iAnywhere SOLUTIONS™
A SYBASE COMPANY

Native UltraLite™ for Java
用户指南

部件号：DC32103-01-0902-01
上次修改时间：2005 年 1 月

目录

关于本手册

主题

本手册介绍 Native UltraLite for Java。利用 Native UltraLite for Java，可以开发数据库应用程序，并将其部署到运行 Jeode 或 CrEme VM 的 Windows CE 设备。

读者

凡是希望在数据存储和同步过程中利用 UltraLite 关系数据库数据存储和同步过程的高性能、资源效率、稳健性和安全性的 Java 应用程序开发人员，都适于使用本手册。

SQL Anywhere Studio 文档

本手册是 SQL Anywhere 文档集的一部分。本节介绍文档集中包含的手册及其使用方法。

SQL Anywhere Studio 文档

我们以多种形式提供了 SQL Anywhere Studio 文档：联机文档，将所有手册都合并到一个大帮助文件中； PDF 文件，每本手册有一个单独的 PDF 文件；以及可以购买到的印刷手册。这些文档包括以下手册：

- **SQL Anywhere Studio 介绍** 这本手册概述了 SQL Anywhere Studio 数据库管理和同步技术。手册中提供了一些教程，分别介绍 SQL Anywhere Studio 的各个组成部分。
- **SQL Anywhere Studio 的新功能** 这本手册面向的读者是该软件旧版本的用户。该手册列出了本产品此版本和以前版本的新功能，并介绍了升级步骤。
- **Adaptive Server Anywhere 数据库管理指南** 这本手册介绍有关运行、管理和配置数据库和数据库服务器等方面的内容。
- **Adaptive Server Anywhere SQL 用户指南** 这本手册介绍如何设计和创建数据库；如何导入、导出和修改数据；如何检索数据以及如何建立存储过程和触发器。
- **Adaptive Server Anywhere SQL 参考手册** 这本手册为 Adaptive Server Anywhere 使用的 SQL 语言提供了一套完整的参考资料。还介绍了 Adaptive Server Anywhere 系统表和过程。
- **Adaptive Server Anywhere 编程指南** 这本手册介绍如何使用 C、C++ 和 Java 编程语言建立和部署数据库应用程序。使用 Visual Basic 和 PowerBuilder 等工具的用户可以使用这些工具提供的编程接口。它还介绍了 Adaptive Server Anywhere ADO.NET 数据提供程序。
- **Adaptive Server Anywhere SNMP Extension Agent 用户指南** 这本手册介绍如何配置 Adaptive Server Anywhere SNMP Extension Agent，以使用 SNMP 管理应用程序来管理 Adaptive Server Anywhere 数据库。

- **Adaptive Server Anywhere 错误消息** 这本手册提供了 Adaptive Server Anywhere 错误消息及其诊断信息的完整列表。

- **SQL Anywhere Studio 安全指南** 这本手册提供有关 Adaptive Server Anywhere 数据库中的安全功能的信息。 Adaptive Server Anywhere 7.0 荣获美国政府授予的 TCSEC （可信计算机系统评估标准） C2 安全等级。对于那些希望以相当于 C2 认证环境的方式运行当前版本的 Adaptive Server Anywhere 的用户，他们可能会对这本手册很感兴趣。

- **MobiLink 管理指南** 这本手册介绍如何使用用于移动计算的 MobiLink 数据同步系统，该系统支持在单个 Oracle、 Sybase、 Microsoft 或 IBM 数据库与多个 Adaptive Server Anywhere 或 UltraLite 数据库之间共享数据。

- **MobiLink 客户端** 这本手册介绍如何设置和同步 Adaptive Server Anywhere 及 UltraLite 远程数据库。

- **MobiLink 服务器启动同步用户指南** 这本手册介绍 MobiLink 服务器启动的同步，利用这项 MobiLink 功能，可从统一数据库启动同步。

- **MobiLink 教程** 这本手册提供多个教程来指导您如何设置和运行 MobiLink 应用程序。

- **QAnywhere 用户指南** 这本手册介绍 MobiLink QAnywhere。该产品是一个消息传送平台，支持开发和部署用于移动和无线客户端，以及传统桌面和便携式客户端的消息传送应用程序。

- **iAnywhere Solutions ODBC 驱动程序** 这本手册介绍如何设置 ODBC 驱动程序以从 MobiLink 同步服务器和 Adaptive Server Anywhere 远程数据访问功能访问除 Adaptive Server Anywhere 之外的统一数据库。

- **SQL Remote 用户指南** 这本手册介绍用于移动计算的 SQL Remote 数据复制系统的各个方面，此系统支持在单个 Adaptive Server Anywhere 或 Adaptive Server Enterprise 数据库与多个 Adaptive Server Anywhere 数据库之间共享数据 （使用电子邮件或文件传输等间接链接）。

- **SQL Anywhere Studio 帮助** 这本手册包含 Sybase Central、Interactive SQL 和其它图形工具的上下文相关帮助。印刷文档集中没有这本手册。
- **UltraLite 数据库用户指南** 这本手册面向的读者为所有 UltraLite 开发人员。它介绍了 UltraLite 数据库系统并提供了所有 UltraLite 编程接口的公共信息。
- **UltraLite 接口指南** 对于各个 UltraLite 编程接口都单独提供了一本手册。这些接口中有一些是作为 UltraLite 组件提供的，以便于快速开发应用程序；另外一些是作为静态接口提供的，以便于 C、C++ 和 Java 开发。

除本文档集外， PowerDesigner 和 InfoMaker 还有自己的联机文档。

文档格式

SQL Anywhere Studio 按下列形式提供文档：

- **联机文档** 这些联机文档包含完整的 SQL Anywhere Studio 文档，其中包括手册和 SQL Anywhere 工具的上下文相关帮助。每次发布产品的维护版本时都会对联机文档进行更新，因此联机文档是最全和最新的文档来源。

  若要在 Windows 操作系统中阅读联机文档，请选择 [开始] → [程序] → [SQL Anywhere 9] → [联机手册]。可以使用左窗格中的 HTML 帮助目录、索引和搜索功能，以及右窗格中的链接和菜单来浏览联机文档。

  若要在 UNIX 操作系统上阅读联机文档，请查看 SQL Anywhere 安装目录下的 HTML 文档。

- **PDF 手册** 以一组 PDF 文件的形式提供的 SQL Anywhere 手册，可以使用 Adobe Acrobat Reader 进行查看。

  可通过联机手册或 Windows [开始] 菜单来查看 PDF 手册。

- **印刷手册** 可从 Sybase 销售部门或 eShop 购买整套文档集，Sybase 网上商店的网址为 http://eshop.sybase.com/eshop/documentation。

文档约定

本节列出了文档中使用的印刷和图形约定。

语法约定

在 SQL 语法说明中，使用了以下约定：

- **关键字** 所有 SQL 关键字都以大写字母显示，就像下例中的 ALTER TABLE：

  ALTER TABLE [*owner*.]*table-name*

- **占位符** 对于必须替换为相应的标识符或表达式的项，其显示方式就像下例中的 *owner* 和 *table-name*：

  ALTER TABLE [*owner*.]*table-name*

- **重复项** 对于重复项列表，只显示列表中的一个元素，后跟省略号 （三个句点），如以下示例中的 *column-constraint*：

  ADD *column-definition* [*column-constraint, …*]

  允许指定一个或多个列表元素。在此示例中，如果指定了多个元素，则必须用逗号将它们隔开。

- **可选部分** 语句的可选部分放在方括号内。

  RELEASE SAVEPOINT [*savepoint-name*]

  上面的方括号表示 *savepoint-name* 是可选的。不应键入方括号。

- **选项** 如果不必选择项目列表的项目或者只能选中一个，则用垂直条分隔这些项目，并将列表放在方括号内。

  [**ASC** | **DESC**]

  例如，可以从 ASC、 DESC 中任选一个，或者一个也不选。不应键入方括号。

- **二选一选项** 如果必须明确选择一个选项，则会将替换选项放在大括号内，并用一条竖线分隔选项。

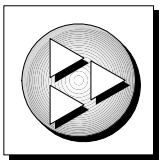
  [**QUOTES** { **ON** | **OFF** }]

如果使用了 QUOTES 选项，则必须提供 ON 和 OFF 两者之一。不应键入方括号和大括号。

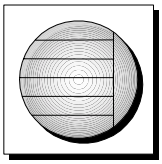
图标

本文档中使用了下列图标。

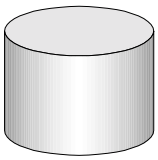
- 客户端应用程序。



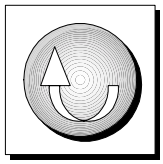
- 数据库服务器，例如 Sybase Adaptive Server Anywhere。



- 数据库。在某些高水平的图中，可以使用此图标表示数据库和管理数据库的数据库服务器。



- 复制或同步中间件。用于帮助在数据库之间共享数据。例如 MobiLink 同步服务器和 SQL Remote 消息代理。



- 编程接口。

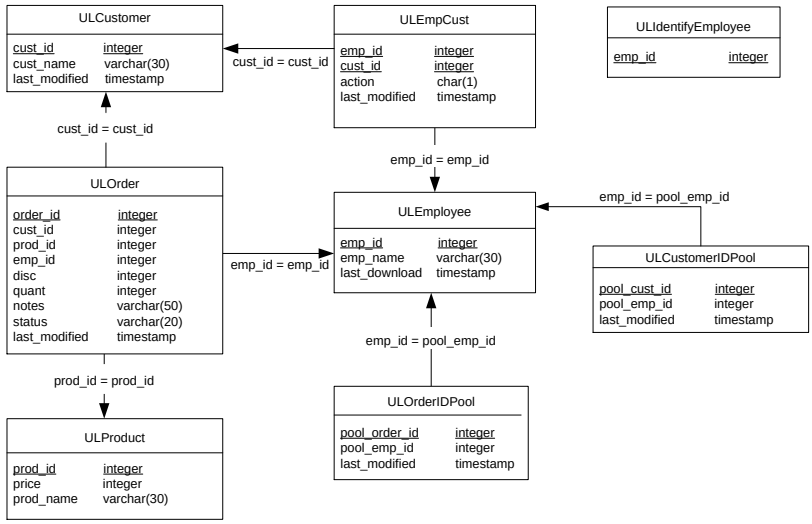
CustDB 示例数据库

MobiLink 和 UltraLite 文档中的许多示例均使用 UltraLite 示例数据库。

UltraLite 示例数据库的参考数据库包含在名为 *custdb.db* 的文件中，位于 SQL Anywhere 目录的 *Samples\UltraLite\CustDB* 子目录下。此外，还提供了在此数据库上创建的一个完整应用程序。

该示例数据库是一家硬件供应商的销售状态数据库。其中包含该供应商的客户、产品和销售人员信息。

下图显示 CustDB 数据库中的表以及这些表之间的相互关系。



查找详细信息并提供反馈

查找详细信息

附加信息和资源，包括交换代码，可从 iAnywhere Developer Network 获得，网址是 http://www.ianywhere.com/developer/。

如果您有问题或是需要帮助，可将邮件发布到下面所列的 iAnywhere Solutions 新闻组。

当您向这些新闻组发布邮件时，请提供问题的详细信息，包括 SQL Anywhere Studio 版本的编译版本号。您可以在命令提示符下键入 **dbeng9 -v** 来得到这些信息。

新闻组位于 *forums.sybase.com* 新闻服务器上。这些新闻组包括：

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

新闻组免责声明

iAnywhere Solutions 没有义务为其新闻组提供解决方案、信息或建议；除提供系统操作员监控服务和确保新闻组的运行和可用性外，iAnywhere Solutions 没有义务提供任何其它服务。

如果时间允许， iAnywhere Solutions 技术顾问和其他员工也会提供新闻组服务。他们是在自愿的基础上提供帮助的，所以可能无法定期提供解决方案和信息。他们可以提供多少帮助取决于他们的工作量。

反馈

我们欢迎您就本文档提出意见、建议或其它反馈信息。

您可以将意见和建议通过电子邮件发送到 SQL Anywhere 文档小组，地址为 iasdoc@ianywhere.com。虽然我们不回复该地址收到的电子邮件，但我们会认真阅读所有建议。

此外，您还可以通过上面所列的新闻组就文档和软件提供反馈信息。

第 1 章

Native UltraLite for Java 介绍

关于本章

本章介绍了 Native UltraLite for Java。它假定您熟悉 『UltraLite 数据库用户指南』> 「欢迎使用 UltraLite」中介绍的 UltraLite 的功能。

有关使用 有关使用 Native UltraLite for Java 创建应用程序的详细信息，请参见 『Native UltraLite for Java 用户指南』> 「了解 Native UltraLite for Java 开发」。

有关介绍 UltraLite for C/C++ 的有关介绍 Native UltraLite for Java 的实践教程，请参见 『Native UltraLite for Java 用户指南』> 「教程：构建 Native UltraLite for Java 应用程序」或 『Native UltraLite for Java 用户指南』> 「教程：CustDB 示例应用程序」。

UltraLite 与 Java

希望利用 UltraLite 数据库功能的 Java 开发人员有两种选择：

- Static Java API 是 『UltraLite Static Java 用户指南』> 「UltraLite Static Java 用户指南」中介绍的纯 Java UltraLite 技术。
- 另一个选择是 Native UltraLite for Java （在本书中介绍），它提供一个 Native UltraLite for Java 程序包以及一个本机 (C++) UltraLite 运行库。这样就将 Java 开发的优点与本地应用程序的性能结合到了一起，并且还可以访问操作系统特有的功能 （例如 ActiveSync 同步）。

Native UltraLite for Java 与 UltraLite for Java 在以下方面存在差别：

- **本机组件** 对于 Native UltraLite for Java 而言，其 UltraLite 运行库使用本机方法。也就是说，它不是用 Java 而是用 C++ 编写的，并编译成基础 CPU 和操作系统特有的二进制形式。相比之下，Static Java API 的 UltraLite 运行时是纯粹的 Java 实现。
- **组件 API** Native UltraLite for Java 与 UltraLite 组件套的其它成员共享 API 功能和结构。 Static Java API 使用 JDBC 作为编程接口。
- **Windows CE 部署** Native UltraLite for Java 将 Windows CE 作为部署目标。它支持许多 Windows CE 设备附带的 Jeode VM 及 CrEme VM，也支持 ActiveSync 同步。

  有关平台支持的详细信息，请参见 『SQL Anywhere Studio 介绍』> 「UltraLite 目标平台」。

有关 UltraLite 数据库功能的详细信息，请参见 『UltraLite 数据库用户指南』> 「UltraLite 数据库」。

系统要求和支持的平台

开发平台

要使用 Native UltraLite for Java 开发应用程序，需要以下各项。

- Microsoft Windows NT/2000/XP。
- 对于 Native UltraLite for Java 应用程序开发，建议使用 JDK 1.1.8 或 Personal Java 1.2。提供 Borland JBuilder 7 和 8 集成。

目标平台

Native UltraLite for Java 支持以下目标平台：

- 使用 ARM 处理器的设备（包括 Compaq iPaq 和 NEC MobilePro P300）上的 Windows CE 3.0 或更高版本，以及与 Jeode Personal Java 1.2 兼容的 VM。
- 具有 JRE 1.1.8 或更高版本的 Windows NT/2000/XP。

有关详细信息，请参见 『SQL Anywhere Studio 介绍』> 「UltraLite 开发平台」和 『SQL Anywhere Studio 介绍』> 「UltraLite 目标平台」。

Native UltraLite for Java 体系结构

Native UltraLite for Java 程序包的名称为 ianywhere.native_ultralite。

下面的列表描述了一些比较常用的高层次类。

- **DatabaseManager** 您要为每个应用程序创建一个 DatabaseManager 对象。
- **连接** 每个 Connection 对象都表示一个指向 UltraLite 数据库的连接。您可以创建一个或多个 Connection 对象。
- **表** Table 对象提供对数据库中数据的访问。
- **PreparedStatement、 ResultSet 和 ResultSetSchema** 使用这些动态 SQL 对象，您可以创建 [动态 SQL] 语句，进行查询和执行 INSERT、 UPDATE 和 DELETE 等语句，以及实现对数据库结果集的程序化控制。
- **SyncParms** 使用 SyncParms 对象可将 UltraLite 数据库与 MobiLink 同步服务器同步。

API 参考以 Javadoc 格式在 SQL Anywhere 安装的子目录 *UltraLite\NativeUltraLiteForJava\doc* 中提供。有关访问 API 参考的详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」。

第 2 章

教程：构建 Native UltraLite for Java 应用程序

关于本章

本章提供的教程可指导您完成构建 Native UltraLite for Java 示例应用程序的过程。

简介

本教程指导您完成构建 Native UltraLite for Java 应用程序的过程。

本教程使用文本编辑器来编辑 Java 文件。您也可以在 Borland JBuilder 开发环境中使用 Native UltraLite for Java。

有关详细信息，请参见使用 Borland JBuilder 开发应用程序。

计时

如果您复制并粘贴代码，此教程需要大约 30 分钟时间。如果您自己输入代码，则会大大延长所需时间。

能力和经验

本教程假定：

- 您熟悉 Java 编程语言
- 您的计算机上已安装 JDK 1.1.8 或 Personal Java 1.2
- 您知道如何使用 UltraLite 模式管理器创建 UltraLite 模式

  有关详细信息，请参见 『UltraLite 数据库用户指南』> 「第 1 课：创建 UltraLite 数据库模式」。

注意
此教程的同步部分需要 SQL Anywhere Studio。

目标

本教程旨在帮助您掌握和熟悉开发 Native UltraLite for Java 应用程序的过程。

第 1 课：连接到数据库

在第一步中，创建一个数据库模式。然后使用创建的模式编写、编译和运行 Java 应用程序，以创建数据库或连接到现有数据库。

❖ 创建数据库模式

1　创建一个目录，用于存放在本教程中创建的文件。

本教程假定该目录为 *c:\tutorial\java*。如果您创建其它名称的目录，请在教程中自始至终使用该目录，而不要使用 *c:\tutorial\java* 目录。

2　使用 UltraLite 模式管理器创建具有以下特征的数据库模式。

有关详细信息，请参见 『UltraLite 数据库用户指南』> 「第 1 课：创建 UltraLite 数据库模式」。

模式文件名：**tutcustomer.usm**

表名：**Customer**

Customer 中的列：

列名	数据类型（大小）	列是否允许 NULL 值？	缺省值
ID	integer	否	自动增量
FName	char(15)	否	无
LName	char(20)	否	无
City	char(20)	是	无
Phone	char(12)	是	555-1234

主键：升序 **ID**

❖ 连接到 UltraLite 数据库

1 在教程目录中，创建一个名为 *Customer.java* 的文件。

2 将下面的代码复制到 *Customer.java* 中。

该代码执行下列任务：

- 导入 UltraLite 库和 JDBC SQLException 类。
- 声明名为 Customer 的类。
- 声明一个静态变量以保存数据库连接对象。本教程后面的几个方法中将用到该对象。
- 调用类构造函数。
- 如果出现错误，则输出错误代码和堆栈跟踪。

  有关错误代码的详细信息，请参见 『ASA 错误消息』> 「ASA 错误消息」。

```
import ianywhere.native_ultralite.*;
 import java.sql.SQLException;

 public class Customer {
   static Connection conn;
   public static void main( String args[] ) {
     try {
       Customer cust = new Customer();
       // Clean up
       conn.close();
     } catch( SQLException e ) {
       System.out.println(
         "Exception: " + e.getMessage() +
         " sqlcode=" + e.getErrorCode()
           );
       e.printStackTrace();
     }
   }
 }
```

3 向该类添加一个构造函数。

将下面的代码复制到您的文件中。此代码执行下列任务。

- 实例化一个新的 DatabaseManager 对象。所有 UltraLite 对象都创建自 DatabaseManager 对象。

- 实例化并定义一个新 CreateParms 对象。 CreateParms 存储连接到或创建数据库所必需的参数。此处的参数为数据库文件和模式文件在桌面上的位置。

  此例中，为了方便起见，位置是硬编码的。但在实际应用程序中可能并非如此。此外，这些参数只对于开发环境中的连接是足够的；但对于要在 Windows CE 设备上运行的应用程序，还需要使用其它参数。这些附加参数将在本教程后面的内容中进行说明。

- 如果该数据库文件不存在，则抛出 SQLException。捕获此异常的代码使用模式文件来创建一个新的数据库并建立指向该数据库的连接。

- 如果数据库文件确实存在，则建立连接。

```
public Customer() throws SQLException
 {
   // Connect
   DatabaseManager dbMgr = new DatabaseManager();
   CreateParms parms = new CreateParms();
   parms.databaseOnDesktop =
"c:\\tutorial\\java\\tutcustomer.udb";
   parms.schema.schemaOnDesktop =
"c:\\tutorial\\java\\tutcustomer.usm";
   try {
     conn = dbMgr.openConnection( parms );
     System.out.println(
            "Connected to an existing database." );
   } catch( SQLException econn ) {
     if( econn.getErrorCode() ==
         SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND
){
       conn = dbMgr.createDatabase( parms );
       System.out.println(
             "Connected to a new database." );
     } else {
       throw econn;
     }
   }
 }
```

4 编译 Customer 类。

建议使用 Sun JDK 1.1.8 或 Personal Java 1.2 来编译该类。此外，必须将 UltraLite 库 *jul9.jar* 添加到类路径。此库位于 SQL Anywhere 安装目录下的 *ultralite\NativeUltraLiteForJava* 子目录中。

在命令提示符下的一行中输入以下命令。

```
javac -classpath

"%ASANY9%\ultralite\NativeUltraLiteForJava\jul9.jar
;%classpath%"
 Customer.java
```

5 运行应用程序。

如同上一步，类路径下必须包含 UltraLite 库 *jul9.jar*。

该应用程序必须还能够装载包含 UltraLite 本机方法的 DLL。此 DLL 名为 *jul9.dll*，位于 SQL Anywhere 9 的安装目录下的 *ultralite\NativeUltraLiteForJava\win32* 子目录中。此 DLL 可位于系统路径中，也可以在 Java 命令行指定它，如下所示：

```
java -classpath

".;%ASANY9%\ultralite\NativeUltraLiteForJava\jul9.j
ar"
 -Djul.library.dir=
 "%ASANY9%\ultralite\NativeUltraLiteForJava\win32"
Customer
```

此命令必须全部包含在一行中，并且在每个参数的中间不能有空格。

第 2 课：向数据库中插入数据

以下过程演示如何向数据库中添加数据。

❖ 向数据库中添加行

1 将下面的方法添加到 *Customer.java* 文件。

此方法执行下列任务：

- 打开表。必须打开 Table 对象才能对表执行操作。若要获取 Table 对象，请使用 **connection.getTable()** 方法。
- 获取表中某些列的标识符。该表中的其它列可以接受 NULL 值或具有缺省值。在本教程中，仅指定了必需的值。
- 如果此表为空，请添加两行。为插入每一行，该代码使用 InsertBegin 将模式设置为插入模式，为每个必需的列设置值，并执行插入操作将这些行添加到该数据库。

  在这里并不严格需要使用提交方法，因为应用程序的缺省模式是在每次插入后提交操作。此处代码中包含该方法旨在强调：如果关闭了自动提交行为，则必须提交更改，以便它成为永久性更改。
- 如果此表非空，则报告表中的行数。
- 关闭 Table 对象。

```
private void insert() throws SQLException
 {
   // Open the Customer table
   Table t = conn.getTable( "Customer" );
   t.open();

   short id = t.schema.getColumnID( "ID" );
   short fname = t.schema.getColumnID( "FName" );
   short lname = t.schema.getColumnID( "LName" );

   // Insert two rows if the table is empty
   if( t.getRowCount() == 0 ) {
```

```
            t.insertBegin();
            t.setString( fname, "Gene" );
            t.setString( lname, "Poole" );
            t.insert();

            t.insertBegin();
            t.setString( fname, "Penny" );
            t.setString( lname, "Stamp" );
            t.insert();

            conn.commit();
            System.out.println( "Two rows inserted." );
        } else {
            System.out.println( "The table has " +
            t.getRowCount() + " rows." );
        }
        t.close();
    }
```

2 调用已创建的 Insert 方法。

将下面一行添加到 `main()` 方法，位置紧跟在调用 Customer 构造函数的语句之后。

```
cust.insert();
```

3 编译并运行应用程序，参见 『Native UltraLite for Java 用户指南』> 「第 1 课：连接到数据库」。

第 3 课：从表中选择行

以下过程演示如何循环遍历表的行。它从表中检索行，并将它们输出到命令行。

❖ 列出表中的行

1　将下面的方法添加到 *Customer.java* 文件。

此方法执行下列任务。

- 打开 Table 对象。
- 检索列标识符。
- 将当前位置设置在表的第一行之前。

  对表的任何操作都在当前位置执行。此位置可以在表的第一行之前、某一行中或最后一行之后。缺省情况下（如在本例中），行是按照其主键值 (id) 进行排序的。要以其它方式对行进行排序，可以向数据库中添加新的索引，并使用该索引打开表。
- 对于每一行，将写出 ID 和名称。循环将持续进行，直到 **moveNext** 返回 False （检索完最后一行后将发生此情况）。
- 关闭 Table 对象。

```
private void select() throws SQLException
 {
   // Fetch rows
   // Open the Customer table
   Table t = conn.getTable( "Customer" );
   t.open();
   short id = t.schema.getColumnID( "ID" );
   short fname = t.schema.getColumnID( "FName" );
   short lname = t.schema.getColumnID( "LName" );
   t.moveBeforeFirst();
   while( t.moveNext() ) {
       System.out.println(
         "id= " + t.getInt( id )
```

```
            + ", name= " + t.getString( fname )
            + " " + t.getString( lname )
            );
      }
      t.close();
   }
```

2 调用已创建的 select 函数。

将下面一行添加到 main() 方法，位置紧跟在调用 insert 方法的语句之后。

```
cust.select();
```

3 编译并运行应用程序，参见 『Native UltraLite for Java 用户指南』> 「第 1 课：连接到数据库」。

第 4 课：将应用程序部署到 Windows CE 设备

在 Windows CE 中运行示例应用程序要求您的设备上安装了受支持的 Java VM。

Jeode Runtime (Java VM) 与一些设备一起打包提供，这些设备包括 HP/Compaq iPAQ 和 NEC MobilePro P300 等。同时提供的还有 CrEme Plus，它包括与 Symbol 增值功能捆绑的 CrEme。这两种 VM 都可购买以供其它设备使用。

❖ 确认在 Windows CE 设备中安装了 Jeode VM：

- 选择 [开始] → [程序]。

  如果已安装 Jeode VM，则 Programs 文件夹中会显示 Jeode 文件夹。

❖ 准备在 Windows CE 设备上运行应用程序：

1. 通过指定 schemaOnCE 和 databaseOnCE 参数指定数据库文件和模式文件的位置。

   在 Customer 构造函数中，将 CreateParms 对象更改为如下内容。

   ```
   CreateParms parms = new CreateParms();
    parms.databaseOnDesktop =
   "c:\\tutorial\\java\\tutcustomer.udb";
    parms.databaseOnCE =
   "\\UltraLite\\tutorial\\tutcustomer.udb";
    parms.schema.schemaOnDesktop =
   "c:\\tutorial\\java\\tutcustomer.usm";
    parms.schema.schemaOnCE =
   "\\UltraLite\\tutorial\\tutcustomer.usm"";
   ```

2. 编译应用程序，参见 『Native UltraLite for Java 用户指南』> 「第 1 课：构建 CustDB 应用程序」。

3. 运行应用程序，检查是否引入了错误。在桌面环境中运行时，schemaOnCE 和 databaseOnCE 参数不起作用。

4 准备运行应用程序的快捷方式。

快捷方式是带有 *.lnk* 扩展名的文本文件，它包含一个运行该应用程序的命令行。下一步，将此快捷方式文件复制到设备上的某个位置。

使用文本编辑器，在您的教程目录中创建一个名为 *tutorial.lnk* 的文件，该文件包含以下内容，且这些内容应位于同一行。

```
18#"\Windows\evm.exe"
 -Djeode.evm.console.local.keep=TRUE
 -Djeode.evm.console.local.paging=TRUE
 -Djul.library.dir=\UltraLite\lib
 -cp \UltraLite\tutorial;\UltraLite\lib\jul9.jar
 Customer
```

此命令中的元素的含义如下。

- 第一行启动 Jeode VM 可执行程序。
- `-Djeode` 选项控制用于应用程序输出的文本控制台的显示方式。
- `-Djul.library.dir` 选项将 VM 指向 UltraLite 本机接口运行库 (*jul9.dll*)。
- `-cp` 选项提供 VM 的类路径。它指示应用程序和 UltraLite 运行库的位置。
- 最后一个参数是类，在本例中为 `Customer`。

以下过程将文件复制到您的设备中。您必须复制 UltraLite 运行库文件和应用程序特定文件。

❖ 将 UltraLite 运行库部署到 Windows CE 设备：

1 在 Windows CE 设备上启动 Windows 资源管理器。

- 确保设备已经连接到桌上型计算机。
- 在 ActiveSync 窗口中，单击 [资源管理器]。随即打开 [资源管理器] 窗口。

2　创建目录以保存 UltraLite 运行库和应用程序。

- 在资源管理器窗口中，单击 [我的 Pocket PC] 以访问根目录。
- 创建一个名为 *UltraLite* 的目录。
- 打开此 UltraLite 目录并创建名为 *lib* 和 *tutorial* 的子目录。目录路径 *\UltraLite\lib* 是 UltraLite 运行库文件的位置，路径 *\UltraLite\tutorial* 是应用程序的位置。这些目录与上述快捷方式文件中的选项相匹配。

3　将 UltraLite 运行时库文件复制到 Windows CE 设备。

- 另外启动一个 [资源管理器] 窗口，并浏览到您的桌上型计算机上的 SQL Anywhere 安装目录。
- 将下面的文件从桌面拖至该设备

相对于 SQL Anywhere 目录的桌面位置	Windows CE 位置
UltraLite\NativeUltraLiteForJava\jul9.jar	\UltraLite\lib\jul9.jar
UltraLite\NativeUltraLiteForJava \ce\arm\jul9.dll	\UltraLite\lib\jul9.dll

或者，您可以部署 UltraLite 引擎，它是允许对 UltraLite 数据库进行并发多进程访问的可执行文件。如果使用 UltraLite 引擎，您必须部署适当版本的组件 (*julclient9.dll*)，且必须在数据库管理器中指定 [运行时类型] 属性。

4　将应用程序文件复制到 Windows CE 设备。

- 在 [资源管理器] 窗口中，浏览到教程目录。
- 将下面的文件从桌面拖至该设备

桌面位置	Windows CE 位置
Customer.class	\UltraLite\tutorial

桌面位置	Windows CE 位置
tutcustomer.usm	\UltraLite\tutorial

在本教程中，不要将数据库文件复制到 Windows CE 设备。您应将模式文件部署到该设备，应用程序将基于该模式文件创建一个数据库。

5 运行 CrEme VM 时，如果希望使用 ActiveSync，请将 *\Windows\CrEme\lib\crmex.jar* 添加到类路径中。

6 将快捷方式文件复制到 Windows CE 设备。

- 将下面的文件从桌面拖至该设备。

桌面位置	Windows CE 位置
tutorial.lnk	\Windows\Start Menu

以下过程将运行应用程序

❖ 运行应用程序：

1 在 Windows CE 设备上，选择 [开始] →tutorial。

此快捷方式是您复制到该设备上的 *tutorial.lnk* 文件。

- Jeode VM 被装载并显示控制台。
- 控制台输出如下消息。

```
Connected to a new database.
 Two rows inserted.
 id= 1, name= Gene Poole
 id= 2, name= Penny Stamp
 Application finished. Please close console
```

- 关闭控制台。

2 在 Windows CE 设备上，再次选择 [开始] →tutorial。

这一次前两个消息显示如下。

```
Connected to an existing database.
 The table has 2 rows.
```

3　关闭控制台。

第 5 课：向应用程序添加同步

以下过程将同步代码添加到应用程序中，启动 MobiLink 同步服务器，并运行要同步的应用程序。

注意
本课使用 SQL Anywhere Studio 包含的 MobiLink 同步。要完成本课，您的计算机上必须安装了 SQL Anywhere Studio。

同步是通过 ASA 9.0 示例数据库完成的。 ASA 9.0 示例数据库中有一个 Customer 表，该表的列与您的 UltraLite 数据库内 customer 表中的列匹配。同步期间，该表中的数据下载到您的 UltraLite 应用程序。

本课假定您对 MobiLink 同步有一定了解。

❖ 向应用程序添加同步

1 将下面的方法添加到 *Customer.java* 中。

此代码执行下列任务。

- 将同步流设置为 TCP/IP。同步还可以通过 HTTP、ActiveSync 或 HTTPS 执行。 HTTPS 同步要求您获得单独许可的 SQL Anywhere Studio 安全性选件。

  有关详细信息，请参见 **ianywhere.native_ultralite.StreamType** 和 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Connection**。

  Connection 对象的 syncParms 字段提供对 SyncParms 对象的方便访问。有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.SyncParms**。

- MobiLink 同步是由 MobiLink 同步服务器上的脚本控制的。脚本版本确定要使用的脚本集。setSendColumnNames 方法与 MobiLink 同步服务器上的一个选项一起使用可以自动生成脚本。

  有关详细信息，请参见 『MobiLink 管理指南』> 「自动生成脚本」。

- 设置 MobiLink 用户名。此值用于 MobiLink 同步服务器上的鉴定，它与 UltraLite 数据库用户 ID 不同，不过在某些应用程序中您可能希望使它们相同。

- 将同步设置为仅下载数据。缺省情况下， MobiLink 同步是双向的。在这里，我们使用仅下载同步，以便您的表中的行不会上载到示例数据库中。

```
private void sync() throws SQLException {
   conn.syncParms.setStream( StreamType.TCPIP );
   conn.syncParms.setVersion( "ul_default" );
   conn.syncParms.setUserName( "sample" );
   conn.syncParms.setSendColumnNames( true );
   conn.syncParms.setDownloadOnly( true );
   conn.synchronize();
 }
```

2 调用 sync 方法。

将下面的行添加到 main() 方法中，放在调用 insert 方法的语句和调用 select 方法的语句之间：

```
cust.sync();
```

3 编译应用程序，参见 『Native UltraLite for Java 用户指南』> 「第 1 课：连接到数据库」。但暂时不要运行它。

❖ 同步数据

1 启动 MobiLink 同步服务器。

在命令行提示符下，运行以下 MobiLink 命令行。

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -v+ -zu+ -za
```

ASA 9.0 示例数据库包含一个 Customer 表，该表与您已经创建的 UltraLite 数据库中的列匹配。您可以将您的 UltraLite 应用程序与 ASA 9.0 示例数据库同步。

`-zu+` 和 `-za` 命令行选项提供自动添加用户和生成同步脚本的功能。

有关这些选项的详细信息，请参见 『MobiLink 管理指南』> 「MobiLink 同步服务器选项」。

2 运行应用程序，参见 「第 1 课：连接到数据库」第 7 页。

MobiLink 同步服务器窗口显示表明同步进度的状态消息。最后一条消息显示 [同步已完成]。

小结

学到的知识

在本教程中，您学习了以下内容：

- 创建一个数据库模式
- 创建一个 Native UltraLite for Java 应用程序
- 将一个 UltraLite 远程数据库与一个 Adaptive Server Anywhere 统一数据库同步
- 掌握开发 Native UltraLite for Java 应用程序的过程

示例

- 有关更多代码示例，请参见 *Samples\NativeUltraLiteForJava\Simple\Simple.java*。

第 3 章

教程：CustDB 示例应用程序

关于本章

本章提供的教程可指导您使用 Native UltraLite for Java 构建和部署多表应用程序。

此教程还演示了 UltraLite 远程数据库和统一数据库之间的同步。此教程的同步部分需要 SQL Anywhere Studio。

简介

本教程可指导您编译、运行和部署 CustDB 示例应用程序。它演示了使用 Native UltraLite for Java 执行多种常见任务的方法。

SQL Anywhere 安装目录的 *Samples\NativeUltraLiteForJava\CustDB* 子目录中提供了 CustDB 的源代码。

计时

如果您复制并粘贴代码，此教程需要大约 30 分钟时间。如果您自己输入代码，则会大大延长所需时间。

能力和经验

本教程假定：

- 您可以访问运行 Jeode VM 的 Windows CE 设备
- 您的计算机上已安装 Java 开发工具包。建议使用 JDK 1.1.8 或 Personal Java 1.2，因为它们与用于部署的 Jeode VM 兼容。
- 您知道如何使用 UltraLite 模式管理器创建 UltraLite 模式

  有关详细信息，请参见 『UltraLite 数据库用户指南』> 「第 1 课：创建 UltraLite 数据库模式」。

注意
此教程的同步部分需要 SQL Anywhere Studio。

目标

本教程旨在帮助您掌握和熟悉 Native UltraLite for Java 应用程序的开发。

第 1 课：构建 CustDB 应用程序

以下过程使用 *build.bat* 脚本 （位于安装目录下的 *Samples\NativeUltraLiteForJava\CustDB* 子目录）从源 Java 文件构建 CustDB 示例。

同时提供的 *clean.bat* 脚本用于删除该构建行为的结果。

❖ 构建示例应用程序：

1. 打开命令提示窗口，浏览到 SQL Anywhere 安装目录的 *Samples\NativeUltraLiteForJava\CustDB* 子目录。

   在整个教程中，您应当保持该命令提示符处于打开状态。

2. 设置所需的环境变量。

   确保正确地定义了环境变量 ASANY9 和 JAVA_HOME。

   - 安装程序将 ASANY9 设置为 SQL Anywhere 安装位置。
   - 将 JAVA_HOME 环境变量设置为计算机上 JDK 的位置。建议使用 JDK 1.1.8 或 Personal Java 1.2，因为它与用于部署的 Jeode VM 兼容。

     例如，如果 JDK 为版本 1.1.8，路径为 *c:\jdk1.1.8*，则运行下面的命令。

     ```
     set JAVA_HOME=c:\jdk1.1.8
     ```

3. 构建应用程序。

   在命令提示符下，运行以下命令：

   ```
   build
   ```

   build 批处理文件将执行以下操作。

   - 编译 CustDB 示例代码。

     已编译的类存储在 *builddir* 子目录中。

- 将已编译的类放入一个 JAR 文件中。

  JAR 文件存储在 *CustDB* 目录中。

- 使用 *ulinit* 命令行实用程序创建数据库模式文件。

  模式文件将从 Adaptive Server Anywhere UltraLite 9.0 示例数据库创建。

  有关 ulinit 工具的详细信息，请参见 『UltraLite 数据库用户指南』> 「ulinit 实用程序」。

示例代码

下面给出了示例代码文件及简要说明。

文件	说明
Application.java	主用户界面框架、事件处理和 ActiveSync 支持。
CustDB.java	数据库的主界面。大多数数据库处理都在此文件中进行。
DialogDelOrder.java	删除确认对话框。
DialogNewOrder.java	新订单输入表单，显示用数据库中的数据填充列表框。
Dialogs.java	对话的公用基类。
DialogSyncHost.java	它提示输入同步主机名。
DialogUserID.java	它提示输入雇员 ID。
GetOrder.java	表示订单数据的类。它显示如何进行键连接。
GetOrderData.java	它用于计算最小和最大订单 ID。等价于 SELECT max(order_id), min(order_id) FROM ULOrder。

第 2 课：运行 CustDB 应用程序

下面的步骤演示如何在 Windows 中运行 CustDB 示例。「第 3 课：将 CustDB 部署到 Windows CE 设备」第30页中介绍了向 Windows CE 设备的部署。本课使用 MobiLink 同步，所以需要 SQL Anywhere Studio。

❖ 在 Windows 中运行该示例应用程序：

1. 在与上一课相同的命令行提示符处，键入 `win32\run.bat`。

   此命令执行以下操作。

   - 启动 MobiLink 同步服务器。

     该服务器连接到 UltraLite 9.0 示例数据库 （该数据库用作 CustDB 应用程序的统一数据库）。

   - 启动 CustDB 示例应用程序。

     第一次运行时，该应用程序在启动时不带有 UltraLite 数据库 （*.udb* 文件），因此它从 UltraLite 模式创建此文件。

   - 显示登录窗口。

     该窗口位于屏幕中央，但可能在其它窗口的后面。可能必须移动其它窗口才能找到该登录窗口。

2. 登录到该应用程序。

   单击 [OK] 以使用 MobiLink 用户名 **50** 的用户身份登录。该应用程序进行同步操作，并显示同步进度信息。片刻之后，将显示包含单个订单的窗口。

3. 使用此应用程序。

   您可以浏览数据库中的各行，批准和拒绝订单以及进行同步操作。退出此应用程序后，批处理文件将关闭 MobiLink 同步服务器。

第 3 课：将 CustDB 部署到 Windows CE 设备

以下过程将把您的应用程序部署到 Windows CE 设备。部署需要 Jeode Runtime (Java VM)。

❖ 将应用程序部署到 Windows CE 设备：

1. 在 Windows CE 设备上启动 Windows 资源管理器。
   - 确保设备已经连接到桌上型计算机。
   - 在 ActiveSync 窗口中，单击 [资源管理器]。

     随即打开 [资源管理器] 窗口。
2. 创建目录以保存 UltraLite 运行库和应用程序。

   如果您已经完成前面的教程，这些目录中的一部分可能已经存在。
   - 在 [资源管理器] 窗口中，单击 [我的 Pocket PC]。这是根目录，其路径为 \。
   - 创建一个名为 *UltraLite* 的目录。
   - 打开 UltraLite 目录并创建名为 *lib* 和 *CustDB* 的子目录。

     \UltraLite\lib 是 UltraLite 运行库文件的位置，*\UltraLite\CustDB* 是应用程序的位置。这些目录与 *ce* 子目录中提供的快捷方式文件中的选项匹配。
3. 将 UltraLite 运行时库文件复制到 Windows CE 设备。

   如果已经完成前面的教程，那么您可能已经执行了此操作。您不需要重复此步骤。
   - 另外启动一个资源管理器窗口，并浏览到您的桌上型计算机上的 SQL Anywhere 安装目录。
   - 将下面的文件从桌面拖至该设备。

相对于 SQL Anywhere 目录的桌面位置	Windows CE 位置
UltraLite\NativeUltraLiteForJava\jul9.jar	*\UltraLite\lib*
UltraLite\NativeUltraLiteForJava\ce\arm\jul9.dll	*\UltraLite\lib*

4　将应用程序文件复制到 Windows CE 设备。

- 在 [资源管理器] 窗口中，浏览至 *Samples\NativeUltraLiteForJava\CustDB* 目录。
- 将下面的文件从桌面拖至该设备。

桌面位置	Windows CE 位置
custdb.jar	*\UltraLite\CustDB*
ul_custapi.usm	*\UltraLite\CustDB*

在本教程中，不要将数据库文件复制到 Windows CE 设备。您应将模式文件部署到该设备，应用程序将创建一个数据库。

5　将快捷方式文件复制到 Windows CE 设备。

- 将下面的文件从桌面拖至该设备。

相对于 SQL Anywhere 目录的桌面位置	Windows CE 位置
ce\CustDB.lnk	*\Windows\Start Menu*

6　安装 ActiveSync 提供程序。

ActiveSync 提供程序在进行同步时是必需的。

有关说明，请参见 『MobiLink 客户端』> 「ActiveSync 提供程序的安装实用程序」。

7　注册要和 ActiveSync 一起使用的应用程序。

注册 CustDB 时，请使用下面的属性。

属性	值
名称	JULCustDB
类名	JULCustDB
文件位置	*\Windows\evm.exe*
参数	-Djeode.evm.console.local.keep=TRUE -Djul.library.dir=\UltraLite\lib -cp \UltraLite\CustDB\custdb.jar;\UltraLite\lib\jul9.jar custdb.Application ACTIVE_SYNC_LAUNCH

您可以运行批处理文件 *Samples\NativeUltraLiteForJava\CustDB\ce\as_register.bat* 来执行此操作。

SQL Anywhere Studio 用户
有关 ActiveSync 同步的安装步骤，请参见 『MobiLink 客户端』> 「ActiveSync 提供程序的安装实用程序」。

❖ 运行应用程序：

1 启动 MobiLink 同步服务器。

运行 *startdb.bat* 批处理文件 （该文件位于 SQL Anywhere 安装目录下的 *Samples\NativeUltraLiteForJava\CustDB\ce* 子目录中）。

2 在 CE 设备上，从 [开始] 菜单运行 *CustDB* 应用程序。

小结

在本教程中，您学习了以下内容：

- 在您的桌上型计算机中构建并运行了 CustDB 示例应用程序。
- 将 Native UltraLite for Java 应用程序部署到了 Windows CE 设备。

第 4 章

了解 Native UltraLite for Java 开发

关于本章

本章讲解如何使用 Native UltraLite for Java 开发应用程序。

有关实践教程，请参见 『Native UltraLite for Java 用户指南』> 「教程：构建 Native UltraLite for Java 应用程序」或 『Native UltraLite for Java 用户指南』> 「教程：CustDB 示例应用程序」。

UltraLite 数据库模式

数据库模式是对数据库的描述。它是数据库中的表、索引、键和发布，以及它们之间的所有关系的集合。

您不要直接更改 UltraLite 数据库的模式，而应创建一个模式 (*.usm*) 文件，并调用应用程序中的一个内置 UltraLite 函数通过该文件升级数据库模式。

最初创建数据库时，也要使用一个模式文件来指定数据库的结构。

创建 UltraLite 数据库模式文件

可以使用 UltraLite 模式管理器或 *ulinit* 实用程序来创建 UltraLite 模式文件。

- **UltraLite 模式管理器** UltraLite 模式管理器是一个图形实用程序，用于创建和编辑 UltraLite 模式文件。

  若要启动模式管理器，请选择 [开始] → [程序] → [SQL Anywhere 9] → [UltraLite] → [UltraLite 模式管理器]，或在 Windows 资源管理器中双击某个模式 (*.usm*) 文件。

  有关使用 UltraLite 模式管理器的详细信息，请参见 『UltraLite 数据库用户指南』> 「第 1 课：创建 UltraLite 数据库模式」。

- **ulinit 实用程序** 如果您拥有 Adaptive Server Anywhere 数据库管理系统，则可以使用 *ulinit* 命令行实用程序来生成 UltraLite 模式文件。

  有关使用 *ulinit* 实用程序的详细信息，请参见 『UltraLite 数据库用户指南』> 「ulinit 实用程序」。

升级数据库模式

要修改现有数据库结构，请使用 DatabaseSchema.applyFile 方法。在大多数情况下将不会发生数据丢失，但如果列被删除，或者如果列的数据类型被更改为不兼容的类型，则会发生数据丢失的情况。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 DatabaseSchema.ApplyFile。

示例

以下代码使用 ApplyFile 方法应用新模式文件。它假定您已经按照 「连接到数据库」第 38 页中的说明连接到了数据库。

```
DatabaseSchema schema = conn.schema;
 SchemaParms parms = new SchemaParms();
 parms.schemaOnDesktop =
"c:\tutorial\tutcustomer.usm";
 try {
   schema.applyFile(parms );
 } catch {
   // handle any errors
 }
```

连接到数据库

UltraLite 应用程序必须先连接到数据库，然后才能对数据库中的数据进行操作。本节将介绍如何连接到 UltraLite 数据库。

使用 Connection 对象

Connection 对象的以下属性控制全局应用程序行为。

- **提交行为** 缺省情况下， Native UltraLite for Java 应用程序处于 [自动提交] 模式。每个插入、更新或删除语句都被立即提交给数据库。您也可以将 Connection.AutoCommit 设置为 False，以将事务构建到应用程序中。

  有关详细信息，请参见 「UltraLite 中的事务处理」第 54 页。

- **用户鉴定** 可以使用 [授予] 和 [撤消] 连接权限的方法将应用程序的用户 ID 和口令从其缺省值 DBA 和 SQL 更改为其它值。每个应用程序最多可以有四个用户 ID。

  有关详细信息，请参见 「用户鉴定」第 57 页。

- **同步** 可以通过 Connection 对象访问控制同步的一组对象。

  有关详细信息，请参见 「同步 UltraLite 应用程序」第 58 页。

- **表** 使用 Connection 对象的方法可以访问 UltraLite 表。

  有关详细信息，请参见 「使用 Table API 访问和操作数据」第 47 页。

- **预准备语句** 系统提供了一组对象来处理动态 SQL 语句的执行和浏览结果集。

  有关详细信息，请参见 「使用动态 SQL 访问和操作数据」第 42 页。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Connection**

多线程应用程序

每个 Connection 对象以及从该对象中创建的所有对象都应该在单个线程中使用。如果您的应用程序需要使用多个线程访问 UltraLite 数据库，则每个线程都需要一个单独的连接。

❖ 连接到 UltraLite 数据库

1　创建并初始化 DatabaseManager 对象。

DatabaseManager 对象位于对象层次结构的根部。对于每个应用程序，只应创建一个 DatabaseManager 对象。通常，最好将 DatabaseManager 对象声明为应用程序范围内的全局对象。

```
DatabaseManager dbMgr = new DatabaseManager();
```

有关详细信息，请参见 『Native UltraLite for Java 用户指南』>「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.DatabaseManager**。

2　声明 Connection 对象。

大多数应用程序都使用一个与 UltraLite 数据库的连接，并使该连接保持打开状态。只有多线程数据访问才需要多个连接。因此，通常最好将 Connection 对象声明为应用程序范围内的全局对象。

```
static Connection conn;
```

3　打开与现有数据库的连接，或者，如果指定的数据库文件不存在，创建一个新数据库。

大多数 UltraLite 应用程序都部署模式文件而不是数据库文件，并让 UltraLite 在第一次尝试连接时创建数据库文件。因此，以下代码尝试连接到现有数据库。如果数据库文件不存在，应用程序将创建数据库文件。

```
ConnectionParms parms = new ConnectionParms();
 // specify the location of the database file
 parms.databaseOnDesktop = "mydata.udb";
 try {
   conn = dbMgr.openConnection( parms );
 } catch( SQLException econn ) {
   if( econn.getErrorCode() ==
```

```
            SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND ){
            CreateParms parms = new CreateParms();
            parms.databaseOnDesktop = "mydata.udb";
            parms.schema.schemaOnDesktop = "mydata.usm";
            try {
              conn = dbMgr.createDatabase( parms );
          }
        }
```

示例

以下代码用于打开到一个名为 *mydata.udb* 的 UltraLite 数据库的连接。如果该数据库不存在，则使用名为 *mydata.usm* 的 UltraLite 模式文件创建该数据库。

有关详细信息，请参见 *Samples\NativeUltraLiteForJava\Simple\Simple.java* 中的代码。

```
CreateParms parms = new CreateParms();
 parms.databaseOnDesktop = "mydata.udb";
 parms.schema.schemaOnDesktop = "mydata.usm";
 try {
   conn = dbMgr.openConnection( parms );
   System.out.println(
     "Connected to an existing database." );
 }
 catch( SQLException econn ) {
   if( econn.getErrorCode() ==
       SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND ){
     conn = dbMgr.createDatabase( parms );
     System.out.println(
     "Connected to a new database." );
   } else {
     throw econn;
   }
 }
```

加密和模糊处理

您可以使用 Native UltraLite for Java 为 UltraLite 数据库加密或进行模糊处理。

加密

要创建加密的数据库，请使用 **ianywhere.native_ultralite.ConnectionParms.EncryptionKey** 属性指定一个加密密钥，然后调用 **createDatabase** 方法。调用 **createDatabase** 方法时，就会创建数据库并会使用指定密钥为其加密。

有关详细信息，请参见 『UltraLite 数据库用户指南』> 「Encryption Key 连接参数 」。

通过使用 **ianywhere.native_ultralite.Connection.changeEncryptionKey** 方法指定新加密密钥，可以更改加密密钥。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Connection**。

对数据库进行了加密后，与数据库的连接必须指定正确的加密密钥。否则，连接将失败。

模糊处理

若要对数据库进行模糊处理，请指定 obfuscate=1 作为一个创建参数。

有关数据库加密的详细信息，请参见 『UltraLite 数据库用户指南』> 「加密 UltraLite 数据库」。

使用动态 SQL 访问和操作数据

UltraLite 应用程序可以使用动态 SQL 或 Table API 访问表数据。本节介绍如何使用动态 SQL 访问数据。

有关使用 Table API 的信息，请参见「使用 Table API 访问和操作数据」第 47 页。

本节讲解如何使用动态 SQL 执行以下任务。

- 插入、删除和更新行。
- 将行检索到一个结果集。
- 滚动浏览结果集中的行。

本节不介绍 SQL 语言本身。有关动态 SQL 功能的信息，请参见『UltraLite 数据库用户指南』> 「动态 SQL」。

有关任何 SQL 操作都需要的一系列操作的概述，请参见 『UltraLite 数据库用户指南』> 「使用动态 SQL」。

数据操作：INSERT、 UPDATE 和 DELETE

使用 UltraLite 可以执行 SQL 数据操作语言操作。这些操作是使用 PreparedStatement.executeStatement 方法执行的。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.PreparedStatement**。

在 预准备语句中使用参数

在 SQL 语句中，参数的占位符由 ? 字符表示。对于任何 INSERT、UPDATE 或 DELETE 语句，每个 ? 都是根据其在预准备语句中的序号位置引用的。例如，第一个 ? 引用为 1，第二个引用为 2。

❖ 插入一行

1　声明 PreparedStatement。

```
PreparedStatement prepStmt;
```

2　将一条 SQL 语句指派给 PreparedStatement 对象。

```
prepStmt = conn.prepareStatement(
    "INSERT INTO MyTable(MyColumn) values (?)");
```

3　为该语句指派输入参数值。

以下代码显示一个字符串参数。

```
String newValue;
 // assign value
 prepStmt.setStringParameter(1, newValue);
```

4　执行该语句。

返回值表示受该语句影响的行数。

```
long rowsInserted = prepStmt.executeStatement();
```

5　如果禁用了 [自动提交]，请提交更改。

```
conn.commit();
```

❖ 更新一行

1　声明 PreparedStatement。

```
PreparedStatement prepStmt;
```

2　将一条语句指派给 PreparedStatement 对象。

```
prepStmt = conn.prepareStatement(
    "UPDATE MyTable SET MyColumn1 = ? WHERE
MyColumn2 = ?");
```

3　为该语句指派输入参数值。

```
String newValue;
 String oldValue;
 // assign values
 prepStmt.setStringParameter( 1, newValue );
 prepStmt.setStringParameter( 2, oldValue );
```

4 执行该语句。

```
long rowsUpdated = prepStmt.executeStatement();
```

5 如果禁用了 [自动提交]，请提交更改。

```
conn.commit();
```

❖ **删除一行**

1 声明 PreparedStatement。

```
PreparedStatement prepStmt;
```

2 将一条语句指派给 PreparedStatement 对象。

```
prepStmt = conn.prepareStatement(
    "DELETE FROM MyTable WHERE MyColumn = ?");
```

3 为该语句指派输入参数值。

```
String deleteValue;
 prepStmt.setStringParameter(1, deleteValue);
```

4 执行该语句。

```
long rowsDeleted = prepStmt.executeStatement();
```

5 如果禁用了 [自动提交]，请提交更改。

```
conn.commit();
```

数据检索：SELECT

使用 SELECT 语句可从数据库中检索信息。本节介绍如何执行 SELECT 语句，以及如何处理该语句返回的结果集。

❖ 执行 SELECT 语句

1　声明一个 PreparedStatement 对象，用于保存查询。

```
PreparedStatement prepStmt;
```

2　将一条语句指派给该对象。

```
prepStmt = conn.prepareStatement(
 "SELECT MyColumn FROM MyTable");
```

3　执行该语句。

在以下代码中， SELECT 查询的结果包含一个字符串，该字符串输出到命令提示符。

```
ResultSet customerNames = prepStmt.executeQuery();
 customerNames.moveBeforeFirst();
 while( customerNames.moveNext() ) {
    for ( int i = 1;
          i <=
customerNames.schema.getColumnCount();
          i++ ) {
       System.out.print(
         customerNames.getString( i ) + " "
         );

   }
   System.out.println();
 }
```

浏览动态 SQL 结果集

可以使用与 ResultSet 对象关联的方法浏览结果集。

结果集对象提供了以下方法用于浏览结果集。

- **moveAfterLast()** 移至最后一行之后的位置。
- **moveBeforeFirst()** 移至第一行之前的位置。
- **moveFirst()** 移至第一行。
- **moveLast()** 移至最后一行。

- **moveNext()** 移至下一行。
- **movePrevious()** 移至上一行。
- **moveRelative(offset)** 根据指定的偏移值，相对于当前行移动一定行数。如果偏移值为正，则相对于游标在结果集中的当前位置在结果集中向前移动，如果偏移值为负，则在结果集中向后移动。如果偏移值为零，则不移动游标，但可以重新填充行缓冲区。

结果集模式说明

使用 ResultSet.schema 属性，可以检索有关结果集的信息，例如列名、总列数、列小数位数、列大小和列 SQL 类型。

示例

下面的示例演示如何使用 ResultSet.schema 属性在主控台窗口中显示模式信息。

```
for ( int i = 1;
       i <= MyResultSet.schema.getColumnCount();
       i++ ) {
    System.out.println(
       MyResultSet.schema.getColumnName(i) + " " +
       MyResultSet.schema.getColumnSQLType(i)
    );
 }
```

使用 Table API 访问和操作数据

UltraLite 应用程序可以使用动态 SQL 或 Table API 访问表数据。本节介绍如何使用 Table API 访问数据。

有关动态 SQL 的信息，请参见「使用动态 SQL 访问和操作数据」第 42 页。

本节讲解如何使用 Table API 执行以下任务。

- 滚动浏览表中的行。
- 访问当前行中的值。
- 使用 Find 和 Lookup 方法来查找表中的行。
- 插入、删除和更新行。

浏览表中的行

Native UltraLite for Java 为您执行大量浏览任务提供了多种浏览表的方法。

Table 对象提供了以下方法来浏览表。

- **moveAfterLast()** 移至最后一行之后的位置。
- **moveBeforeFirst()** 移至第一行之前的位置。
- **moveFirst()** 移至第一行。
- **moveLast()** 移至最后一行。
- **moveNext()** 移至下一行。
- **movePrevious()** 移至上一行。
- **moveRelative(offset)** 根据指定的偏移值，相对于当前行移动一定行数。如果偏移值为正，则相对于游标在表中的当前位置在表中向前移动，如果偏移值为负，则在表中向后移动。如果偏移值为零，则不移动游标，但可以重新填充行缓冲区。

示例 以下代码打开 MyTable 表并显示每一行的 MyColumn 列的值。

```
Table t = conn.getTable( "MyTable" );
 short colID = t.schema.getColumnID( "MyColumn" );
 t.open();
 t.moveBeforeFirst();
 while ( t.moveNext() ){
    System.out.println( t.getString( colID ) );
 }
```

打开表对象时，应用程序可以访问表中的行。缺省情况下，这些行按主键值排序，但可在打开表时指定一个索引，以便以特定的顺序访问行。

示例 以下代码移动到按 ix_col 索引排序的 MyTable 表的第一行。

```
Table t = conn.getTable( "MyTable" );
 t.open( "ix_col" );
 t.moveFirst();
```

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Table** 和 **ianywhere.native_ultralite.TableSchema**。

使用 UltraLite 模式

UltraLite 模式确定将把缓冲区中的值用于何种用途。除缺省模式外，UltraLite 还具有以下四种操作模式。

- **插入模式** 调用 insert 方法时，将缓冲区中的数据作为新行添加到表中。
- **更新模式** 调用 update 方法时，缓冲区中的数据将替换当前行。
- **查找模式** 在调用某一种查找方法时，用于定位其值与缓冲区中的数据完全匹配的行。
- **查寻模式** 调用 lookup 方法之一时，用于定位其值与缓冲区中的数据匹配，或大于缓冲区中的数据的行。

访问当前行中的值

Table 对象始终位于以下位置之一。

- 在表的第一行之前。
- 在表的某一行上。
- 在表的最后一行之后。

如果 Table 对象位于某一行上，可以在适合该数据类型的一组方法中选择一种方法来检索或修改各列的值。

检索列值

Table 对象提供了一组用于检索列值的方法。这些方法都将列 ID 作为参数。

示例

以下代码检索 lname 列的值，该值是字符串。

```
short lname = t.schema.getColumnID( "lname" );
 String lastname = t.getString( lname );
```

以下代码检索 cust_id 列的值，该值是一个整数。

```
short cust_id = t.schema.getColumnID( "cust_id" );
 int id = t.getInt( cust_id );
```

修改列值

除用于检索值的方法外，还有用于设置值的方法。这些方法将列 ID 和值作为参数。

示例

例如，以下代码将 lname 列的值设置为 Kaminski。

```
t.setString( lname, "Kaminski" );
```

通过给这些属性指派值，您不用变更数据库中数据的值。即使是在表的第一行之前或最后一行之后，您也可以为属性指派值。但是，如果当前行位于这两个位置之一，那么尝试访问数据 （比如，将属性指派给变量）就会出现错误。

```
// This code is incorrect
 tCustomer.moveBeforeFirst();
 id = tCustomer.getInt( cust_id );
```

转换值 所选方法必须与要指派的数据类型匹配。 UltraLite 自动转换兼容的数据库数据类型，这样您就能使用 getString 方法将一个整数值读取到字符串变量中，依此类推。

使用 find 和 lookup 来搜索行

UltraLite 具有几种用于处理数据的操作模式。其中， find 和 lookup 两种模式用于搜索。 Table 对象具有相应于这些模式的方法，用于定位表中的特定行。

注意
使用 Find 方法和 Lookup 方法搜索的列必须在用于打开该表的索引中。

- **Find 方法** 按照打开 Table 对象时指定的排序顺序，移动到与指定的搜索值完全匹配的第一行。如果找不到搜索值，则应用程序将定位到第一行之前或最后一行之后。
- **Lookup 方法** 按照打开 Table 对象时指定的排序顺序，移动到与指定的搜索值匹配或大于指定的搜索值的第一行。

❖ 搜索行

1 进入 find 或 lookup 模式。

通过在表对象上调用一个方法可进入该模式。例如，以下代码将进入 find 模式。

```
t.findBegin();
```

2 设置搜索值。

可以通过设置当前行中的值来完成设置。设置这些值仅影响保存当前行的缓冲区，而会不影响数据库。例如，以下代码将缓冲区中的值设置为 Kaminski。

```
short lname = t.schema.getColumnID( "lname" );
 t.setString( lname, "Kaminski" );
```

3 搜索行。

使用正确的方法来执行搜索。例如，以下指令在当前索引中查找与指定值完全匹配的第一行。

对于多列索引，将始终使用第一列的值，但可以忽略其它列。

```
tCustomer.findFirst();
```

4　搜索该行的下一个实例。

使用正确的方法来执行搜索。对于查找操作， findNext() 在索引中查找参数的下一个实例。对于查寻， moveNext() 会查找下一下实例。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Table**。

更新行

以下过程更新一行。

❖ 更新一行

1　移动到想要更新的行。

可以通过滚动浏览表或使用 find 和 lookup 方法在表中进行搜索来移动到某一行。

2　进入更新模式。

例如，以下指令在 t 上进入 update 模式。

```
t.beginUpdate();
```

3　为要更新的行设置新值。例如，以下指令将缓冲区中的 id 列设置为 3。

```
t.setInt( id , 3);
```

4　执行更新。

```
t.update();
```

完成更新操作后，当前行就是已更新的行。如果更改了在打开 Table 对象时指定的索引中的列的值，则当前行不确定。

缺省情况下， Native UltraLite for Java 在 [自动提交] 模式下运行，所以更新会立即应用到永久存储中的行中。如果已经禁用了 [自动提交] 模式，那么只有在执行提交操作后才会应用更新。有关详细信息，请参见「UltraLite 中的事务处理」第 54 页。

注意
不能更新行的主键：而应删除该行并添加新行。

插入行

插入行的步骤与更新行的步骤非常相似，区别在于无需在执行插入操作之前在表中定位行。在表中插入行的顺序无关紧要。

示例

以下代码会插入一个新行。

```
t.insertBegin();
 t.setInt( id, 3 );
 t.setString( lname, "Carlo" );
 t.insert();
```

如果没有设置其中一列的值，并且该列有缺省值，则使用该缺省值。如果该列没有缺省值，将使用以下条目之一。

- 对于可为空的列，添加空值。
- 对于禁止使用空值的数字列，添加 0。
- 对于禁止使用空值的字符列，添加空字符串。
- 若要显式将一个值设置为空值，可使用 setNull 方法。

对于更新操作，在执行提交操作后，插入将永久性地应用到数据库中。在动提交模式中，提交作为插入方法的一部分执行。

删除行

删除行的步骤比插入或更新行的步骤更简单。没有与插入或更新模式对应的删除模式。

以下过程删除一行。

❖ 删除一行

1. 移到想要删除的行。
2. 执行 Table.delete() 方法。

```
t.delete();
```

UltraLite 中的事务处理

UltraLite 提供了事务处理机制以确保数据库中数据的完整性。事务是一个逻辑工作单元。或者执行整个事务，或者不执行事务中的任何语句。

缺省情况下， Native UltraLite for Java 在 [自动提交] 模式下运行，所以每个插入、更新或删除都作为独立的事务来执行。一旦操作完成后，也就完成了对数据库的更改。

如果您将 Connection.AutoCommit 属性设置为 False，则可以使用多语句事务。例如，如果应用程序在两个帐户之间转移资金，要么必须同时减少汇出帐户的金额并增加汇入帐户的金额，要么两个帐户都保持不变。

如果 AutoCommit 设置为 false，则必须执行 Connection.commit() 语句以完成事务并对数据库进行永久更改，否则必须执行 Connection.rollback() 语句取消事务的所有操作。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Connection** 类。

访问模式信息

API 中的对象表示表、列、索引和同步发布。每个对象都有一个 schema 属性，该属性用于访问有关该对象的结构的信息。

您无法通过 API 修改模式，只能检索关于模式的信息。

有关修改模式的信息，请参见 「升级数据库模式」第 36 页。

您可以访问以下模式对象和信息。

- **DatabaseSchema**　提供数据库中表的数量和名称，以及日期和时间格式等全局属性。

  要获取 DatabaseSchema 对象，请访问 Connection.schema。

  有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Connection**。

- **TableSchema**　表的列和索引的数量和名称。

  要获取 TableSchema 对象，请访问 Table.schema。

- **IndexSchema**　关于索引中的列的信息。由于索引没有与其直接关联的数据，因此没有单独的 Index 类，而只有一个 IndexSchema 类。

  要获取 IndexSchema 对象，请调用 TableSchema.getIndex、TableSchema.getOptimalIndex 或 TableSchema.getPrimaryKey 方法。

- **PublicationSchema**　发布中包含的表和列的列表。由于发布也只由模式组成，因此没有 Publication 对象。

  要获取 PublicationSchema 对象，请调用 DatabaseSchema.getPublicationSchema 方法。

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.TableSchema**。

错误处理

可以使用标准 Java 错误处理功能来处理错误。大多数方法都抛出 java.sql.SQLException 错误。可以使用 SQLException.getErrorCode() 来检索指派给此错误的 SQLCode 值。 SQLCode 错误是表示错误类型的负数。

有关错误代码的列表，请参见 『ASA 错误消息』> 「ASA 错误消息」。

在同步之后，您可以使用连接的 SyncResult 属性来获取更详细的错误信息。

有关详细信息，请参见以下内容：

- 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.SyncResult**

用户鉴定

必须从现有连接添加新用户。由于所有 UltraLite 数据库都是用缺省用户 ID 和口令（分别为 DBA 和 SQL）创建的，因此必须首先以该初始用户的身份进行连接。

不能更改用户 ID。而只能添加一个用户，然后删除现有用户。每个 UltraLite 数据库最多允许使用四个用户 ID。

有关详细信息，请参见 『UltraLite 数据库用户指南』> 「UltraLite 中的用户鉴定」。

❖ 添加用户或更改现有用户的口令

1. 以具有 DBA 权限的用户身份连接到数据库。
2. 使用 Connection.grantConnectTo 方法，以设定所需口令的方式授予用户对数据库的访问权限。

   无论要添加新用户还是要更改现有用户的口令，此过程均相同。

   有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **ianywhere.native_ultralite.Connection**。

❖ 删除现有用户

1. 以具有 DBA 权限的用户身份连接到数据库。
2. 使用 Connection.revokeConnectFrom 方法撤消用户的连接权限。

同步 UltraLite 应用程序

您可以将 UltraLite 应用程序与统一数据库同步。同步需要使用 SQL Anywhere Studio 附带的 MobiLink 同步软件。

本节简单介绍了同步，并说明了 Native UltraLite for Java 的用户特别感兴趣的一些功能。

有关同步的详细说明，请参见 『MobiLink 客户端』> 「UltraLite 客户端」。

您还可以参见 CustDB 示例应用程序中的同步工作示例。有关详细信息，请参见 SQL Anywhere 9 的安装子目录 *Samples\Ultralite\NativeUltraLiteForJava\CustDB*。

Native UltraLite for Java 支持 TCP/IP、 HTTP 和 HTTPS 同步。同步由 UltraLite 应用程序启动。在所有情况中，都可以使用 SyncParms 对象的属性来控制同步。

注意

要使用 HTTPS 进行同步，必须单独购买安全性选件的许可证。要订购该选件，请参见 SQL Anywhere Studio 程序包中的卡片或者访问 http://www.sybase.com/detail?id=1015780。

有关详细信息，请参见 『SQL Anywhere Studio 介绍』> 「欢迎使用 SQL Anywhere Studio」。

向应用程序添加 ActiveSync 同步

这一节介绍如何给 Native UltraLite for Java 应用程序添加 ActiveSync，以及如何在最终用户的计算机上注册您的应用程序以便与 ActiveSync 配合使用。通过 Jeode 和 CrEme Java VM，可在 Windows CE 设备上提供 ActiveSync。

如果您使用的是 CrEme VM 并且要使用 ActiveSync，必须给类路径添加 *\Windows\CrEme\lib\crmex.jar*。

同步需要 SQL Anywhere Studio。有关设置 ActiveSync 同步的一般信息，请参见 『MobiLink 客户端』> 「部署使用 ActiveSync 的应用程序」。有关在应用程序中添加同步的一般信息，请参见 『MobiLink 客户端』> 「UltraLite 客户端」。

ActiveSync 同步只能由 ActiveSync 启动。当设备放置在底座中或者从 ActiveSync 窗口选择 [同步] 命令时， ActiveSync 会启动同步操作。

当 ActiveSync 启动同步时， MobiLink ActiveSync 提供程序也启动 UltraLite 应用程序 （如果它尚未运行）并向它发送消息。应用程序必须执行 ActiveSyncListener 以接收和处理来自 MobiLink 提供程序的消息。应用程序必须使用 setActiveSyncListener 方法指定 listener 对象，其中 MyAppClassName 是应用程序的唯一 Windows 类名。

```
dbMgr.setActiveSyncListener(
    listener, "MyAppClassName" );
```

有关详细信息，请参见 『Native UltraLite for Java 用户指南』> 「Native UltraLite for Java API 参考」中的 **anywhere.native_ultralite.DatabaseManager**。

当 UltraLite 收到 ActiveSync 消息时，它将在另一个线程上调用指定监听器的 activeSyncInvoked(boolean) 方法。为避免多线程问题，activeSyncInvoked(boolean) 方法应将事件发布到用户界面。

如果应用程序是多线程的，请使用一个单独的连接并利用 Java **synchronized** 关键字来访问与应用程序的其余部分共享的任何对象。activeSyncInvoked() 方法应为其连接的 syncInfo 流指定一个 StreamType.ACTIVE_SYNC，然后调用 Connection.synchronize()。

注册应用程序时，请设置以下参数。

- **类名**　该应用程序用于 Connection.setActiveSyncListener 方法的同一个类名。
- **路径**　Jeode VM 的路径 (*\Windows\evm.exe*)。
- **参数**　包括类路径 (`-cp`) 和其它 Jeode 命令行参数、应用程序名和应用程序参数。

如果通过指定唯一参数来表示 ActiveSync 激活，那么应用程序可以执行特殊的启动序列，该序列知道应用程序将在 ActiveSync 同步完成时关闭。

CustDB 和 ActiveSync

CustDB 的 Native UltraLite for Java 版本的示例通过使用套接字的应用程序菜单和通过 ActiveSync 提供同步示例。

您可以在 *Samples\NativeUltraLiteForJava\CustDB\Application.java* （位于 SQL Anywhere 目录下）中找到此示例的源代码。本节将说明此示例应用程序中的代码。

- CustDB 通过分析自己的参数来检查用于指示它被 ActiveSync 的 MobiLink 提供程序启动的特殊标志。这允许它加速初始化过程（比如避免表单填充），原因是为 ActiveSync 启动的应用程序需要在同步后关闭。

```
// Normal versus Active sync launch
 boolean isNormalLaunch = true;
 int alen = args.length;
 if( alen > 0 ) {
   String asflag = args[ alen - 1 ].toUpperCase();
  if( asflag.compareTo( "ACTIVE_SYNC_LAUNCH" ) ==
0) {
     isNormalLaunch = false;
     --alen;
   }
```

- 对于正常启动 （即非 ActiveSync 启动）， CustDB 执行连接初始化并确定雇员 ID。然后，它通过指定监听器为 ActiveSync 进行初始化并装载其主表单。对于 ActiveSync 启动， CustDB 执行 ActiveSync 同步，然后关闭。

```
if( isNormalLaunch ) {
   db.initActiveSync( "JULCustDB", main );
   db.getOrder( 1 );
 } else {
   // ActiveSync launch
   db.activeSync( false );
   main.quit();
 }
 public void initActiveSync(
```

```
    String appName, ActiveSyncListener listener )
 {
   DEBUG( "initActiveSync" );
   _conn.setActiveSyncListener( appName, listener
);
 }
 public void activeSync( boolean useDialog )
 {
   try {
     // Change stream
     _conn.syncInfo.setStream(
                      StreamType.ACTIVE_SYNC );
     // since if "stream=" not in parms,
     //it defaults to tcpip, no
     // need to change stream parms
     _conn.synchronize( useDialog );
     freeLists();
     allocateLists();
     skipToValidOrder();
   } catch( SQLException e ) {
     System.out.println(
     "Can't synchronize, sql code=" +
     e.getErrorCode()
       );
   }
 }
```

- Application 类可实现 ActiveSyncListener 接口，以便能够通知正在运行的应用程序执行 ActiveSync 同步。

```
public class Application
 extends Frame
 implements ActionListener,
 // ActiveSyncListener functional only on CE
devices
 ActiveSyncListener
```

- 调用 activeSyncInvoked() 时，它将消息发布到 UI 线程。

```
static final int ACTIVE_SYNC_EVENT_MASK =
    AWTEvent.RESERVED_ID_MAX + 1;
 static class ActiveSyncEvent extends AWTEvent {
    ActiveSyncEvent( Object source ) {
       super( source, ACTIVE_SYNC_EVENT_MASK );
    }
```

```
/** Process ActiveSync message
*/
public void activeSyncInvoked(
        boolean launchedByProvider ) {
  // This method is invoked on a special thread.
  // Post an event so that active sync
  // takes places on the same thread
  // as the rest of the application.
  DEBUG( "activeSyncInvoked()" );
  getToolkit().getSystemEventQueue().postEvent(
            new ActiveSyncEvent( this )
  );
  DEBUG( "ActiveSync Event posted" );
}
```

- UI 线程通过覆盖 processEvent 捕获消息

```
/** Intercept my special action events
  *    for ActiveSync
  */
protected void processEvent( AWTEvent e ) {
  if( e instanceof ActiveSyncEvent ) {
    _db.activeSync( true );
    refresh();
  } else {
    super.processEvent( e );
  }
}
```

但是，必须启用接收事件的应用程序。这通过应用程序的构造函数来完成。

```
// ActiveSync support
enableEvents( ACTIVE_SYNC_EVENT_MASK );
```

使用 Borland JBuilder 开发应用程序

Borland JBuilder 是 Java 应用程序开发环境。 Native UltraLite for Java 包括与 JBuilder 7 和 JBuilder 8 的集成。本节介绍了如何在 JBuilder 环境中使用 Native UltraLite for Java。

准备在 JBuilder 中使用 Native UltraLite for Java

如果安装了 JBuilder，则 UltraLite 安装程序就可以启用 JBuilder 集成。如果 JBuilder 是在 UltraLite 之后安装的，请重新运行 UltraLite 安装程序以启用 JBuilder 集成。

设置 JDK

开发 Native UltraLite for Java 应用程序时，您应当使用 JDK 1.1.8 或 Personal Java 1.2，以便与 Windows CE 设备上的 Jeode VM 兼容。

JBuilder SE 和 Enterprise 完全支持 JDK 转换，并且 JBuilder Personal 允许您编辑单个 JDK。

❖ **设置 JDK 版本：**

1. 在 JBuilder Personal 中，单击 [Tools] → [Configure JDK]，然后编辑 JDK。
2. 在 JBuilder SE 和 Enterprise 中，右击项目窗格中的项目文件，然后选择 [Properties]。在 [Paths] 选项卡上，单击 JDK 并查找您需要的 JDK 位置。

使用 Native UltraLite for Java 设置

Native UltraLite for Java 设置可以用于现有项目或新建项目。

❖ **向 JBuilder 项目添加 UltraLite 功能：**

1. 打开一个 JBuilder 项目。
2. 添加 Native UltraLite for Java 设置。

- 选择 [File] → [New]。随即出现 [Object Gallery]。
- 在 [General] 选项卡上，选择 [Native UltraLite Setup]，然后单击 [OK]。
- 选择一个数据库名称和在 Windows CE 设备上的部署目录。单击 [下一步]。
- 添加 Windows CE 快捷方式的名称 （它将显示在 [开始] 菜单中）、 JAR 名称和主类。单击 [完成]。

一个链接文件被添加到项目中。该链接文件用于在 Windows CE 设备上运行应用程序。

Native UltraLite for Java 设置对 JBuilder 项目进行了以下更改。

- 在可用库列表中添加 Native UltraLite for Java。
- 为代码理解模板添加项目属性。
- 修改运行时配置，以便在从开发环境中运行应用程序时定位 *jul9.dll*。

使用 Native UltraLite for Java 模板

在开发过程中，您可以将 Native UltraLite for Java 代码模板作为代码的一些标准部分。若要使用模板，请在 .java 文件中的适当位置输入模板名称，然后按 CTRL+J 键扩展模板。

提供了以下模板。

- **julimp** 添加一行以导入 Native UltraLite for Java 程序包。在文件的导入部分使用此模板。
- **juldb** 添加代码以声明 DatabaseManager 对象。
- **julconn** 添加代码以连接到数据库。
- **julskel** 添加 juldb 和 julconn 代码，作为一个 `main` 方法。

从 JBuilder 访问 Native UltraLite for Java 实用程序

您可以从 JBuilder 接口中访问以下 Native UltraLite for Java 实用程序。

- **模式管理器**　UltraLite 模式管理器是一个用于创建和编辑数据库定义的工具。

  若要打开模式管理器，请选择 [工具] → [UltraLite 模式管理器]。

- **联机帮助**　此文档可通过界面访问。

  要打开联机帮助，请选择 [帮助] → [Native UltraLite 参考]。

第 5 章

Native UltraLite for Java API 参考

关于本章

Native UltraLite for Java 程序包的名称为 ianywhere.native_ultralite。在 SQL Anywhere 9 的安装目录的 *docs\javadocs\NativeUltraLiteForJava* 子目录中提供了 Javadoc 格式的 Native UltraLite for Java API 参考。

有关详细信息，请参见以下类：

- **ianywhere.native_ultralite.ActiveSyncListener**
- **ianywhere.native_ultralite.AuthStatusCode**
- **ianywhere.native_ultralite.Connection**
- **ianywhere.native_ultralite.ConnectionParms**
- **ianywhere.native_ultralite.CreateParms**
- **ianywhere.native_ultralite.Cursor**
- **ianywhere.native_ultralite.CursorSchema**
- **ianywhere.native_ultralite.DatabaseManager**
- **ianywhere.native_ultralite.DatabaseNameParms**
- **ianywhere.native_ultralite.DatabaseSchema**
- **ianywhere.native_ultralite.IndexSchema**
- **ianywhere.native_ultralite.PreparedStatement**
- **ianywhere.native_ultralite.ResultSet**
- **ianywhere.native_ultralite.ResultSetSchema**
- **ianywhere.native_ultralite.SchemaParms**
- **ianywhere.native_ultralite.SchemaUpgradeData**
- **ianywhere.native_ultralite.SchemaUpgradeListener**
- **ianywhere.native_ultralite.SQLCode**
- **ianywhere.native_ultralite.SQLType**
- **ianywhere.native_ultralite.StreamErrorCode**
- **ianywhere.native_ultralite.StreamErrorID**
- **ianywhere.native_ultralite.StreamType**
- **ianywhere.native_ultralite.SyncParms**
- **ianywhere.native_ultralite.SyncProgressData**
- **ianywhere.native_ultralite.SyncProgressDialog**
- **ianywhere.native_ultralite.SyncProgressListener**

- **ianywhere.native_ultralite.SyncResult**
- **ianywhere.native_ultralite.Table**
- **ianywhere.native_ultralite.TableSchema**
- **ianywhere.native_ultralite.UUID**

索引

字母

B

C

D

F

G

H

J

K

L

M

P

S

T

W

X

Y

Z