务器和 Adaptive Server 在同一主机上

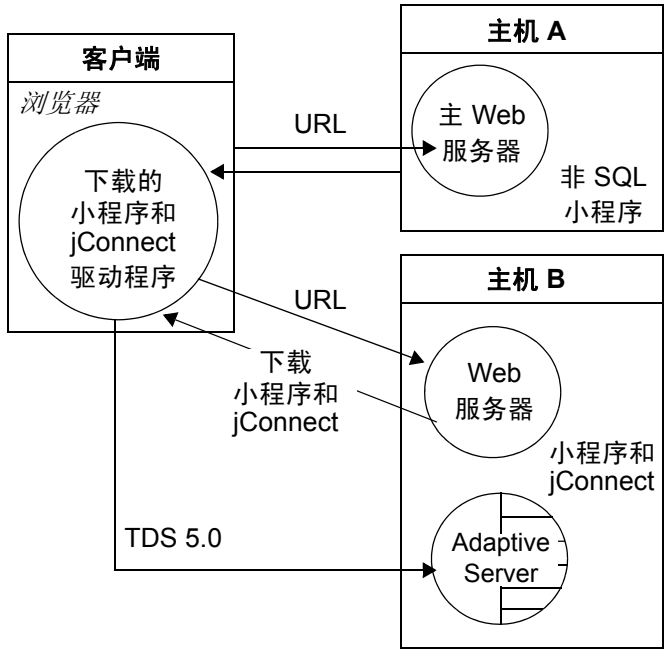


专用 JDBC Web 服务器和 Adaptive Server 在同一主机上

在此配置中，主 Web 服务器在一台单独的主机上。另一台主机由专用于 Adaptive Server 访问的 Web 服务器和 Adaptive Server 共享。来自主服务器的链接发送请求，要求 SQL 访问专用 Web 服务器。在另一台主机上安装：

- 在第二台 (Adaptive Server) 主机上安装 jConnect。
- 不需要任何网关。

图 7-2：专用 JDBC Web 服务器和 Adaptive Server 在同一主机上

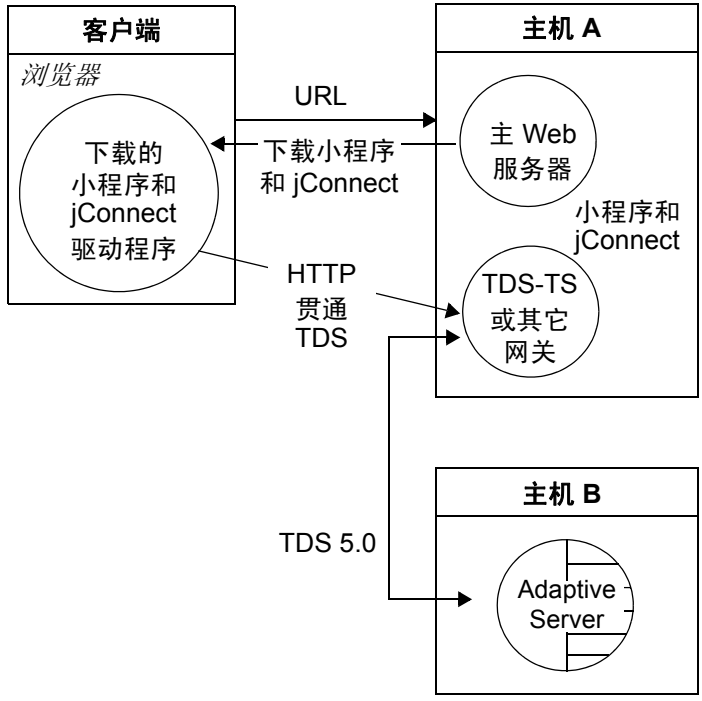


Web 服务器和 Adaptive Server 在不同的主机上

在三层配置中， Adaptive Server 和 Web 服务器在不同的主机上。jConnect 需要一个网关来充当 Adaptive Server 的代理。

- 在 Web 服务器主机上安装 jConnect。
- 安装 TDS 贯通服务器小程序或其它网关。

图 7-3：Web 服务器和 Adaptive Server 在不同的主机上

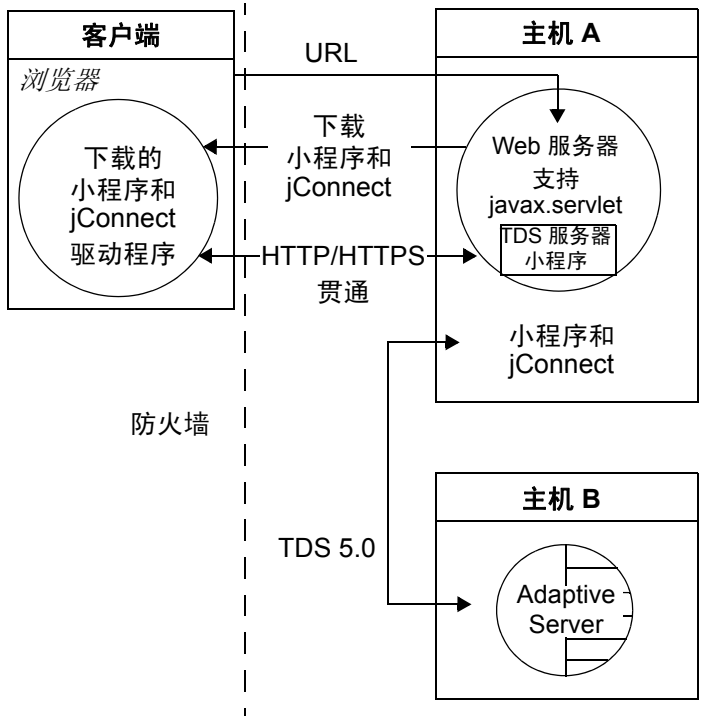


通过防火墙连接到服务器

若要连接到受防火墙保护的服务器，必须使用带有 TDS 贯通服务器小程序的 Web 服务器来支持在 Internet 上传输数据库请求响应。

- 在 Web 服务器主机上安装 jConnect。
- 要求一台支持 javax.servlet 接口的 Web 服务器。

图 7-4：通过防火墙连接到服务器



使用要求

以下几节介绍 Web 服务器网关的使用要求。

阅读 *index.html* 文件

使用 Web 浏览器查看 jConnect 安装目录中的 *index.html* 文件。*index.html* 提供了到 jConnect 文档和示例代码的链接。

注释　如果在安装有 jConnect 的同一台计算机上使用 Netscape，请确保浏览器无权访问 CLASSPATH 环境变量。请参见 *Sybase jConnect for JDBC 安装指南和发行公告* 的第 3 章中的 “使用 Netscape 时设置 CLASSPATH 的限制”。

❖ **若要查看 index.html 文件**

1　打开 Web 浏览器。

2　输入与安装相匹配的 URL。例如，如果浏览器和 Web 服务器运行在同一主机上，输入：

```
http://localhost:8000/index.html
```

如果浏览器和 Web 服务器运行在不同的主机上，输入：

```
http://host:port/index.html
```

其中 *host* 是在其上运行 Web 服务器的主机的名称， *port* 是侦听端口。

运行示例 Isql 小程序

在浏览器中装载 *index.html* 文件后：

❖ **若要运行示例小程序**

1 单击 “运行示例 JDBC 小程序”。

此操作将带您进入 “jConnect 示例程序” (jConnect Sample Programs) 页面。

2 向下移动 “示例程序” (Sample Programs) 页面以找到 “可执行的示例” (Executable Samples) 下的表。

3 在该表中定位 “Isql.java”并在行末单击 “运行” (Run)。

示例 Isql.java 小程序提示在示例数据库上执行简单查询并显示结果。小程序显示缺省的 Adaptive Server 主机名、端口号、用户名 (*guest*)、口令 (*sybase*)、数据库和查询。小程序使用缺省值连接到 Sybase 示例数据库。单击 “执行” (Go) 后将返回结果。

故障排除

在 UNIX 上，如果小程序未能按预期显示，则可以修改小程序的屏幕维度：

❖ **若要修改小程序的屏幕维度**

1 使用文本编辑器编辑下面的内容：

$JDBC_HOME/sample2/gateway.html

2 将第 7 行的高度参数更改为 650。可尝试使用不同的高度设置。

3 在浏览器上重装 Web 页。

使用 TDS 贯通服务器小程序

若要使用 TDS 贯通服务器小程序，需要一台支持 javax.servlet 接口的 Web 服务器（如 Sun Microsystems Java Web 服务器）。安装 Web 服务器时，把 jConnect TDS 贯通服务器小程序添加到活动服务器小程序列表中。还可以通过设置服务器小程序参数来定义连接超时和最大包大小。

使用 TDS 贯通服务器小程序时，从客户端到后端服务器的请求贯通网关，这样的请求包括 GET 或 POST 命令、 TDS 会话 ID （在初始请求后）、后端地址和请求状态。

TDS 在请求正文中。两个标头字段表示 TDS 流的长度和网关指派的会话 ID。

当客户端发送请求时， Content-Length 标头字段表示 TDS 内容的大小，请求命令为 POST。如果由于客户端正在检索来自服务器的响应数据的下一部分内容或正在关闭连接，从而造成请求中没有任何 TDS 数据，此时请求命令为 GET。

下例说明如何使用 TDS 贯通 HTTPS 协议在客户端和 HTTPS 网关之间传递信息；该示例显示了一个到名为 “DBSERVER”的后端服务器的端口号 “1234”的连接。

表 7-1：客户端到网关的登录请求。无会话 ID。

查询	POST/tds?ServerHost=dbserver&ServerPort=1234& Operation=more HTTP/1.0
标头	内容长度：605
内容 (TDS)	登录请求

表 7-2：网关到客户端。标头中包含由 TDS 服务器小程序分配的会话 ID。

查询	200 SUCCESS HTTP/1.0
标头	内容长度：210 TDS 会话：TDS00245817298274292
内容 (TDS)	登录确认 EED

表 7-3：客户端到网关。所有后续请求的标头都包含会话 ID。

查询	POST/tds?TDS- Session=TDS00245817298274292&Operation=more HTTP/1.0
标头	内容长度：32
内容 (TDS)	查询 “SELECT * from authors”

表 7-4：网关到客户端。所有后续响应的标头都包含会话 ID。

查询	200 SUCCESS HTTP/1.0
标头	内容长度：2048 TDS 会话：TDS00245817298274292
内容 (TDS)	行格式与某些来自查询响应的行

检查要求

若要对 TDS 贯通的 HTTP 使用 jConnect 服务器小程序，需要：

- 一台支持 javax.servlet 接口的 Web 服务器。若要安装该服务器，请遵循它提供的操作说明。
- 一个支持 JDK 1.1 的 Web 浏览器，比如 Netscape 4.0、 Internet Explorer 4.0 或 HotJava。

安装服务器小程序

jConnect 安装包括 *classes* 目录下的 *gateway2* 子目录。该子目录包含 TDS 贯通服务器小程序所需的文件。

将 jConnect gateway 包复制到 Web 服务器的 *servlets* 目录下的 *gateway2* 子目录中。复制好服务器小程序后，按照 Web 服务器操作说明激活服务器小程序。

设置服务器小程序参数

将服务器小程序添加到 Web 服务器时，可输入可选的参数来自定义性能：

- *SkipDoneProc [true|false]* — Sybase 数据库通常会在查询执行期间执行中间处理步骤时返回行计数信息。通常，客户端应用程序会忽略此数据。如果将 *SkipDoneProc* 设置为 “true”，服务器小程序会随即从响应中删除此额外信息，这将减少客户端上的网络使用和处理要求。这在使用 HTTPS/SSL 时尤其有效，因为不需要的数据在被忽略前不会被加密 / 解密。
- *TdsResponseSize* — 为贯通的 HTTPS 设置最大 TDS 包大小。如果只有几个用户有大量数据， *TdsResponseSize* 大点会更加有效。如果有许多执行小事务的用户，请使用较小的 *TdsResponseSize*。
- *TdsSessionIdleTimeout* — 定义在服务器连接自动关闭前该连接能够维持空闲状态的时间长度（以毫秒为单位）。*TdsSessionIdleTimeout* 缺省值为 600,000 （10 分钟）。

如果交互式客户端程序可能长时间处于空闲状态而您又不希望中断该连接，则增加 *TdsSessionIdleTimeout*。

还可从 jConnect 客户端使用 SESSION_TIMEOUT 连接属性设置连接超时值。这在存在可能长时间处于空闲状态的特定应用程序时很有用。在这种情况下，可通过 SESSION_TIMEOUT 连接属性为连接而不是为服务器小程序设置更长的超时值。

- *Debug* — 打开调试程序。请参见第 117 页的 “使用 jConnect 进行调试”。

输入服务器参数 （以逗号分隔的字符串）。例如：

```
TdsResponseSize=[size],TdsSessionIdleTimeout=
  [timeout],Debug=true
```

有关输入服务器小程序参数的完整指导，请参见 Web 服务器文档。

调用服务器小程序

jConnect 根据 *proxy* 连接属性的路径扩展确定何时使用安装有 TDS 贯通服务器小程序的网关。 jConnect 识别 *proxy* 的服务器路径扩展并调用指定网关上的服务器小程序。

用下面的格式定义连接 URL：

```
http://host:port/TDS-servlet-path
```

jConnect 通过调用 Web 服务器上的 TDS 贯通服务器小程序来使 TDS 贯通 HTTP。服务器小程序的路径必须是服务器别名列表中为 Web 服务器定义的路径。

跟踪活动的 TDS 会话

可查看活动 TDS 会话的相关信息，其中包括每个会话的服务器连接。使用 Web 浏览器打开管理 URL：

```
http://host:port/TDS-servlet-path?Operation=list
```

例如，如果服务器为 “myserver”， TDS 服务器小程序路径为 */tds*，则输入：

```
http://myserver:8080/tds?Operation=list
```

这将显示活动 TDS 会话的列表。单击会话可查看更多信息 （包括服务器连接）。

终止 TDS 会话

可使用上述 URL 终止任何活动的 TDS 会话。在第一页上单击会话列表中的某个活动会话，然后单击 “终止会话” (Terminate This Session)。

恢复 TDS 会话

必要时，可通过设置 SESSION_ID 连接属性来恢复现有的开放式连接。当您指定了一个 SESSION_ID， jConnect 将跳过协议的登录阶段并使用指定的会话 ID 恢复与网关的连接。如果指定的会话 ID 不在服务器小程序中，在首次尝试使用连接时， jConnect 会抛出一个 SQL 例外。

在 Solaris 上使用 TDS 和 Netscape Enterprise Server 3.5.1

Netscape Enterprise Server 3.5.1 不支持 javax.servlet.ServletConfig.getInitParameters 或 javax.servlet.ServletConfig.getInitParameterNames 方法。若要提供必要的参数值，需要用 *TDSTunnelServlet.java* 中的硬编码参数值替换对 getInitParameter( ) 和 getInitParameterNames 的调用。

若要在 *TDSTunnelServlet.java* 中输入所需的参数值并在 Solaris 上通过 Netscape Enterprise Server 3.5.1 使用 TDS 贯通，步骤如下：

1 *TDSTunnelServlet.java* 中的硬编码参数值。

2 从 *TDSTunnelServlet.java* 中的类声明创建 *.class* 文件。该操作应生成如下文件：

- *TDSTunnelServet.class*
- *TdsSession.class*
- *TdsSessionManager.class*

3 在 Netscape Enterprise Server 3.5.1 (*NSE_3.5.1*) 安装目录下，为 *.class* 文件创建一个目录，如下所示：

```
mkdir NSE_3.5.1_install_dir/plugins/java/servlets/gateway
```

4 将从 *TDSTunnelServlet.java* 派生的 *.class* 文件复制到刚创建的目录中。

5 将 *$JDBC_HOME/classes/com/sybase* 下的类复制到 *NSE_3.5.1_install_dir/docs/com/sybase*。

执行此操作的一个简单的方法是将 *$JDBC_HOME/classes* 下的所有内容递归复制到 *NSE_3.5.1_install_dir/docs*，如下所示：

```
cp -r $JDBC_HOME/classes NSE_3.5.1_install_dir/docs
```

此操作复制了许多不在 *$JDBC_HOME/classes/com/sybase* 下的文件和目录。虽然额外的文件和目录是无害的，但会占用磁盘空间。您可以删除它们以回收磁盘空间。

6 设置 TDS 贯通服务器小程序的 *proxy* URL。

例如，在 *$JDBC_HOME/sample2/gateway.html* 中，编辑 *proxy* 参数，如下所示：

```
<param name=proxy value="http://hostname/servlet/
   gateway_name.TDSTunnel_Servlet_name">
```

附 录 A

SQL 例外与警告消息

下表列出了在使用 jConnect 时会遇到的 SQL 例外与警告消息。

SQL 状态	消息 / 说明 / 操作
010AF	严重警告：断言失败，请使用 devclasses 确定此严重错误的起因。 消息 = _____。 **说明：**jConnect 驱动程序中的内部断言失败。 **操作：**使用 devclass 调试类确定显示此消息的原因并向 Sybase 技术支持部门报告此问题。
010DF	在登录时设置数据库失败。 错误消息：_____。 **说明：**jConnect 无法连接到在连接 URL 中指定的数据库。 **操作：**确保 URL 中的数据库名称正确无误。此外，如果连接到 Adaptive Server Anywhere，请使用 SERVICENAME 连接属性指定数据库。
010DP	忽略重复的连接属性 _____。 **说明：**某个连接属性被定义了两次。可能在驱动程序连接属性列表中用不同的大小写形式对该连接属性定义了两次，例如 “password” 和 “PASSWORD”。连接属性名称不区分大小写；因此 jConnect 不会区分大小写形式不同的同名属性。 也可以同时在连接属性列表和 URL 中定义中连接属性。在这种情况下，连接属性列表中的属性值优先。 **操作：**确保应用程序只定义一次连接属性。不过，或许您想利用属性列表中定义的连接属性优先于 URL 中定义的属性这一特性。这时，可以放心地忽略此警告。
010HA	服务器拒绝了使用高可用性功能的请求。 请重新配置数据库，或不要请求高可用性会话。 **说明：**服务器拒绝了 jConnect 试图建立高可用性连接的操作。 **操作：**重新配置服务器使其支持高可用性故障切换，或者不要将 REQUEST_HA_SESSION 设置为 “true”。
010HD	这种数据库服务器不支持 Sybase 的高可用性故障切换。 **说明：**jConnect 试图连接到的数据库不支持高可用性故障切换。 **操作：**应当只连接到支持高可用性故障切换的数据库服务器。
010HN	客户端未指定 SERVICE_PRINCIPAL_NAME 连接属性。 因此， jConnect 使用主机名 _____ 作为服务主体名称 **操作：**确保通过连接属性显式指定服务主体名称。

SQL 状态	消息 / 说明 / 操作
010HT	`Hostname` 属性被截断，最大长度为 `30`。 **说明：**您为 HOSTNAME 连接属性提供的字符串超过 30 个字符，或者运行 jConnect 应用程序的主机拥有长度超过 30 字节的名称。 **操作：**无需任何操作，这只是提醒您 jConnect 将把名称截断到 30 字节。然而，如果希望避免此警告，应将 HOSTNAME 设置为长度小于等于 30 个字节。
010KF	服务器拒绝了 `Kerberos` 登录。 这很可能是因为发生了通用安全服务 (`GSS`) 例外。 请检查 `Kerberos` 环境和配置。 **操作：**检查 Kerberos 环境，确保已正确鉴定到 KDC。有关详细信息，请参见第 3 章“安全性”。
010MX	找不到有关此数据库的元数据访问程序信息。 请安装 `jConnect` 文档中所述的必需表。 试图检索元数据信息时出错：`_____` **说明：**服务器可能不具有返回元数据信息所需的存储过程。 **操作：**确保服务器上安装有用于提供元数据的存储过程。请参见 *jConnect for JDBC 安装指南* 第 3 章中的 “安装存储过程”。
010P4	收到并忽略了一个输出参数。 **说明：**执行的查询返回一个输出参数，但应用程序的结果处理代码未读取该参数，因此将其忽略。 **操作：**如果应用程序需要输出参数数据，则须重写该应用程序以便能获取数据。这可能需要使用 CallableStatement 来执行查询，并添加对 registerOutputParameter 和 getXXX 的调用。也可以将 DISABLE_UNPROCESSED_PARAM_WARNINGS 连接属性设置为“true”，从而禁止 jConnect 返回此警告，这样做或许还可提高性能。
010PF	无法装载在 `PRELOAD_JARS` 连接属性中指定的一个或多个 `jar`。 **说明：**如果在使用 DynamicClassLoader 时将 PRELOAD_JARS 连接属性设置为以逗号分隔的 *.jar* 文件名的列表，则会出现此消息。当 DynamicClassLoader 打开与要从其中装载类的服务器的连接时，它会尝试 “预装载”此连接属性中提到的所有 *.jar* 文件。如果服务器上不存在指定的一个或多个 *.jar* 文件名，则会出现上述错误消息。 **操作：**确认在应用程序的 PRELOAD_JARS 连接属性中提到的所有 *.jar* 文件都存在于服务器上，而且都能访问。
010PO	属性 `LITERAL_PARAM` 已设置为 “`false`”，因为 `DYNAMIC_PREPARE` 的设置为 “`true`”。 **说明：**如果想使用预编译过的动态语句，必须允许将参数发送给这些语句 （如果语句使用参数的话）。将 LITERAL_PARAMS 设置为 “true”会强制以送往服务器的 SQL 中的文本值的形式发送所有参数。所以不能将这两个属性都设置为 “true”。 **操作：** 为了避免此警告，当您想使用动态 SQL 时不要将 LITERAL_PARAMS 设置为“true”。有关详细信息，请参见第 129 页的“对动态 SQL 中的预准备语句的性能调优”。
010RC	不支持请求的 `ResultSet` 类型和并发。 它们已被转换。 **说明：**您请求的 ResultSet 类型和并发组合不受支持。已转换所请求的值。有关在 jConnect 中可使用何种 ResultSet 类型和并发的详细信息，请参见第 50 页的 “对结果集使用游标”。 **操作：**请求受支持的 ResultSet 类型和并发组合。

SQL 状态	消息 / 说明 / 操作
010SJ	找不到有关此数据库的元数据访问程序信息。 请安装 jConnect 文档中所述的必需表。 **说明：**服务器上没有配置元数据信息。 **操作：**如果应用程序需要元数据，请安装 jConnect 中附带的用于返回元数据的存储过程（参见 *jConnect for JDBC 安装指南* 的第 3 章中的 “安装存储过程”）。如果不需要元数据，请将 USE_METADATA 属性设置为 “false”。
010SK	数据库无法设置连接选项 _____。 **说明：**连接的数据库不支持应用程序尝试的操作。 **操作：**可能需要升级数据库，或者确保安装了最新版本的元数据信息。
010SL	发现此数据库的元数据访问程序信息已过时。 请数据库管理员装载最新的脚本。 **说明：**服务器上的元数据信息已过时，需要更新。 **操作：**安装 jConnect 中附带的用于返回元数据的存储过程（参见 *jConnect for JDBC 安装指南* 的第 3 章中的 “安装存储过程”）。
010SM	此数据库不支持最初提议的功能集，正在重试。 **说明：**Adaptive Server Enterprise 11.9.2 版及更低版本有一个错误，该错误使它们在服务器没有客户端请求的功能时拒绝客户端登录。此警告说明 jConnect 检测到这种情况，且正以服务器可接受的最多功能数重试该连接。当 jConnect 遇到这项错误时，它会两次连接服务器。 **操作：**客户端可以放心地忽略此警告；但若想消除此警告并确保 jConnect 只进行一次连接尝试，客户端可以将 ELIMINATE_010SM 连接属性设置为 “true”。注意：在连接到 Adaptive Server 12.0 及更高版本时，不应将此属性设置为 “true”。
010SN	写入文件权限被拒绝。 文件：_____. 错误消息：_____ **说明：**因 VM 中的安全冲突，对 PROTOCOL_CAPTURE 连接属性中指定的文件的写入权限被拒绝。当一个小程序尝试写入指定文件时，会出现此消息。 **操作：**如果要通过小程序写入文件，必须确保该小程序可以访问目标文件系统。
010SP	无法打开文件进行写入。 文件：_____。 错误消息：_____ **操作：**确保文件名正确无误且文件可写。
010SQ	连接或登录被拒绝，正在通过主机 / 端口地址重试连接。 **说明：**CONNECTION_FAILOVER 连接属性被设置为 “true”， jConnect 无法连接到要连接的服务器列表中的某个数据库服务器。因此， jConnect 现在尝试连接到列表中的下一个服务器。 **操作：**只要 jConnect 能连接到另一数据库服务器，就无需任何操作。然而，您应当确定 jConnect 为何无法连接到导致警告的特定服务器。
010TP	服务器无法转换该连接的初始字符集 _____。 将使用服务器建议的字符集 _____，并由 jConnect 执行转换。 **说明：**服务器无法使用最初由 jConnect 请求的字符集，已经用不同的字符集响应。jConnect 接受这一更改并执行必要的字符集转换。 本消息只用来提供信息，无其它影响。 **操作：**为了避免此消息，可将 CHARSET 连接属性设置为服务器支持的字符集。

SQL 状态	消息 / 说明 / 操作
010TQ	`jConnect` 无法确定服务器的缺省字符集。 这可能是元数据问题引起的。 请安装 `jConnect` 文档中所述的必需表。 该连接缺省采用 `ascii_7` 字符集，该字符集只能处理 `0x00` 到 `0x7F` 之间的字符。 **说明：**jConnect 无法确定服务器的缺省字符集。当发生这种情况时，只有前 127 个 ASCII 码字符能够确保得以正确转换。因此，这时 jConnect 恢复到 7 位 ASCII 码。本消息只用来提供信息，无其它影响。 **操作：**安装 jConnect 中附带的用于返回元数据的存储过程（参见 *jConnect for JDBC 安装指南* 的第 3 章中的 “安装存储过程”）。
010UF	尝试执行 `use database` 命令失败。 错误消息：`_____` **说明：**jConnect 无法连接到在连接 URL 中指定的数据库。两种可能的原因是： • URL 中输入的名称有误。 • USE_METADATA 设置为 “true”（缺省设置），但未安装用于返回元数据的存储过程。结果，jConnect 尝试对 URL 中的数据库执行 use *database* 命令，但命令失败。这可能是由于您尝试访问 Adaptive Anywhere 数据库。SQL Anywhere 数据库不支持 use *database* 命令。 **操作：**确保 URL 中的数据库名正确无误。确保服务器上安装了用于返回元数据的存储过程 （参见 *jConnect for JDBC 安装指南和发行公告* 的第 3 章中的 “安装存储过程”）。若要访问 SQL Anywhere 数据库，则不要在 URL 中指定数据库名称，或者将 USE_METADATA 设置为 “false”。
010UP	忽略无法识别的重复连接属性 `_____`。 **说明：**您试图在 URL 中设置一个 jConnect 目前无法识别的连接属性。jConnect 将忽略该无法识别的属性。 **操作：**检查应用程序中的 URL 定义，确保其只引用有效的 jConnect 驱动程序连接属性。
0100V	正在使用的 `TDS` 协议版本过旧。 版本：`_____` **说明：**服务器不支持所需 TDS 协议版本。 jConnect 要求 5.0 版或更高版本。 **操作：**使用支持所需的 TDS 版本的服务器。有关详细信息，请参见 jConnect 安装指南中的系统要求部分。
JZ001	`User name` 属性 “`_____`” 过长。 最大长度为 `30`。 **操作：**不要超出 30 字节的最大长度限制。
JZ002	`Password` 属性 “`_____`” 过长。 最大长度为 `30`。 **操作：**不要超出 30 字节的最大长度限制。

SQL 状态	消息 / 说明 / 操作
JZ003	错误的 URL 格式。 URL(U)：_____ **操作：**检查 URL 格式。请参见第 24 页的 “URL 连接属性参数”。 如果使用的是 PROXY 连接属性并且该属性的格式有误，则在尝试连接时会产生 JZ003 例外。 该级联代理的 PROXY 格式为： *ip_address:port_number* TDS 贯通服务器小程序的 PROXY 格式为： http[s]:///*host*:*port*/*tunneling_servlet_alias*
JZ004	DriverManager.getConnection(..., Properties) 中缺少 User name 属性 **操作：**提供必需的用户属性。
JZ006	捕获到 IOException：_____ **说明：**从低层检测到意外的 I/O 错误。当捕获这种 I/O 例外时，它们将通过 ERR_IO_EXCEPTION JZ006 sqlstate 作为 SQL 例外被再次抛出。这些错误通常是由网络通信问题引起的。如果 I/O 例外导致数据库连接被关闭，则 jConnect 会将 JZ0C1 例外链接到 JZ006。客户端应用程序可以查找链中的 JZ0C1 例外，以查看该连接是否仍然可用。 **操作：**检查原始 I/O 例外消息的文本，并从该处继续。
JZ008	无效的列索引值 _____。 **说明：**请求的列索引值小于 1 或大于最大可用值。 **操作：**检查对 getXXX 方法的调用和原始查询的文本，或确保调用 rs.next。
JZ009	转换时发生错误。 错误消息：_____ **说明：**可能的部分原因有： • 尝试在两个不兼容的数据类型间进行转换，例如将 date 转换为 int。 • 试图将包含非数值字符的字符串转换为数值类型。 • 存在格式错误，例如 time/date 字符串格式有误。 **操作：**确保 JDBC 规范支持尝试进行的类型转换。确保字符串格式正确无误。如果字符串包含非数值字符，不要试图将其转换为数值类型。
JZ00B	数值溢出。 **说明：**试图将 BigInteger 作为 TDS 数值发送，而该值过大；或者试图将 Java long 作为 int 发送，而该值过大。 **操作：**不能在 Sybase 中存储这些值。对于 long，请考虑使用 Sybase 数值类型。尚无法解决 Bignum 的问题。
JZ00C	指定的精度和标度不匹配数值 _____。 **说明：**在使用 setBigDecimal 方法时， BigDecimal 值的精度或标度超出指定范围。 **操作：**确保指定的精度和标度与 BigDecimal 值相匹配。

SQL 状态	消息 / 说明 / 操作
JZ00E	视图在已调用 `setCursorName()` 的语句中调用 `execute()` 或 `executeUpdate()`。 **操作：**不要尝试在已设置游标名的语句中调用 execute 或 executeUpdate。请单独使用一条语句删除或更新游标。有关详细信息，请参见第 50 页的 “对结果集使用游标”
JZ00F	已通过 `setCursorName()` 设置了游标名。 **操作：**不要为同一语句设置两次游标名。关闭当前游标语句的结果集。
JZ00G	未为此行更新设置任何列值。 **说明：**您试图更新一行，但未在该行中更改任何列值。 **操作：**要更改行中的列值，请在调用 updateRow 之前调用 updateXX 方法。
JZ00H	结果集不可更新。 使用 `Statement.setResultSetConcurrencyType()`。 **操作：**要将结果集从只读改为可更新，请使用 Statement.setResultSetConcurrencyType 方法，或将 for update 子句添加到 SQL select 语句中。
JZ00I	无效精度。 指定的精度必须 `>=0`。 **说明：**标度值必须大于零。 **操作：**确保标度值非负。
JZ00L	登录失败。 检查链接到此例外的 `SQLWarnings` 来查找原因。 **操作：**查看消息文本；根据给出的登录失败原因继续操作。
JZ00M	登录超时。 检查数据库服务器是否在指定的主机和端口号上运行。 同时检查可能导致数据库服务器挂起的其它情况 （例如 `tempdb` 已满）。 **操作：**按照错误消息中提供的建议进行操作。
JZ010	无法将 `Object` 值非序列化。 错误文本：`_____` **操作：**确保数据库中的 Java 对象实现 Serializable 接口，并在本地 CLASSPATH 变量中。
JZ011	在分析数值连接属性 `_____` 时发生数字格式例外。 **说明：**为数值连接属性指定了非整数值。 **操作：**为连接属性指定整数值。
JZ012	内部错误。 请将错误报告给 `Sybase` 技术支持。 连接属性 `_____` 的访问类型有误。 **操作：**与 Sybase 技术支持部门联系。
JZ013	获取 `JNDI` 条目时出错：`_____` **操作：**更正 JNDI URL，或在目录服务中创建一个新条目。
JZ014	您不能运行 `setTransactionIsolation(Connection.TRANSACTION_NONE)`。 这一级别不能进行设置，只能由服务器返回。 **操作：**检查调用 Connection.setTransactionIsolation 的应用程序代码，并验证传递给该方法的值。
JZ015	为 `GSSMANAGER_CLASS` 连接属性设置了非法值。 属性值必须是扩展 `org.ietf.jgss.GSSManager` 的 `String` 或 `Object`。 **操作：**检查为 GSSMANAGER_CLASS 属性设置的值。
JZ0BD	方法参数中使用了越界值或无效值。 **操作：**验证方法中的参数值是否正确无误。

SQL 状态	消息 / 说明 / 操作
JZ0BI	**消息：** setFetchSize：读取大小应该在以下范围内设置 — 0 <= 行数 <= （结果集中的最大行数）。 **说明：**客户端应用程序在调用 setFetchSize 时使用了无效的行数。 **操作：**验证调用 setFetchSize 时使用的参数值是否位于上述范围内。
JZ0BP	批处理更新语句中不允许使用输出参数。 **操作：**检查应用程序代码，并确保未尝试在批处理中声明输出参数。
JZ0BR	游标所在的行不支持 _____ 方法。 **说明：**尝试调用的 ResultSet 方法对当前行位置无效 （例如，调用 insertRow 时游标不在插入行上）。 **操作：**不要调用对当前行位置无效的 ResultSet 方法。
JZ0BS	不支持批处理语句。 **操作：**在数据库中通过最新版本安装或更新 jConnect 元数据存储过程。
JZ0BT	_____ 类型的 ResultSet 不支持 _____ 方法。 **说明：**您尝试调用的 ResultSet 方法对此类型的 ResultSet 无效。 **操作：**不要尝试调用对 ResultSet 类型无效的 ResultSet 方法。
JZ0C0	连接已关闭。 **说明：**应用程序已对此连接对象调用 Connection.close ，无法再使用此连接。 **操作：**修正代码，在连接关闭时清空连接对象引用。
JZ0C1	出现 IOException，此错误将使连接关闭。 **说明：**出现了不可恢复的 IOException 例外，连接已被关闭。该连接不能再用于其它任何数据库操作。如果出现此例外，则它总会与 JZ006 例外一起出现在一个例外链中 （前面解释过）。 **操作：**确定导致连接中断的 IOException 的起因。
JZ0CL	在使用 PRELOAD_JARS 属性时必须定义 CLASS_LOADER 属性。 **操作：**在将 PRELOAD_JARS 设置为非空值时，请务必指定一个 CLASS_LOADER 。
JZ0CU	成功调用 getMoreResults 或执行方法后，只能调用一次 getUpdateCount。 **说明：**按照 JDBC API 规定，只应对每个结果调用一次 getUpdateCount。 **操作：**确保您的代码没有对一个结果多次调用 getUpdateCount。
JZ0D4	Sybase JDBC URL _____ 中无法识别的协议。 **说明：**您使用非 TDS 协议指定了连接 URL，但目前 jConnect 只支持 TDS 协议。 **操作：**检查 URL 定义。如果 URL 指定 TDS 作为子协议，请确保该条目使用以下格式和大小写形式： jdbc:sybase:Tds:*host*:*port* 如果 URL 指定 JNDI 作为子协议，请确保它的开头为： jdbc:sybase:jndi:

SQL 状态	消息 / 说明 / 操作
JZ0D5	装载协议 _____ 时出错。 **操作：**检查 CLASSPATH 系统变量的设置。
JZ0D6	setVersion 中指定了无法识别的版本号 _____。 选择 SybDriver.VERSION_* 值之一，并确保所用 jConnect 版本等于或高于指定的版本。 **操作：**参见消息文本。
JZ0D7	装载 url 提供程序 _____ 时出错。 错误消息：_____ **操作：**检查 JNDI URL，确保其正确无误。
JZ0D8	初始化 URL 提供程序时出错：_____ **操作：**检查 JNDI URL，确保其正确无误。
JZ0DP	此语句中不含元数据，因为它并非是动态准备的。 将 DYNAMIC_PREPARE 连接属性设置为 true 以确保动态语句的使用。 **操作：**参见错误消息。
JZ0EM	数据结束。 **操作：**请向 Sybase 技术支持部门报告此错误。
JZ0F1	请求了 Sybase 高可用性故障切换连接，但协同服务器地址丢失。 **说明：**将 REQUEST_HA_SESSION 连接属性设置为 “true”时，必须同时指定故障切换服务器。 **操作：**可以用 SECONDARY_SERVER_HOSTPORT 连接属性指定辅助服务器，或者用 JNDI 设置辅助服务器 （请参见第 42 页的 “实现高可用性故障切换支持”）。
JZ0F2	已发生 Sybase 高可用性故障切换。 当前事务已中止，但连接仍可用。 请重试事务。 **说明：**之前连接的后端数据库服务器已关闭，但您已经切换到一个辅助服务器。数据库连接仍可用。 **操作：**客户端代码应捕获这一例外，再从上次提交点重新启动事务。如果正确处理了此例外，就可以对同一连接对象继续执行 JDBC 调用。
JZ0GC	将 _____ 作为 GSSManager 转换时出错。 请检查设置给 GSSMANAGER_CLASS 连接属性的值。 该值必须为一个 String，它要指定 GSSManager 实现的完全限定的类名称。 或者必须是一个扩展 org.ietf.jgss.GSSManager 的 Object。 **操作：**参见消息文本。
JZ0GN	将类 ____ 作为 GSSManager 例示时出错。 例外是 _____。 请检查 CLASSPATH 并确保 GSSMANAGER_CLASS 属性值引用了 GSSManager 实现的完全限定类名称。 **操作：**确保 CLASSPATH 环境变量包含第三方 GSSManager 实现所需的所有 *.jar* 文件。
JZ0GS	出现通用安全服务 API 例外。 主要错误代码为 ____。 主要错误消息为 ____。 次要错误代码为 ____。 次要错误消息为 _____。 **操作：**检查主要和次要错误代码及错误消息。检查 Kerberos 配置。有关详细信息，请参见第 3 章 “安全性”。
JZ0H0	无法启动事件处理程序的线程；事件名称 = _____。 **操作：**请向 Sybase 技术支持部门报告此错误。

SQL 状态	消息 / 说明 / 操作
JZ0H1	收到事件通知但没有找到事件处理程序；事件名称 = _____。 **操作：**请向 Sybase 技术支持部门报告此错误。
JZ0HC	分析十六进制数时遇到非法字符 “_____”。 **说明：**用于表示二进制值的字符串包含了十六进制数限定范围 （0 - 9、 a - f）以外的字符。 **操作：**检查字符串中的字符值，确保它们在要求的范围内。
JZ0I1	I/O 层：读取流时出错。 **说明：**连接无法读取所请求的量。最可能的原因是，超出了语句超时期限而导致连接超时。 **操作：**增大语句的超时值。
JZ0I2	I/O 层：写入流时出错。 **说明：**连接无法写入所请求的输出。最可能的原因是，超出了语句超时期限而导致连接超时。 **操作：**增大语句的超时值。
JZ0I3	未知属性。 此消息表明存在内部产品问题。 请向 Sybase 技术支持部门报告此错误。 **操作：**表明存在内部产品问题。请向 Sybase 技术支持部门报告此错误。
JZ0I5	指定了无法识别的 CHARSET 属性：_____。 **说明：**您为 CHARSET 连接属性指定了一个不受支持的字符集编码。 **操作：**为该连接属性输入有效的字符集编码。请参见第 36 页的 “Connect 字符集转换程序”。
JZ0I6	将 UNICODE 转换为服务器使用的字符集时出错。 错误消息：_____ **操作：**在 jConnect 客户端为 CHARSET 连接属性选择另外的字符集编码，所选字符集编码应支持要发送到服务器的所有字符。可能也需要在服务器上安装另一个字符集。此外，如果您使用的是 jConnect 6.05 或更高版本以及 Adaptive Server Enterprise 12.5 或更高版本，则可以在向服务器发送数据时使用 unichar/univarchar 数据类型。请参见第 35 页的 “使用 jConnect 传递 Unicode 数据”。
JZ0I7	代理网关没有响应。 **说明：**级联或安全网关没有响应。 **操作：**确保网关正常安装和运行。
JZ0I8	代理网关连接被拒绝。 网关响应：_____ **说明：**PROXY 连接属性指定的 Web 服务器 / 网关拒绝了连接请求。 **操作：**检查代理上的访问和错误日志以确定拒绝连接的原因。确保代理是 JDBC 网关。
JZ0I9	此 InputStream 已关闭。 **说明：**您试图读取一个来自 getAsciiStream、 getUnicodeStream 或 getBinaryStream 的 InputStream，但该 InputStream 已关闭。该流被关闭的原因可能是您移到了另一列或取消了结果集，且没有足够的资源来缓存数据。 **操作：**增大高速缓存容量，或按顺序读取列。

SQL 状态	消息 / 说明 / 操作
JZ0IA	试图发送 _____ 时出现截断错误。 **说明：**在发送字符串之前进行字符集转换时发生截断错误。被转换的字符串的长度超过了分配给它的大小。 **操作：**在 jConnect 客户端为 CHARSET 连接属性选择另外的字符集编码，所选字符集编码应支持要发送到服务器的所有字符。可能也需要在服务器上安装另一个字符集。
JZ0IR	不能对已通过 java.io.Reader 在结果集中更新过的列调用 getXXX。 **操作：**删除对使用 Reader 更新过的 ResultSet 列的 getXXX 调用。
JZ0IS	不能对已在结果集中更新过的列调用 getXXXStream。 **说明：**更新结果集中的一个列之后，您试图用以下 SybResultSet 方法之一读取更新后的列值：getAsciiStream、 getUnicodeStream、 getBinaryStream。jConnect 不支持这种用法。 **操作：**不要试图从正在更新的列中读取输入流。
JZ0J0	偏移和 / 或长度值超出了实际的文本 / 图像长度。 **操作：**检查所用的偏移和 / 或长度值是否正确无误。
JZ0LC	不能对正在使用语言游标读取行的 ResultSet 调用 _____ 方法。 尝试将 LANGUAGE_CURSOR 连接属性设置为 false。 **说明：**应用程序试图对一个通过语言游标创建的 ResultSet 调用某种 ResultSet 游标滚动方法。 **操作：**参见错误消息。
JZ0NC	wasNull 调用前没有其它调用来获取列。 **说明：**只能在获取列的调用 （例如 getInt 或者 getBinaryStream）后调用 wasNull。 **操作：**更改代码，移动对 wasNull 的调用。
JZ0NE	错误的 URL 格式。 URL(U)：_____。 错误消息：_____ **操作：**检查 URL 的格式。确保端口号仅包含数值字符。
JZ0NF	无法装载 SybSocketFactory。 确保类名称拼写无误，软件包被完全指定，类在类路径中可用，而且有公共的零参数构造函数。 **操作：**参见消息文本。
JZ0P1	异常的结果类型。 **说明：**数据库返回的结果不能由语句返回给应用程序，或者应用程序此时需要的不是该结果。通常这表明应用程序在错误地使用 JDBC 执行查询或存储过程。如果 JDBC 应用程序连接到一个 Open Server 应用程序，这可能表明该 Open Server 应用程序中有错误，该错误导致 Open Server 发送异常结果序列。 **操作：**使用 com.sybase.utils.Debug(true, "ALL") 调试工具，尝试找出异常结果及其发生原因。
JZ0P4	协议错误。 此消息表明存在内部产品问题。 请向 Sybase 技术支持部门报告此错误。 **操作：**参见消息文本。

SQL 状态	消息 / 说明 / 操作
JZ0P7	没有对列进行高速缓存；请使用 RE-READABLE_COLUMNS 属性。 **说明：**当 REPEAT_READ 连接属性设置为 “false”时，尝试再次读取列或以错误顺序读取列。 当 REPEAT_READ 为 “false”时，只能读取一次某行的列值，且必须按列索引升序顺序读取。例如，在读取一行的第 3 列后，就不能再次读取该列的值，也不能读取该行第 2 列。 **操作：**要么将 REPEAT_READ 设置为 “true”，要么不要试图重复读取某列值并确保按列索引升序顺序进行读取。
JZ0P8	请求的 RSMDA 列类型名称未知。 **说明：**jConnect 无法在 ResultSetMetaData.getColumnTypeName 方法中确定列类型的名称。 **操作：**确保数据库具有用于元数据的最新的存储过程。
JZ0P9	检测到 COMPUTE BY 查询。 该类型的结果不受支持，已被取消。 **说明：**执行的查询返回的 COMPUTE 结果不受 jConnect 支持。 **操作：**更改查询或存储过程，使其不使用 COMPUTE BY。
JZ0PA	查询已被取消，相同的响应被放弃。 **说明：**可能有连接中的另一语句执行了取消操作。 **操作：**检查各语句中的 SQL 例外和警告链，以找出原因。
JZ0PB	服务器不支持请求的操作。 **说明：**当 jConnect 建立与服务器的连接时，它会将需要服务器支持的功能通知服务器，服务器再将自身支持的功能通知 jConnect。当应用程序请求的操作在最初的功能协商中被拒绝时，会发出此错误消息。 例如，如果数据库不支持动态 SQL 语句预编译，而代码调用 SybConnection.prepareStatement(*sql_stmt, dynamic*)，并且 *dynamic* 设置为 “true”，那么 jConnect 将生成此消息。 **操作：**修改代码，使其不请求不受支持的功能。
JZ0PC	查询中参数的数量和大小要求宽表支持。 但是服务器不提供这种支持，或者查询请求不是在登录序列中发出的。 如果想请求宽表支持，请尝试将 JCONNECT_VERSION 属性设置为 >=6。 **说明：**您正试图执行一个拥有大量参数的语句，但服务器没有进行相应配置来处理如此之多的参数。能引起此例外的参数数量因传送的数据类型不同而变化。在传送 481 或更少的参数时，肯定不会产生这种例外。 **操作：**必须在 Adaptive Server 12.5 或更高版本服务器中运行此查询。在连接到数据库时，请将 JCONNECT_VERSION 属性设置为 “6”。

SQL 状态	消息 / 说明 / 操作
JZ0PD	动态准备中的查询过大，需要宽表支持。 但是服务器不提供这种支持，或者查询请求不是在登录序列中发出的。 如果想请求宽表支持，请尝试将 JCONNECT_VERSION 属性设置为 >=6。 **说明：**您正试图执行一个拥有大量参数的动态预准备语句，但服务器没有进行相应配置来处理如此之多的参数。 **操作：**必须在 Adaptive Server 12.5 或更高版本服务器中运行此查询。在连接到数据库时，请将 JCONNECT_VERSION 属性设置为 “6”。
JZ0PE	游标声明中列的数目过大，或者游标声明本身过大，需要宽表支持。 但是服务器不提供这种支持，或者查询请求不是在登录序列中发出的。 如果想请求宽表支持，请尝试将 JCONNECT_VERSION 属性设置为 >=6。 **说明：**当 SELECT 语句试图从 255 个以上的列中返回数据时会出现此错误。或者 SELECT 语句的实际长度很大时 （超过大约 65500 个字符）也会出现此错误。 **操作：**必须在 Adaptive Server 12.5 版或更高版本中运行此查询。在连接到数据库时，请将 JCONNECT_VERSION 属性设置为 “6”。
JZ0PN	指定的端口号 _____ 超出范围。 端口号必须满足以下条件：0<= 端口号 <=65535。 **操作：**检查数据库 URL 中指定的端口号。
JZ0R0	结果集已关闭。 **说明：**已对结果集对象调用 ResultSet.close 方法，该结果集不能另作他用。 **操作：**修正代码，从而在每当关闭结果集时将 ResultSet 对象引用设置为 null。
JZ0R1	结果集处于 IDLE 状态，因为目前没有访问行。 **说明：**应用程序调用了 ResultSet.getXXX 列数据检索方法之一，但没有当前行；应用程序没有调用 ResultSet.next，或 ResultSet.next 返回 “false”，指示没有数据存在。 **操作：**确保先将 rs.next 设置为 “true”后再调用 rs.getXXX。
JZ0R2	查询的结果集不存在。 **说明：**您使用了 Statement.executeQuery，但语句没有返回任何行。 **操作：**对不返回任何行的语句使用 executeUpdate。
JZ0R3	列处于 DEAD 状态。 这是内部错误。 请将错误报告给 Sybase 技术支持。 **操作：**参见消息文本。
JZ0R4	列中不含文本指针。 该列不是文本 / 图像列，或该列为 NULL。 **说明：**不能更新值为 null 的 text/image 列。空 text/image 列中不含文本指针。 **操作：**确保不要更新或获取指向不支持文本 / 图像数据的列的文本指针。确保不要更新值为 null 的 text/image 列。先插入数据，再进行更新。
JZ0R5	ResultSet 当前的定位超出了最后一行。 此状态下不能通过执行 get* 操作读取数据。 **说明：**应用程序已将 ResultSet 行指针移到最后一行之后。该位置没有数据可读，所有 get* 操作均为非法。 **操作：**更改代码，使 ResultSet 定位在最后一行之后时，不再读取列数据。

SQL 状态	消息 / 说明 / 操作
JZ0RD	不能对已通过 deleteRow() 方法删除的行调用任何 ResultSet.get* 方法。 **说明：**应用程序试图从已经删除的行中检索数据。没有有效数据可供检索。 **操作：**更改代码，使应用程序不从已删除的行中检索数据。
JZ0RM	不能在调用 updateRow 或 deleteRow 之后调用 refreshRow。 **说明：**在用 SybCursorResult.updateRow 更新数据库中的行之后，或用 SybCursorResult.deleteRow 删除行之后，又用 SybCursorResult.refreshRow 刷新了数据库中的行。 **操作：**在更新或删除数据库中的行以后，不要再刷新该行。
JZ0S0	语句状态机：语句处于 BUSY 状态。 **说明：**出现此错误的唯一情况是在使用 Statement.setCursorname 方法时。如果应用程序试图在语句已在使用中且需要读取非游标结果时设置游标名称，则会出现此错误。 **操作：**在用语句执行任何查询之前先为其设置游标名称，或者在设置游标名称之前调用 Statement.cancel，以确保该语句不处于繁忙状态。
JZ0S1	语句状态机：正试图在 IDLE 语句中执行 FETCH。 **说明：**语句中发生内部错误。 **操作：**关闭该语句，打开另一个语句。
JZ0S2	语句对象已关闭。 **说明：**已对语句对象调用 Statement.close 方法，该语句不能另作他用。 **操作：**修正应用程序，从而每当关闭语句时都将语句对象引用设置为 null。
JZ0S3	此子类中不能使用继承的方法 _____。 **说明：**PreparedStatement 不支持 executeQuery(String)、 executeUpdate(String) 和 execute(String)。 **操作：**传递查询字符串时，请使用 Statement，不要使用 PreparedStatement。
JZ0S4	不能执行空 （零长度）查询。 **操作：**不要执行空查询 ("")。
JZ0S5	连接中的全局事务处于活动状态时不能使用本地事务方法 _____。 **说明：**使用分布式事务时可能会出现该例外。 **操作：**有关诊断此问题的详细信息，请参见 *JDBC 2.0 可选软件包*（旧称 “JDBC 2.0 标准扩展 API”）中的第 7 章 “分布式事务”。
JZ0S6	本地事务方法 _____ 不能用在系统 12 以前的 XAConnection 中。 **说明：**使用分布式事务时可能会出现该例外。 **操作：**有关诊断此问题的详细信息，请参见 *JDBC 2.0 可选软件包*（旧称 “JDBC 2.0 标准扩展 API”）中的第 7 章 “分布式事务”。
JZ0S8	SQL 查询中的转义序列格式有误：“_____”。 **说明：**此错误由错误的转义语法引起。 **操作：**查看 JDBC 文档，确保语法无误。

SQL 状态	消息 / 说明 / 操作
JZ0S9	不能执行空 （零长度）查询。 **操作：**不要执行空查询 ("")。
JZ0SA	预准备语句：未设置输入参数，索引：_____. **操作：**确保为每个输入参数赋值。
JZ0SB	参数索引超出范围：_____。 **说明：**您尝试获取、设置或注册一个参数，但参数数目已经超越了最大数目限制。 **操作：**检查查询中的参数数目。
JZ0SC	可调用语句：尝试将返回状态设置为 `InParameter`。 **说明：**您已经准备调用一个用于返回状态的存储过程，但却试图设置参数 1，该参数是返回状态。 **操作：**此类调用中可供设置的参数是从 2 开始的。
JZ0SD	未为输出参数找到注册的参数。 **说明：**这表明应用程序存在逻辑错误。您尝试对参数调用 getXXX 或 wasNull，但尚未读取任何参数，或没有输出参数。 **操作：**检查以确保应用程序已经对 CallableStatement 注册了输出参数，且语句已被执行，同时输出参数已读取。
JZ0SE	为 `setObject()` 指定了无效的对象类型。 **说明：**向 PreparedStatement.setObject 传递了非法类型参数。 **操作：**查看 JDBC 文档。参数必须是来自 java.sql.Types 的一个常数。
JZ0SF	未找到任何参数。 是否已发送查询？ **说明：**您试图对不含参数的语句设置参数。 **操作：**在设置参数前确保查询已发送。
JZ0SG	`RPC` 未返回与应用程序注册的参数相同数量的输出参数。 **说明：**如果通过调用 CallableStatement.registerOutParam 注册的参数多于在存储过程中声明为 “OUTPUT” 的参数，则会出现此错误。有关详细信息，请参见第 125 页的 “RPC 返回比已注册参数更少的输出参数”。 **操作：**检查存储过程和 registerOutParameter 调用。确保已将所有相关参数声明为 “OUTPUT” 参数。查看以下代码行： `create procedure yourproc (@p1 int OUTPUT, ...` **注释** 如果在使用 Adaptive Server Anywhere （以前称为 SQL Anywhere）的过程中收到此错误，请升级到 Adaptive Server Anywhere 版本 5.5.04。
JZ0SH	使用了静态函数转义，但服务器里找不到元数据访问程序信息。 **操作：**在使用静态函数转义之前，先安装元数据访问程序信息。
JZ0SI	此服务器不支持使用的静态函数转义 _____。 **操作：**不要使用此转义。

SQL 状态	消息 / 说明 / 操作
JZ0SJ	找不到有关此数据库的元数据访问程序信息。 **操作：**在进行元数据调用前，先安装元数据信息。
JZ0SK	此类数据库服务器不支持 oj 转义。 解决方法 : 使用服务器特定的外部连接语法 （如果支持）。 查阅服务器文档。 **操作：**读取错误消息。同时，请安装最新版本的 jConnect 元数据。
JZ0SL	不支持的 SQL 类型 _____。 **说明：**jConnect 不支持应用程序声明的参数类型。 **操作：**若有可能，尝试用其它类型声明该参数。不要使用 Types.NULL 或 PreparedStatement.setObject (null)。
JZ0SM	jConnect 未能执行存储过程，因为发送参数时发生了错误。 导致此问题的原因可能是服务器不支持特定的数据类型，或者 jConnect 未在连接时为该数据类型请求支持。 可以尝试将 JCONNECT_VERSION 连接属性设置为更高的值。 或者，若有可能，尝试以语言语句的形式发送过程执行指令。
JZ0SN	setMaxFieldSize：字段长度不能为负。 **操作：**在调用 setMaxFieldSize 时使用正值或零 （无限制）。
JZ0SO	无效的 ResultSet 并发类型：_____。 **操作：**确保声明的并发为 ResultSet.CONCUR_READ_ONLY 或 ResultSet.CONCUR_UPDATABLE。
JZ0SP	无效的 ResultSet 类型：_____。 **操作：**确保声明的 ResultSet 类型为 ResultSet.TYPE_FORWARD_ONLY 或 ResultSet.TYPE_SCROLL_INSENSITIVE。 jConnect 不支持 ResultSet.TYPE_SCROLL_SENSITIVE ResultSet 类型。
JZ0SQ	无效的 UDT 类型 _____。 **说明：**在调用 DatabaseMetaData.getUDTs 方法时，如果用户定义的类型不是 Types.JAVA_OBJECT、 Types.STRUCT 或 Types.DISTINCT， jConnect 便会抛出此例外。 **操作：**使用上述三种 UDT 之一。
JZ0SR	setMaxRows：最大行数不能为负。 **操作：**在调用 setMaxRows 时使用正值或零 （无限制）。
JZ0SS	setQueryTimeout：查询超时时间不能为负。
JZ0ST	jConnect 不能在查询中以文字参数的形式发送 Java 对象。 执行此查询时，请确保数据库服务器支持 Java 对象，且 LITERAL_PARAMS 连接属性设置为 false。
JZ0SU	Date 或 Timestamp 参数设置成 _____ 年，但服务器只支持 _____ 至 _____ 之间的年份值。 如果在 Adaptive Server Anywhere 上将数据发往日期、时间戳列或参数，最好以字符串的形式发送，让服务器进行转换。 **说明：**Adaptive Server Enterprise 和 Adaptive Server Anywhere 对 datetime 和 date 的取值范围有不同的规定。datetime 的年份值必须大于等于 1753。而 date 数据类型可以采用大于等于 1 的年份值。 **操作：**确保发送的 date/timestamp 值在可接受的范围内。

SQL 状态	消息 / 说明 / 操作
JZ0T2	监听器线程读错误。 **操作：**检查网络通信。
JZ0T3	读操作超时。 **说明：**超出了为读取查询响应所分配的时限。 **操作：**调用 Statement.setQueryTimeout 增大超时期限。
JZ0T4	写操作超时。 超时期限：_____ 毫秒。 **说明：**超出了为发送请求所分配的时限。 **操作：**调用 Statement.setQueryTimeout 增大超时期限。
JZ0T5	用于存储响应的高速缓存已满。 **操作：**为 STREAM_CACHE_SIZE 连接属性使用缺省值或更大的值。
JZ0T6	读取贯通 TDS URL 时出错。 **说明：**读取 URL 标头时贯通协议失败。 **操作：**检查为连接定义的 URL。
JZ0T7	监听器线程读错误 — 捕获 ThreadDeath。 检查网络连接。 **操作：**检查网络连接，尝试重新运行应用程序。如果线程仍被中止，请联系 Sybase 技术支持。
JZ0T8	收到未知请求的数据。 请向 Sybase 技术支持部门报告此错误。
JZ0T9	发送请求不同步。 请向 Sybase 技术支持部门报告此错误。 **操作：**参见消息文本。
JZ0TC	尝试在两种类型间进行非法转换。 **说明：**在 Java 类型和 SQL 类型间进行的转换失败。 **操作：**检查请求的类型转换，确保 JDBC 规范支持该转换。
JZ0TE	尝试在两种类型间进行非法转换。 有效的数据库类型为：“_____”。 **说明：**数据库列的数据类型和 ResultSet.getXXX 调用中请求的数据类型不能进行隐式转换。 **操作：**使用错误消息中列出的有效数据类型之一。
JZ0TI	jConnect 无法在 _____ 数据库类型和请求的 _____ 类型间进行有意义的转换。 **说明：**当类型无法转换时，例如应用程序试图对从数据库返回的 time 值调用 ResultSet.getObject(int, Types.DATE) 时，可能导致此种例外的发生。 **操作：**确保数据库的数据类型能隐式转换成要检索的对象类型。
JZ0TO	读操作超时。 **说明：**如果套接字读取超时，则会发生此例外。 **操作：**调用 Statement.setQueryTimeout 增大超时期限。同时，检查正在执行的查询或存储过程以确定超时原因。

SQL 状态	消息 / 说明 / 操作
JZ0TS	试图发送 _____ 时出现截断错误。 **说明：**应用程序指定的字符串的长度大于应用程序要发送的字符串长度。因此，字符串被截断为声明的长度。 **操作：**正确设置长度以避免发生截断。
JZ0US	设置了 SybSocketFactory 连接属性，PROXY 连接属性被设置为服务器小程序的 URL。jConnect 驱动程序不支持这种组合。 如果想从在浏览器内运行的小程序中发送安全 HTTP，请使用以 “https://” 开头的代理 URL。 **操作：**参见消息文本。
JZ0XC	_____ 是无法识别的事务协调器类型。 **说明：**元数据信息指示服务器支持分布式事务，但 jConnect 不支持所用的协议。 **操作：**检验是否安装了最新的元数据脚本。如果此错误仍然存在，请联系 Sybase 技术支持。
JZ0XS	服务器不支持 XA 类型的事务。 请确保事务功能已在服务器上启用并得到许可。 **说明：**jConnect 尝试连接的服务器不支持分布式事务。 **操作：**不要为此服务器使用 XADataSource，或者对服务器进行升级或配置以使其支持分布式事务。
JZ0XU	当前用户无权执行 XA 类型的事务。 确保用户具有 _____ 角色。 **说明：**连接到数据库的用户无权执行分布式事务，最可能的原因是该用户不具备适当的角色 （空白处显示的角色）。 **操作：**按错误消息的提示赋予用户相应的角色，或让具有该角色的另一用户执行事务。
S0022	无效的列名称 “_____”。 **说明：**您试图通过名称引用列，但与该名称对应的列不存在。 **操作：**检查列名的拼写是否正确。
ZZ00A	方法 _____ 尚未完成，不应调用它。 **说明：**您试图使用未实现的方法。 **操作：**查阅您的 jConnect 版本附带的发行公告以获取更多信息。也可以访问位于 http://www.sybase.com 的 jConnect Web 页，查看是否有最新版本的 jConnect 实现了该方法。如果没有，请不要使用该方法。

附 录 B

jConnect 示例程序

本附录是 jConnect 示例程序指南，包含以下主题：

主题	页码
运行 IsqlApp	173
运行 jConnect 示例程序和代码	175

运行 IsqlApp

IsqlApp 允许从命令行发出 isql 命令，而且允许运行 jConnect 示例程序。

IsqlApp 的语法是：

```
IsqlApp [-U username]
   [-P password]
   [-S servername]
   [-G gateway]
   [-p {http|https}]
   [-D debug_class_list]
   [-v]
   [-I input_command_file]
   [-c command_terminator]
   [-C charset]
   [-L language]
   [-K service_principal_name]
   [-F JAAS_login_config_file_path]
   [-T sessionID]
   [-V <version {2,3,4,5}>]
```

参数	说明
-U	用于连接到服务器的登录 ID。
-P	指定的登录 ID 的口令。
-S	要连接的服务器的名称。
-G	网关地址。对于 HTTP 协议， URL 是：http://*host*:*port*。 若要使用支持加密的 HTTPS 协议， URL 应为 https://*host*:*port*/*servlet_alias*。
-p	指定是要使用 HTTP 协议还是使用支持加密的 HTTPS 协议。
-D	为所有类或指定的类打开调试程序，类之间用逗号分隔。例如， *-D ALL* 显示所有类的调试输出。 *-D SybConnection, Tds* 仅显示 SybConnection 和 Tds 类的调试输出。
-v	打开详细输出以显示或打印输出。
-I	让 IsqlApp 接受来自文件而不是来自键盘的命令。 在此参数之后，指定用于 IsqlApp 输入的文件的名称。该文件必须包含命令终结符（缺省为 “go”）。
-c	允许指定一个关键字 （如 “go”），当独占一行输入该关键字时可终止命令。这使得在使用终结符关键字之前可以输入多行命令。如果不指定命令终结符，每个新行都会终止命令。
-C	为通过 TDS 的字符串指定字符集。 如果不指定字符集， IsqlApp 将使用服务器的缺省字符集。
-L	为 jConnect 消息和显示从服务器返回的错误消息指定语言。
-K	表示用户想要通过 Kerberos 登录进入 ASE。此参数设置服务主体名称。例如： `-K myASE` 本示例表示希望执行 Kerberos 登录，且服务器的服务主体名称为 myASE。 有关详细信息，请参见第 3 章 “安全性”。
-F	指定 JAAS 登录配置文件的路径。如果使用 -K 选项，则必须设置此属性。例如： `-F /foo/bar/exampleLogin.conf` 请参见 jConnect 安装的 *sample2* 目录中的 *ConnectKerberos.java* 示例。有关详细信息，请参见第 3 章 “安全性”。
-T	如果设置了此参数， jConnect 将假定某个应用程序正试图在由 TDS 贯通网关保持打开的现有 TDS 会话上重新开始通信。 jConnect 将跳过登录协商并将所有来自应用程序的请求转发到指定的会话 ID。
-V	启用特定于版本的使用特性。请参见第 10 页的 “JCONNECT_VERSION 连接属性”。

注释 必须在每个选项标志后面输入一个空格。

若要获取命令行选项的完整描述，请输入：

```
java IsqlApp -help
```

下面的示例显示了如何通过端口 “3756”连接到主机 “myserver”上的数据库，并运行名为 “myscript”的 isql 脚本：

```
java IsqlApp -U sa -P sapassword
   -S jdbc:sybase:Tds:myserver:3756
   -I $JDBC_HOME/sp/myscript -c run
```

注释　提供了对 isql 命令进行 GUI 访问的小程序，如下所示：

$JDBC_HOME/sample2/gateway.html (UNIX)
%JDBC_HOME%\sample2\gateway.html (Windows)

运行 jConnect 示例程序和代码

jConnect 包含多个示例程序，这些示例程序说明了本章所涉及的多个主题，旨在帮助您理解 jConnect 如何处理各种 JDBC 类和方法。此外，本节还提供一个示例代码段作为参考。

示例应用程序

安装 jConnect 时，还可以安装示例程序。这些示例包含源代码，因此可以查看 jConnect 是如何实现各种 JDBC 类和方法的。有关安装示例程序的完整指导，请参见 *jConnect for JDBC 安装指南*。

注释　jConnect 示例程序仅用于示范。

示例程序安装在 jConnect 安装目录的 *sample2* 子目录下。 *sample2* 子目录中的文件 *index.html* 包含可用示例的完整列表以及每个示例的说明。*index.html* 还允许将示例程序作为小程序查看和运行。

运行示例小程序

使用 Web 浏览器，就可以将有些示例程序作为小程序运行。这使您可在查看输出结果的同时查看源代码。

若要将示例作为小程序运行，需要启动 Web 服务器网关。

使用 Web 浏览器打开 *index.html*：

http://localhost:8000/sample2/index.html

运行示例程序和 Adaptive Server Anywhere

所有示例程序均与 Adaptive Server 兼容，但只有有限数目的示例程序与 Adaptive Server Anywhere 兼容。若要获得与 Adaptive Server Anywhere 兼容的示例程序的当前列表，请参见 *sample2* 子目录中的 *index.html*。

若要运行可用于 Adaptive Server Anywhere 的示例程序，必须在 Adaptive Server Anywhere 服务器上安装 *pubs2_any.sql* 脚本。此脚本位于 *sample2* 子目录中。

对于 Windows，请进入 DOS 命令窗口并输入：

```
java IsqlApp -U dba -P password
 -S jdbc:sybase:Tds:[hostname]:[port]
 -I %JDBC_HOME%\sample2\pubs2_any.sql -c go
```

在 UNIX 系统中，输入：

```
java IsqlApp -U dba -P password
 -S jdbc:sybase:Tds:[hostname]:[port]
 -I $JDBC_HOME/sample2/pubs2_any.sql -c go
```

示例代码

下面的代码说明了如何调用 jConnect 驱动程序、进行连接、发出 SQL 语句，以及处理结果。

```
import java.io.*;
 import java.sql.*;

   public class SampleCode
   {
       public static void main(String args[])
       {
           try
           {
           /*
            * Open the connection. May throw a SQLException.
            */
              DriverManager.registerDriver(
              (Driver) Class.forName(
               "com.sybase.jdbc3.jdbc.SybDriver").newInstance());

               Connection con = DriverManager.getConnection(
                  "jdbc:sybase:Tds:myserver:3767", "sa", "");
           /*
            * Create a statement object, the container for the SQL
            * statement. May throw a SQLException.
            */
              Statement stmt = con.createStatement();
           /*
            * Create a result set object by executing the query.
            * May throw a SQLException.
            */
              ResultSet rs = stmt.executeQuery("Select 1");
           /*
            * Process the result set.
            */

              if (rs.next())
              {
                 int value = rs.getInt(1);
                 System.out.println("Fetched value " + value);
              }
             rs.close()
             stmt.close()
             con.close()
          }//end try
      /*
```

```
        * Exception handling.
        */
           catch (SQLException sqe)
           {
               System.out.println("Unexpected exception : " +
                           sqe.toString() + ", sqlstate = " +
                               sqe.getSQLState());
               System.exit(1);
           }//end catch


           catch (Exception e)
           {
               e.printStackTrace();
               System.exit(1);
           }//end catch

           System.exit(0);
       }
   }
```

索引

符号

.jar 文件
  预装载 85

英文

Adaptive Server Anywhere 21
  euro 符号 41
  SERVICENAME 连接属性 23
  存储和检索 Java 对象 77
  发送图像数据 64, 68
  访问元数据 49
APPLICATIONNAME 连接属性 13
ASE 数据类型
  date、 time 和 datetime 68
BE_AS_JDBC_COMPLIANT_AS_POSSIBLE 连接属性 13
BigDecimal 范围重设
  改善驱动程序性能 128
bigint
  支持的数据类型 99
CACHE_COLUMN_METADATA 连接属性 13
CANCEL_ALL 连接属性 6, 11, 13
CAPABILITY_TIME 连接属性 14
CAPABILITY_WIDETABLE 连接属性 14
CHARSET 连接属性 6, 15
  设置 37
CHARSET_CONVERTER 连接属性 6
CHARSET_CONVERTER_CLASS 连接属性 15, 37
CLASS_LOADER 连接属性 15
CLASSPATH
  为调试程序设置 119
compute 语句 100
CONNECTION_FAILOVER 连接属性 15, 25
Debug 类 118
DISABLE_UNICHAR_SENDING 连接属性 16
DISABLE_UNPROCESSED_PARAM_WARNINGS 连接属性 16
DYNAMIC_PREPARE 连接属性 16
ESCAPE_PROCESSING_DEFAULT 连接属性 135
ESCAPE_PROCESSING_DEFAULT 属性 16
EXPIRESTRING 连接属性 16
FAKE_METADATA 连接属性 17
GET_BY_NAME_USES_COLUMN_LABEL 连接属性 17
GSSMANAGER_CLASS 连接属性 17
HOSTNAME 连接属性 18
HOSTPROC 连接属性 18
HTTP 141
IGNORE_DONE_IN_PROC 连接属性 18
IS_CLOSED_TEST 连接属性 18
isql 小程序
  运行此示例 148
IsqlApp 实用程序 173
Java 对象
  在 ASA 6.0 中存储和检索 77
  作为列数据存储在表中 77
jConnect
  调试程序 117
  调用 11
  定义 2
  改善性能 127
  设置 5
  设置连接属性 12
  使用游标 50
  示例程序 175
  网关 141
  应用程序中的内存问题 124
jConnect 应用程序中的内存问题 124
JCONNECT_VERSION 连接属性 10, 18
JDBC
  定义 1
  接口 1
  驱动程序类型 2

约束 , 限制 , 和偏差 95
JDBC 2.0
标准扩展 85
可选软件包扩展支持 85
JDBC 驱动程序
JDBC-ODBC 桥 2
native-API/partly-Java 2
native-protocol/all-Java 2
net-protocol/all-Java 2
JDBC 数据类型
date、 time 和 timestamp 68
jdbc.drivers 12
JNDI
环境信息 29
使用 25
用于命名数据库 86
LANGUAGE 连接属性 5, 18
LANGUAGE_CURSOR 136
LANGUAGE_CURSOR 连接属性 18
LITERAL_PARAMS 连接属性 18
native-API/partly-Java 驱动程序 2
native-protocol/all-Java 驱动程序 2
net-protocol/all-Java 驱动程序 2
PACKETSIZE 连接属性 19
password 19
PRELOAD_JARS 连接属性 19
PreparedStatement
使用游标 56
PROTOCOL_CAPTURE 连接属性 19
PROXY 连接属性 19
PureConverter 类 37
QUERY_TIMEOUT_CANCELS_ALL 连接属性 19
REMOTEPWD 连接属性 19
REPEAT_READ 128
REPEAT_READ 连接属性 19
REQUEST_HA_SESSION 20
REQUEST_KERBEROS_SESSION 20
RMNAME 连接属性 20
rs.getByte() 69
SECONDARY_SERVER_HOSTPORT 连接属性 20
SELECT_OPENS_CURSOR 连接属性 21
SERIALIZE_REQUESTS 连接属性 21
SERVER_INITIATED_TRANSACTIONS 连接属性 21
SERVERTYPE 连接属性 21
SERVICE_PRINCIPAL_NAME 连接属性 21
SERVICENAME 连接属性 21
SESSION_ID 连接属性 22
SESSION_TIMEOUT 连接属性 22
setRemotePassword() 47
SkipDoneProc 服务器小程序参数 151
SQL 例外与警告消息 155
SQLINITSTRING 连接属性 22
Statement.cancel() 方法 11
STREAM_CACHE_SIZE 连接属性 22
Sybase 扩展更改 138
SybEventHandler 70
SybMessageHandler 74
SYBSOCKET_FACTORY 连接属性 22
TDS 3
安装服务器小程序 151
捕获通信 121
服务器小程序 142
服务器小程序系统要求 151
跟踪会话 153
贯通 142
恢复会话 153
设置服务器小程序参数 151
TdsResponseSize 服务器小程序参数 151
TdsSessionIdleTimeout 服务器小程序参数 151
TEXTSIZE 22
TruncationConverter 类 37, 41
TYPE_SCROLL_INSENSITIVE 限制 58
unichar 6
unichar 和 univarchar 数据类型 35
unitext 99
unsigned int
支持的数据类型 99
URL
连接属性参数 24
语法 23
USE_METADATA 连接属性 18
VERSIONSTRING 连接属性 22
Web 服务器网关 141
XAServer 91

A

安装
TDS 服务器小程序 151

错误消息处理程序 75

B

本地化 35
捕获 TDS 通信 121

C

处理
  错误消息 73
创建游标 50
存储过程
  错误 125
  通过结果集更新数据库 63
  执行 100
错误
  存储过程 125
  连接 122, 123
错误消息
  SQL 例外与警告 155
  Sybase 特定的 73
  安装错误消息处理程序 75
  处理 73
  错误消息处理程序示例 75
  自定义处理 74

D

调试程序 117
  方法 119
  获取 Debug 类的一个实例 117
  设置 CLASSPATH 119
  在应用程序中打开 118
  在应用程序中关闭 118
调试服务器小程序参数 152
调用 jConnect 11
定位型更新和删除
  使用 JDBC 1.x 方法 52
  使用 JDBC 2.0 方法 54
动态类装载 81
读者 vii
多线程
  进行调节 95
多字节字符集
  支持的 38
  转换程序类 36

F

发送图像数据 64
非序列化 84
分布式事务支持 91, 92
服务器到服务器的远程过程调用 47
服务器小程序 141
  TDS 142
服务器小程序参数
  SkipDoneProc 151
  TdsResponseSize 151
  TdsSessionIdleTimeout 151
  调试 152

G

高级功能 70
高可用性 (HA) 支持 42
跟踪 TDS 会话 153
更新
  数据库通过存储过程的结果集 63
故障排除 117
贯通
  TDS 142
归集连接 89
国际化 35

H

恢复
  TDS 会话 153
货币符号 , euro 41

J

将 Java 对象作为列数据存储在表中 77
将 Java 对象发送到数据库 78
接收来自数据库的 Java 对象 79
前提条件 77
接口 , JDBC 1

K

宽表 48
扩展更改 , Sybase 138

L

连接
错误 122, 123
归集 89
网关连接被拒绝 122
连接到
服务器使用 JNDI 25
中删除语言 23
连接属性
APPLICATIONNAME 13
BE_AS_JDBC_COMPLIANT_AS_POSSIBLE 13
CACHE_COLUMN_METADATA 13
CANCEL_ALL 6, 11, 13
CAPABILITY_TIME 14
CAPABILITY_WIDETABLE 14
CHARSET 6, 15
CHARSET_CONVERTER 6
CHARSET_CONVERTER_CLASS 15, 37
CLASS_LOADER 15
CONNECTION_FAILOVER 15, 25
DISABLE_UNICHAR_SENDING 16
DISABLE_UNPROCESSED_PARAM_WARNINGS 16
DYNAMIC_PREPARE 16
ESCAPE_PROCESSING_DEFAULT 16, 135
EXPIRESTRING 16
FAKE_METADATA 17
GET_BY_NAME_USES_COLUMN_LABEL 17
GSSMANAGER_CLASS 17
HOSTNAME 18
HOSTPROC 18
IGNORE_DONE_IN_PROC 18
IS_CLOSED_TEST 18
JCONNECT_VERSION 10, 18
LANGUAGE 5, 18
LANGUAGE_CURSOR 18, 136
LANGUAGE_CURSOR 和游标性能 136
LITERAL_PARAMS 18
PACKETSIZE 19
password 19
PRELOAD_JARS 19
PROTOCOL_CAPTURE 19
PROXY 19
QUERY_TIMEOUT_CANCELS_ALL 19
REMOTEPWD 19
REPEAT_READ 19, 128
REQUEST_HA_SESSION 20
REQUEST_KERBEROS_SESSION 20
RMNAME 20
SECONDARY_SERVER_HOSTPORT 20
SELECT_OPENS_CURSOR 21
SERIALIZE_REQUESTS 21
SERVER_INITIATED_TRANSACTIONS 21
SERVERTYPE 21
SERVICE_PRINCIPAL_NAME 21
SERVICENAME 21
SESSION_ID 22
SESSION_TIMEOUT 22
SQLINITSTRING 22
STREAM_CACHE_SIZE 22
SYBSOCKET_FACTORY 22
TEXTSIZE 22
URL 中的设置 24
USE_METADATA 18
VERSIONSTRING 22
设置 12
用户 22
列
游标结果集中的删除 53
在游标结果集中更新 54

O

欧洲货币符号 41

P

批处理更新 62
存储过程 61

Q

迁移 jConnect 应用程序
jConnect 应用程序 , 迁移 137
轻量目录访问协议 (LDAP) 25
驱动程序
JDBC 类型 2
属性 13

S

设置
jConnect 5
jConnect 连接属性 12
TDS 服务器小程序参数 151
实用程序
IsqlApp 173
示例程序 175
事件通知 70
示例 71
属性
驱动程序 13
数据
图像 64
数据库
将 Java 对象作为列数据存储在表中 77
用于命名的 JNDI 86
数据类型
ASE date、 time 和 datetime 68
JDBC date、 time 和 timestamp 数据类型 68
unichar 和 univarchar 35

T

图像数据
TextPointer 类中的公共方法 65
发送 64
获取 TextPointer 对象 66
使用 TextPointer.sendData 执行更新 66
使用 TextPointer.sendData() 更新列 65

W

网关 141
连接被拒绝 122
配置 142

X

系统属性
jdbc.drivers 12
相关文档 vii
小程序 147
行
从游标结果集中删除 55
在游标结果集中插入 56
性能 , 改善
BigDecimal 范围重设 128
游标 136
字符集转换 129
性能 , 改善 127
对动态 SQL 中的预准备语句的调优 129
选择字符集转换程序类 37

Y

应用程序
打开调试程序 118
关闭调试程序 118
用户 22
游标 50
创建 50
用于 PreparedStatement 56
游标结果集
插入行 56
定位型更新 53
更新列 54
删除 53

删除行 55
使用 JDBC 1.x 方法进行定位型更新和删除 52
使用 JDBC 2.0 方法进行定位型更新和删除 54
用于更新数据库的方法 54
游标性能 136
和 LANGUAGE_CURSOR 连接属性 136
与 JDBC 标准的偏差 95
语法约定 ix
预装载 .jar 文件 85
元数据
USE_METADATA 18
访问 49
服务器端实现 49
远程过程调用 (RPC)
服务器到服务器 47
约定 ix

Z

在应用程序中打开调试程序 118
在应用程序中关闭调试程序 118
支持的数据类型 99
中删除语言
连接到 23
连接示例 24
字符集
设置 37
支持的 38
转换程序类 36
字符集转换
改善驱动程序性能 129
改善性能 38
字符集转换程序类
PureConverter 37
TruncationConverter 36
选择 37
字体约定 ix