SYBASE iAnywhere

UltraLite® - M-Business Anywhere 编程

2009 年 2 月

11.0.1 版

版权和商标

目录

关于本手册 v

关于 SQL Anywhere 文档 vi

UltraLite for M-Business Anywhere 简介 1

UltraLite for M-Business Anywhere 功能 2
UltraLite for M-Business Anywhere 体系结构 3

了解 UltraLite for M-Business Anywhere 开发 5

UltraLite for M-Business Anywhere 快速入门 6
连接到 UltraLite 数据库 10
跨页维护连接和应用程序状态 11
M-Business Anywhere 应用程序中的永久名称 12
数据库加密和模糊处理 15
使用 SQL 处理数据 16
使用 Table API 处理数据 20
访问模式信息 26
处理错误 27
验证用户 28
同步数据 29
部署 UltraLite for M-Business Anywhere 应用程序 32

教程：M-Business Anywhere 示例应用程序 35

M-Business Anywhere 开发教程简介 36
第 1 课：建立数据库 37
第 2 课：创建应用程序文件 39
第 3 课：配置 M-Business Anywhere 40
第 4 课：向应用程序添加启动代码 41
第 5 课：添加数据操作和导航 43
第 6 课：将导航添加到应用程序 46
第 7 课：将同步添加到应用程序 47

main.htm 和 tutorial.js 的列表 48

UltraLite for M-Business Anywhere API 参考 53

UltraLite for M-Business Anywhere 中的数据类型 54
AuthStatusCode 类 55
Connection 类 56
ConnectionParms 类 70
CreationParms 类 72
DatabaseManager 类 74
DatabaseSchema 类 78
IndexSchema 类 83
PreparedStatement 类 86
PublicationSchema 类 95
ResultSet 类 96
ResultSetSchema 类 113
SQLError 类 117
SQLType 类 126
SyncParms 类 128
SyncResult 类 138
TableSchema 类 141
ULTable 类 152
UUID 类 177

术语表 179

术语表 181

索引 209

关于本手册

主题

本手册介绍 UltraLite for M-Business Anywhere。利用 UltraLite for M-Business Anywhere，用户可以开发基于 Web 的数据库应用程序，并将它们部署到运行 Palm OS、Windows Mobile 或 Windows 的手持式设备、移动设备或嵌入式设备。

M-Business Anywhere 是 iAnywhere 平台，用于开发和部署基于 Web 的移动应用程序。此产品以前的名称是 AvantGo M-Business Server。

读者

本手册适用于希望在数据存储和同步过程中利用 UltraLite 关系数据库的高性能、资源效率、稳健性和安全性的应用程序开发人员。

关于 SQL Anywhere 文档

完整的 SQL Anywhere 文档以四种形式提供，但所包含信息均相同。

- **HTML 帮助** 联机帮助文档包含完整的 SQL Anywhere 文档，其中包括手册和 SQL Anywhere 工具的上下文相关帮助。

  如果使用 Microsoft Windows 操作系统，则联机帮助文档以 HTML 帮助 (CHM) 格式提供。若要访问此文档，请选择 [开始] » [程序] » [**SQL Anywhere 11**] » [文档] » [联机手册]。

  管理工具使用同一联机文档来实现帮助功能。

- **Eclipse** 在 Unix 平台上以 Eclipse 格式提供完整的联机帮助。要访问文档，请从 SQL Anywhere 11 安装的 *bin32* 或 *bin64* 目录下运行 *sadoc*。

- **DocCommentXchange** DocCommentXchange 是一个用于访问和讨论 SQL Anywhere 文档的社区。

  使用 DocCommentXchange 可以执行以下任务：

  - 查看文档
  - 检查是否有用户对文档各部分所做出的阐明
  - 提供建议和修正意见以在将来的版本中为所有用户改进文档

  访问 http://dcx.sybase.com。

- **PDF** 整套 SQL Anywhere 手册会以一组 Portable Document Format (PDF) 文件的形式提供。您必须有 PDF 阅读器才能查看信息。要下载 Adobe Reader，请访问 http://get.adobe.com/reader/。

  若要在 Microsoft Windows 操作系统上访问 PDF 文档，请选择 [开始] » [程序] » [**SQL Anywhere 11**] » 文档 » [联机手册 - PDF 格式]。

  要在 Unix 操作系统上访问 PDF 文档，请使用 Web 浏览器打开 *install-dir/documentation/zh/pdf/index.html*。

关于文档集中的手册

SQL Anywhere 文档由以下手册组成：

- **SQL Anywhere 11 - 简介** 本手册介绍 SQL Anywhere 11，一个提供数据管理和数据交换技术的综合数据包，通过它可以为服务器环境、台式机环境、移动环境以及远程办公环境快速开发由数据库驱动的应用程序。
- **SQL Anywhere 11 - 更改和升级** 本手册介绍 SQL Anywhere 11 以及该软件以前版本中的新功能。
- **SQL Anywhere 服务器 - 数据库管理** 本手册介绍如何运行、管理及配置 SQL Anywhere 数据库。它介绍了数据库连接、数据库服务器、数据库文件、备份过程、安全性、高可用性、使用复制服务器进行复制以及管理实用程序和选项。

- **SQL Anywhere 服务器 - 编程** 本手册介绍如何使用 C、C++、Java、PHP、Perl、Python 和 .NET 编程语言（例如 Visual Basic 和 Visual C#）建立和部署数据库应用程序。其中介绍了各种编程接口，如 ADO.NET 和 ODBC。
- **SQL Anywhere 服务器 - SQL 参考** 本手册提供了系统过程和目录（系统表和视图）的参考信息。也介绍了 SQL 语言（搜索条件、语法、数据类型和函数）的 SQL Anywhere 实现。
- **SQL Anywhere 服务器 - SQL 的用法** 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。
- **MobiLink - 入门** 本手册介绍基于会话的关系数据库同步系统 MobiLink。MobiLink 技术支持双向复制并且非常适用于移动计算环境。
- **MobiLink - 客户端管理** 本手册介绍如何设置、配置和同步 MobiLink 客户端。MobiLink 客户端可以是 SQL Anywhere 或者 UltraLite 数据库。本手册同时也介绍了 Dbmlsync API，通过它可无缝地将同步集成到 C++ 或 .NET 客户端应用程序中。
- **MobiLink - 服务器管理** 本手册说明如何设置和管理 MobiLink 应用程序。
- **MobiLink - 服务器启动的同步** 本手册介绍 MobiLink 服务器启动的同步，这种功能允许 MobiLink 服务器启动同步或在远程设备上进行操作。
- **QAnywhere** 本手册介绍 QAnywhere，一个用于移动、无线、台式机和膝上型客户端的消息传递平台。
- **SQL Remote** 本手册介绍用于移动计算的 SQL Remote 数据复制系统，此系统支持使用电子邮件或文件传输等间接链接共享 SQL Anywhere 统一数据库和多个 SQL Anywhere 远程数据库之间的数据。
- **UltraLite - 数据库管理和参考** 本手册介绍适用于小型设备的 UltraLite 数据库系统。
- **UltraLite - C 及 C++ 编程** 本手册介绍 UltraLite C 和 C++ 编程接口。利用 UltraLite，可以开发数据库应用程序，并将它们部署到手持式设备、移动设备或嵌入式设备。
- **UltraLite - M-Business Anywhere 编程** 本手册介绍 UltraLite for M-Business Anywhere。利用 UltraLite for M-Business Anywhere，用户可以开发基于 Web 的数据库应用程序，并将它们部署到运行 Palm OS、Windows Mobile 或 Windows 的手持式设备、移动设备或嵌入式设备。
- **UltraLite - .NET 编程** 本手册介绍 UltraLite.NET。利用 UltraLite.NET，您可以开发数据库应用程序，并将它们部署到计算机、手持式设备、移动设备或嵌入式设备。
- **UltraLiteJ** 本手册介绍 UltraLiteJ。利用 UltraLiteJ，可以在支持 Java 的环境中开发和部署数据库应用程序。UltraLiteJ 支持 BlackBerry 智能手机和 Java SE 环境。UltraLiteJ 基于 iAnywhere UltraLite 数据库产品。
- **错误消息** 本手册提供了 SQL Anywhere 错误消息及其诊断信息的完整列表。

文档约定

本节列出了本文档中使用的约定。

操作系统

SQL Anywhere 可以在各种平台上运行。在大多数情况下，该软件在所有平台上的行为都是相同的，但也有变动或限制。这些变动或限制通常基于基础操作系统（Windows、Unix），很少基于特定变型（AIX、Windows Mobile）或版本。

为了简化对操作系统的提及，本文档按如下方式对支持的操作系统进行分组：

- **Windows** Microsoft Windows 系列包括 Windows Vista 和 Windows XP（主要用于服务器、台式计算机和膝上型计算机），以及 Windows Mobile（用于移动设备）。

  除非另外指定，否则当本文档提及 Windows 时，是指所有基于 Windows 的平台，包括 Windows Mobile。

- **Unix** 除非另外指定，否则当本文档提及 Unix 时，是指所有基于 Unix 的平台，包括 Linux 和 Mac OS X。

目录和文件名

大部分情况下，对目录和文件名的引用在所有支持的平台上都是类似的，只需在不同形式之间进行简单的转换。这时需使用 Windows 约定。在细节更为复杂的情况下，文档显示所有相关形式。

下面是文档编写中用于简化目录和文件名的约定：

- **大写和小写目录名** 在 Windows 和 Unix 上，目录和文件名可以包括大写和小写字母。创建目录和文件时，文件系统会保留字母大小写。

  在 Windows 上，对目录和文件的提及*不区分大小写*。混合使用大小写的目录和文件名很常见，但使用所有小写字母来提及目录和文件的形式也很常见。SQL Anywhere 安装包含诸如 *Bin32* 和 *Documentation* 的目录。

  在 Unix 上，对目录和文件的提及*区分大小写*。混合使用大小写的目录和文件名不常见。大多数的目录和文件名全部使用小写字母。SQL Anywhere 安装包含诸如 *bin32* 和 *documentation* 的目录。

  本文档采用 Windows 形式的目录名。大多数情况下，在 Unix 上可以将大小写混合形式的目录名转换成小写字母的等效目录名。

- **分隔目录和文件名的斜线** 文档使用反斜线作为目录分隔符。例如，PDF 格式的文档位于 *install-dir\Documentation\zh\PDF*（Windows 形式）。

  在 Unix 上，用正斜线替换反斜线。PDF 文档位于 *install-dir/documentation/zh/pdf* 下。

- **可执行文件** 文档使用 Windows 约定显示可执行文件名（带有诸如 *.exe* 或 *.bat* 后缀）。在 Unix 上，可执行文件名没有后缀。

  例如，在 Windows 上，网络数据库服务器是 *dbsrv11.exe*。在 Unix 上是 *dbsrv11*。

- ***install-dir*** 在安装过程中，选择 SQL Anywhere 的安装位置。创建环境变量 SQLANY11，用来表示此位置。文档中以 *install-dir* 表示此位置。

  例如，本文档将此文件表示为 *install-dir*\readme.txt。在 Windows 上，这等同于 *%SQLANY11%\readme.txt*。在 Unix 上，这等同于 *$SQLANY11/readme.txt* 或 *${SQLANY11}/readme.txt*。

有关 *install-dir* 缺省位置的详细信息，请参见“SQLANY11 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

- ***samples-dir*** 在安装过程中，选择 SQL Anywhere 随附的示例的安装位置。创建环境变量 SQLANYSAMP11，用来表示此位置。文档中以 *samples-dir* 表示此位置。

  要在 *samples-dir* 中打开 Windows 资源管理器窗口，请在 [**开始**] 菜单中，选择 [**程序**] » [**SQL Anywhere 11**] » [**示例应用程序和项目**]。

  有关 *samples-dir* 缺省位置的详细信息，请参见“SQLANYSAMP11 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

命令提示符和命令 shell 语法

大多数操作系统都提供一种或多种使用命令 shell 或命令提示符来输入命令和参数的方法。Windows 命令提示符包括 Command Prompt（DOS 提示符）和 4NT。Unix 命令 shell 包括 Korn shell 和 bash。每个 shell 都具有一些功能，其能力不仅仅局限于简单命令。这些功能通过特殊字符来驱动。特殊字符和功能随 shell 的不同而不同。如果没有正确使用这些特殊字符，通常会导致语法错误或意外行为。

本文档以普通形式提供命令行示例。如果这些示例中包含 shell 的特殊字符，则命令需要根据特定 shell 进行修改。修改方法不在本文档所述范围之内，但通常是在包含这些特殊字符的参数两旁加上引号，或是在特殊字符前面使用转义字符。

下面是命令行语法的一些示例，不同的平台可能会有不同的形式：

- **括号和大括号** 有些命令行选项需要一个参数，该参数将以列表形式接受详细的值指定。该列表通常用括号或大括号括起来。本文档使用括号。例如：

  ```
  -x tcpip(host=127.0.0.1)
  ```

  如果括号导致出现语法问题，用大括号替代：

  ```
  -x tcpip{host=127.0.0.1}
  ```

  如果两种形式都将产生语法问题，应按照 shell 的要求，用引号将整个参数括起来：

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **引号** 如果必须在参数值中指定引号，该引号可能会与用于括参数的引号的传统用法发生冲突。例如，要指定值中包含双引号的加密密钥，则可能必须用引号括起密钥，然后转义嵌入的引号：

  ```
  -ek "my \"secret\" key"
  ```

  在许多 shell 中，密钥的值为 `my "secret" key`。

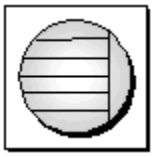
- **环境变量** 本文档介绍设置环境变量。在 Windows shell 中，环境变量使用语法 *%ENVVAR%* 来指定。在 Unix shell 中，环境变量使用语法 *$ENVVAR* 或 *${ENVVAR}* 来指定。

图标

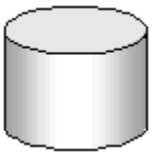
本文档中使用了下列图标。

- 客户端应用程序。

- 数据库服务器，如 Sybase SQL 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该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 服务器和 SQL Remote 消息代理。

- 编程接口。

联系文档小组

我们欢迎您就本帮助文档提出意见、建议和反馈信息。

要提交意见和建议，请发送电子邮件到 SQL Anywhere 文档小组，地址为 iasdoc@sybase.com。虽然我们不对这些电子邮件进行回复，但您的反馈会帮助我们改进文档，因此我们真诚地欢迎您提出宝贵的意见和建议。

DocCommentXchange

也可以使用 DocCommentXchange 将意见或建议直接置于帮助主题中。DocCommentXchange (DCX) 是一个用于访问和讨论 SQL Anywhere 文档的社区。使用 DocCommentXchange 可以执行以下任务：

- 查看文档
- 检查是否有用户对文档各部分所做出的阐明
- 提供建议和修正意见以在将来的版本中为所有用户改进文档

访问 http://dcx.sybase.com。

查找详细信息并请求技术支持

附加信息和资源可从 Sybase iAnywhere 开发人员社区获得，网址是 http://www.sybase.com/developer/library/sql-anywhere-techcorner。

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

当您向这些新闻组发布邮件时，请务必提供问题的详细信息，包括 SQL Anywhere 版本的内部版本号。可以通过运行以下命令找到此信息：**dbeng11 -v**。 **dbeng11 -v**.

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

有关 Web 开发问题，请访问 http://groups.google.com/group/sql-anywhere-web-development。

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 也没有义务提供任何其它服务。

如果时间允许，iAnywhere 技术顾问以及其他员工也会对新闻组服务提供帮助。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。

UltraLite for M-Business Anywhere 简介

目录

UltraLite for M-Business Anywhere 功能 **2**
UltraLite for M-Business Anywhere 体系结构 **3**

UltraLite for M-Business Anywhere 功能

UltraLite for M-Business Anywhere 是一种用于移动设备的关系数据管理系统。它具有业务应用程序所要求的性能、资源效率、稳健性和安全性。UltraLite 还提供与企业数据存储库的同步。

系统要求和支持的平台

开发平台

要使用 UltraLite for M-Business Anywhere 开发应用程序，需要以下各项：

- M-Business Anywhere 是 AvantGo M-Business Server 的新名称。此软件需要 M-Business Server 5.3 或更高版本，以及相应的 M-Business Anywhere 客户端。

目标平台

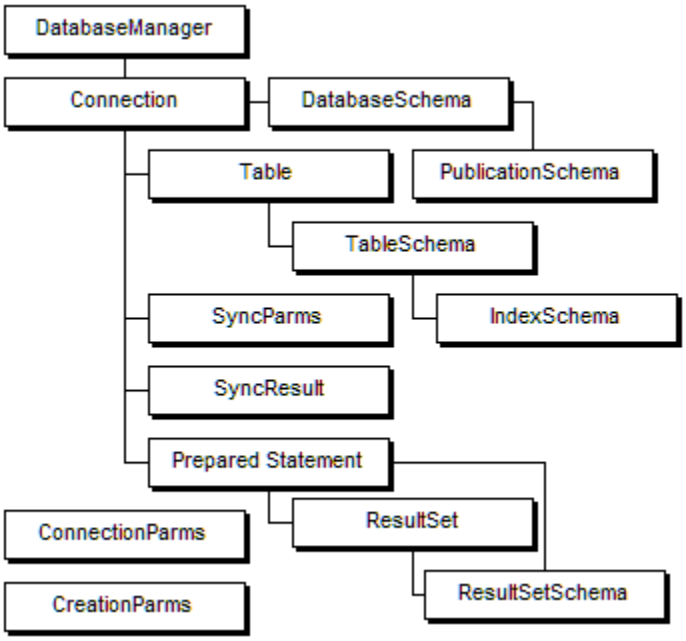
UltraLite for M-Business Anywhere 支持以下目标平台：

- 使用 ARM 处理器的 Pocket PC 上的 Windows Mobile 3.0 和更高版本，包括 Windows Mobile 5.0。
- Palm OS 版本 5.0 和更高版本。
- Windows（从 M-Business Anywhere 5.5 开始）。

有关部署的详细信息，请参见 http://www.sybase.com/detail?id=1062617。

UltraLite for M-Business Anywhere 体系结构

UltraLite 编程接口提供一组对象，用于使用 UltraLite 数据库的数据操作。下图介绍了对象层次。



以下列表介绍了一些比较常用的高层次对象。

- **DatabaseManager** 管理与 UltraLite 数据库的连接。请参见“DatabaseManager 类”一节第 74 页。
- **ConnectionParms** 包含一组连接参数。请参见“ConnectionParms 类”一节第 70 页。
- **CreationParms** 包含一组数据库创建参数。请参见“CreationParms 类”一节第 72 页。
- **连接** 用于表示数据库连接并控制事务。请参见“Connection 类”一节第 56 页。
- **PreparedStatement、ResultSet 和 ResultSetSchema** 使用 SQL 管理数据库请求及其结果。请参见：
  - “PreparedStatement 类”一节第 86 页
  - “ResultSet 类”一节第 96 页
  - “ResultSetSchema 类”一节第 113 页
- **Table** 使用基于表的 API 管理数据。请参见“ULTable 类”一节第 152 页。
- **SyncParms 和 SyncResult** 通过 MobiLink 服务器管理同步。

  有关使用 MobiLink 同步的详细信息，请参见“UltraLite 客户端” 《UltraLite - 数据库管理和参考》。

了解 UltraLite for M-Business Anywhere 开发

目录

UltraLite for M-Business Anywhere 快速入门 6
连接到 UltraLite 数据库 10
跨页维护连接和应用程序状态 11
M-Business Anywhere 应用程序中的永久名称 12
数据库加密和模糊处理 15
使用 SQL 处理数据 16
使用 Table API 处理数据 20
访问模式信息 26
处理错误 27
验证用户 28
同步数据 29
部署 UltraLite for M-Business Anywhere 应用程序 32

UltraLite for M-Business Anywhere 快速入门

以下过程介绍了如何运行提供的 CustDB 和 Simple 示例应用程序。

启动之前，要确保您已安装并运行 M-Business Anywhere 6.0 或更高版本，并且您在服务器上拥有管理权限。您还必须拥有一台支持的手持式设备。

♦ 安装并运行 M-Business Anywhere 示例

1. 将 UltraLite for M-Business Anywhere 示例文件复制到安装目录中以进行部署。
   a. 打开命令提示符，将目录更改为 SQL Anywhere 安装目录的 *samples-dir\UltraLiteForMBusinessAnywhere\CustDB* 子目录。
   b. 运行以下命令：

      ```
      build.bat deploy-dir
      ```

      其中 *deploy-dir* 是要部署 CustDB 和 UltraLite 文件的目录。例如，您可能选择 *C:\tutorial\mba*。

      批处理文件将所需的文件复制到您指定的位置。要查看正在复制哪些文件，可使用文本编辑器查看文件 *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\build.bat*。
2. 在 Web 服务器上创建一个虚拟目录，该目录指向第 1 步中指定的 *deploy-dir* 目录。以下是针对 Microsoft IIS 的说明：
   a. 打开 IIS 管理工具。
   b. 右击 Web 站点并选择 [**New**] » [**Virtual Directory**]。将该虚拟目录命名为 **CustDB** 并指定您的部署目录 *deploy-dir* 作为内容目录。将其它设置保留为其缺省值。
   c. 右击新的虚拟目录并选择 [**Properties**]。在 [**HTTP Headers**] 选项卡上，单击 [**File Types**] 并将以下文件扩展名注册为 application/octet-stream 类型：
      - 对于 Windows 和 Windows Mobile：cab、dll
      - 对于 Palm OS：pdb、prc
      - udb
   d. 记录下访问此虚拟目录中的 *main.htm* 文件的 URL。在缺省安装下此 URL 为 *http://localhost/CustDB/main.htm*。
3. 向 M-Business Anywhere 添加用户。

   有以下三种方法向 M-Business Anywhere 添加新用户：通过创建新用户配置文件、通过允许用户自行注册和通过导入 CSV 文件。这些说明介绍了怎样创建新用户配置文件。有关详细信息，请参见 M-Business Anywhere 文档。
   a. 以管理员身份登录到 M-Business Anywhere。

      缺省的管理员帐户设置用户 ID 为 **Admin**，口令为空。
   b. 在左侧面板中，单击 [**Users**]。
   c. 单击 [**Create User**]。
   d. 在 [**User Name**] 字段中键入唯一的用户名。

e. 在 [**Password**] 和 [**Confirm Password**] 字段中键入口令。

f. 单击 [**保存**]。

4. 将 M-Business Anywhere 客户端部署到手持式设备或 PC 中。

a. 在 M-Business Anywhere 登录页上单击 [**Download Client Software Only**] 链接。运行安装程序安装客户端。

b. 在手持式设备或 PC 上，配置 M-Business 连接使其与 M-Business Anywhere 服务器同步。

输入您所创建的新 M-Business 用户帐户的用户 ID 和口令。

c. 与 M-Business Anywhere 服务器同步。

如果在这阶段您的连接出现了问题，请使用 IP 号而不要使用主机名指定主机（以避免 ActiveSync 的一些版本出现名称解析问题）。

有关详细信息，请参见 M-Business Anywhere 文档。

5. 向 M-Business Anywhere 添加组。

该组将用来测试 UltraLite for M-Business Anywhere。

a. 从 Web 浏览器连接到 M-Business Anywhere。

缺省 URL 是 *http://localhost* 或 *http://localhost:8091*。

b. 使用管理员帐户登录。

c. 在左侧导航面板中，单击 [**Groups**]，然后单击 [**Create Group**]。

d. 将您的组命名为 **UltraLite Samples**。

6. 配置 M-Business Anywhere 通道设置。

a. 在 [**Edit Group**] 下的左侧面板中，使用 [Users] 选项将您在第 3 步中创建的用户添加到 UltraLite Samples 组中。

b. 使用左侧导航面板中组的 "Channels" 选项创建以下通道：

设置	**值**
通道名称	CustDB
位置	*http://localhost/CustDB/main.htm* 或第 2 步中的 URL。
大小	1000
链接深度	3
允许二进制分布	Yes（选中）
隐藏	No（未选中）

设置位置值后，单击 [**View**] 确认您已正确地输入了值。

7. 同步您的客户端。

初始同步将 UltraLite for M-Business Anywhere 文件下载到您的手持式设备上。

♦ **验证您的设置**

1. 检查所需的文件是否存在。
   - 在 Windows Mobile 上，同步了该设备后，检查以下文件是否在 *\Program Files\AvantGo \Pods* 文件夹中：
     - *ulpod11.dll*
     - *custdb.udb*

     如果这些文件中的任何一个丢失了，你可能都要手工将它们复制到该设备中。
   - 在 Palm OS 上，同步了您的设备之后，请检查 Palm OS App Info 查看是否存在以下信息：
     - ulpod
     - custdb

     如果其中的任何一个丢失了，您可能需要使用 Palm 安装实用程序将 UltraLite for M-Business Anywhere 运行库 *.prc* 和示例模式 *.pdb* 文件安装到该设备中。
   - 在 Windows 桌面上，同步了您的设备后，检查以下文件是否在 *AvantGo Connect* 文件夹的 *AvantGo\Pods* 子目录中：
     - *ulpod11.dll*
     - *custdb.udb*

     如果这些文件中的任何一个丢失了，你可能都要手工将它们复制到该设备中。
2. 启动 M-Business 客户端。

   在您的手持式设备或 PC 上，检查 [关于] 屏幕是否显示 UltraLite for M-Business Anywhere 版本号。这就确认了 UltraLite for M-Business Anywhere 已安装成功。
3. 运行 CustDB 示例应用程序。
   a. 在您的台式机上启动 MobiLink 服务器。

      从 [**开始**] 菜单中选择 [**程序**] » [**SQL Anywhere 11**] » [**MobiLink**] » [**同步服务器示例**]。
   b. 在您的 M-Business 客户端上启动 CustDB 应用程序。

      CustDB 应用程序是在 M-Business 主页上的一个链接。
   c. 输入您的用户 ID。

      缺省值为 **50**。
   d. 执行同步。

      对提示 "Do you have a network connection now?" 回答 [是 Yes]，或者在 CustDB 应用程序中，单击 [**Synchronize**]。这使数据与 MobiLink 同步，并且这和与 M-Business Anywhere 同步是两种不同的操作。

      您现在应该能在 [**CustDB**] 字段中看到数据。现在您就可以研究 CustDB 应用程序。

   请参见“研究 MobiLink 的 CustDB 示例” 《MobiLink - 入门》 。

具有多个数据库的 HotSync

在 Palm OS 设备上的每一个 UltraLite 数据库都应该有一个不同的创建者 ID 以与 HotSync 一同正常工作。此外，具有该创建者 ID 的应用程序必须存在于 Palm OS 设备上。

HotSync 管理器将每一个应用程序的创建者 ID 用作标识符来处理同步。它将每一个正确配置的 UltraLite 应用程序传递到 MobiLink 管道进行同步。该管道搜索并同步与应用程序具有相同创建者 ID 的数据库。

有关配置管道的详细信息，请参见“HotSync 同步概述”一节 《UltraLite - 数据库管理和参考》。

所有 UltraLite for M-Business Anywhere 应用程序都继承了 M-Business 客户端的创建者 ID，即 **AvGo**。这种继承意味着只有具有创建者 ID **AvGo** 的 UltraLite 数据库才可以同步，如果您指派一个不同的创建者 ID 到您的数据库中，HotSync 将因为没有具有相应的创建者 ID 的应用程序而无法找到它。

对于这两个示例应用程序（CustDB 和 Simple）来说，这种限制并不是问题，因为它们共享一个公用数据库模式。唯一的副作用是当您同步 CustDB 数据库时，也同步了 Simple 示例。

有关解决此问题的详细信息，请参见“注册 Palm 创建者 ID”一节 《UltraLite - C 及 C++ 编程》。

连接到 UltraLite 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。

这是建立连接最简单的方式。此技术的扩展在以下几节中给出。

```
var DatabaseMgr;
var Connection;
DatabaseMgr = CreateObject("iAnywhere.UltraLite.DatabaseManager.CustDB");
Connection = DatabaseMgr.openConnection("dbf=" + DatabaseMgr.directory + "\
\mydb.udb");
```

使用 Connection 对象

Connection 对象的以下属性控制全局应用程序行为。

有关连接对象的详细信息，请参见“Connection 类”一节第 56 页。

- **提交行为** 缺省情况下，UltraLite 应用程序处于 autoCommit 模式。每个插入、更新或删除语句都被立即提交给数据库。将 Connection.autoCommit 设置为 false，以便将事务构建到您的应用程序中。关闭 autoCommit，然后直接执行提交，可改善应用程序的性能。请参见“commit 方法”一节第 58 页。
- **用户验证** 可以使用 grantConnectTo 和 revokeConnectFrom 方法更改应用程序的用户 ID 缺省值 **DBA** 和口令缺省值 **sql**。请参见“验证用户”一节第 28 页。
- **同步** 可以通过 Connection 对象访问控制同步的一组对象。请参见“同步数据”一节第 29 页。
- **表** UltraLite 表可以使用 Connection.getTable 方法来访问。请参见“getTable 方法”一节第 64 页。

跨页维护连接和应用程序状态

JavaScript 变量的作用域限制在一个 Web 页上。大多数 Web 应用程序都需要多个页面，所以需要一种机制使一些对象具有跨应用程序页面的持久性。

UltraLite for M-Business Anywhere 为 ULTable、ResultSet 和 PreparedStatement 对象提供持久性。要使这些对象之一具有跨页持久性，可在创建对象时提供 **persistent name** 作为一个参数。您可以在后续页中使用该永久名称。

要在页间传送连接对象，您需要在每一页上重新打开连接。实现此操作的一种方法就是使用 reOpen 方法。另一种方法是在每一页中提供一个 open 方法，可能是通过在每个 Web 页上包括一个 JavaScript 文件来初始化设置。有关如何实现此操作的示例，请参见示例文件 *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\main.htm* 和 *samples-dir\UltraLiteForMBusinessAnywhere\Simple\main_page.htm*。

跨页重新打开连接的要求为 UltraLite 应用程序提供了一种安全功能。可以用它来要求用户在页间移动时确认一些信息（多半为口令）。

如果在另一 Web 页中不需要 UltraLite 对象，应用程序应对该对象上发出 close 方法以节省内存。

另请参见

- “reOpenConnection 方法”一节第 76 页
- “PreparedStatement 类”一节第 86 页
- “ResultSet 类”一节第 96 页
- “ULTable 类”一节第 152 页
- “PreparedStatement 类”一节第 86 页

M-Business Anywhere 应用程序中的永久名称

在 HTML 中，当控制转移到新页时，旧页的已分配 JavaScript 对象的所有句柄都会丢失。例如，在 main.html 中，您拥有一个 M-Business Anywhere 数据库连接对象：

```
conn = dbMgr.openConnection("...");
```

如果您在 *main.html* 中单击一个链接并且它将您带到一个不同的页（例如：*insert.html*），那么，您在 *insert.html* 中找不到名为 "conn" 的对象。要取回连接对象，您可能需要再次调用 dbMgr.openConnection("...")。但是，您不必执行此操作，因为连接对象仍在内存中，您仅仅丢失了它的 JavaScript 句柄。

这就是为什么在对 DataManager、Connection、ULTable、PreparedStatement 或 ResultSet 的所有 M-Business Anywhere API 调用中都有一个 persistName 参数的原因。例如，当 M-Business Anywhere 运行库接收到来自 JavaScript 对 UltraLite 连接对象的调用时，M-Business Anywhere 首先检查具有相同 persistName 的连接对象是否存在于内存中。如果运行时能够找到匹配的对象，它将返回该连接对象。否则，M-Business Anywhere 执行常规过程创建一个新的 UltraLite 数据库连接并返回它。

使用永久名称

在 M-Business Anywhere 对象中有两种类型的层次。它们都从 DatabaseManager 和 Connection 开始：

- DatabaseManager -> Connection -> Table（用于表 API）
- DatabaseManager -> Connection -> PreparedStatement -> ResultSet（用于动态 SQL API）

要用永久名称检索这些 M-Business Anywhere 对象中的任何一个，您需要先用永久名称检索顶级对象，然后再沿着层次树检索所有上级 M-Business Anywhere 对象，直到检索到您想要的那一个。

例如，如果您想从 insert.html 中检索一个现有的 ULTable 对象，则需要在 *main.html* 中将永久名称赋给 dbMgr、conn 和 table 对象，然后在 *insert.html* 中使用永久名称将它们全部取回：

main.html 的代码段：

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here. A real database manager object is
allocated

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here. A real database connection is
made.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// a real table is allocated
```

insert.html 的代码段：

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here.
// The allocated database manager object from main.html is returned

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here.
// The existing connection object from memory is returned.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// the existing table object is returned.
```



```
var newTable = conn.getTable( "ULOrder", "simpleOrderTable" );
// since there is no order table from main.html,
// it does not exist in memory. A real order table object is allocated.
```

正确使用永久名称

将常用代码置于一个 JavaScript 文件中。既然 M-Business Anywhere 应用程序中大部分的 HTML 页都需要引用 DatabaseManager、Connection 和一些主要的 ULTable 对象，那么将创建它们（或用永久名称检索它们）的代码置于一个公用 JavaScript 文件中并在使用它们的 HTML 页的顶部包含该文件是很方便的。 M-Business Anywhere "simple" 和 "CustDB" 示例程序都演示了如何实现此操作。

如果您不打算在另一页上使用对象，请关闭该对象。如果 M-Business Anywhere 应用程序只有一个 HTML 页，那么就没有必要拥有永久名称。永久名称参数可以设置为 NULL。另一方面，如果每个 HTML 页都有许多打开的 PreparedStatement 和 ResultSet 对象，那么开发人员需要在以下两点之间取得平衡：在内存中保存这些对象以使用永久名称从另一 html 页中容易地检索它们的方便性，与由于这些对象始终到处存在而浪费的内存使用量。例如，假设您拥有在 *main.html* 中创建的 5 个 PreparedStatement 对象和 10 个 ResultSet 对象。它们占用了相当大的内存空间。当应用程序跳转到 *insert.html* 时，如果您只需用永久名称引用其中的一些对象，则那些不再需要的对象就会浪费内存。如果您想在 *insert.html* 中创建新的 PreparedStatement 和 ResultSet 对象，则可能会使内存不足。解决方案就是在 *main.html* 结尾处显式关闭那些 PreparedStatement 对象或 ResultSet 对象，如果您确定在 *insert.html* 中不再需要这些对象。

当用永久名称检索 M-Business Anywhere 对象时，将保持每一个 M-Business Anywhere 对象的状态。如果您在第 1 页中拥有永久的 ULTable 对象，则当您使用同一永久名称在第 2 页中调用 openTable 方法时，就会取回与第 1 页中对象状态相同的同一 ULtable 对象。如果当您离开第 1 页时，游标处于表的第 n 行，那么当您在第 2 页中将它取回时，游标仍在第 n 行。游标不会是 "在第一行之前"。

对 ResultSet 要谨慎使用永久名称。当在 PreparedStatement 中有占位符时，在是否赋予 ResultSet 一个永久名称上需要特别谨慎。例如，在 *main.html* 中，您拥有以下代码：

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

然后您在 *insert.html* 中需要同一个 ResultSet 对象，则必须执行以下操作：

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

//OrderStmt.setInt(1, 5000); // no need to do this since both the OrderStmt
and
OrderResultSet are retrieve from "cache" without any SQL statement being
actually executed

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

此 OrderResultSet 对象包含与设置为 5000 的 "order_id" 相同的结果。

但是，请考虑一种不同的情况。因为您想要对 Order 表执行同一查询，所以您需要相同的 PreparedStatement。但是您想要使用不是 5000 的订单 ID 进行查询。这种情况下，您可以为

PreparedStatement 指派一个永久名称，但是 ResultSet 不需要永久名称。既然这次的订单 ID 不同，那么结果集也与前一个有所不同。在 *main.html* 中，仍需要执行以下操作：

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // with persistent name

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( null ); // notice here, no
persistent name
```

在 *insert.html* 中，执行以下操作以获取一个新的 ResultSet：

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // get the prepared statement from memory with
persistent name

OrderStmt.setInt(1, 6000); // set a different place holder value

var OrderResultSet = OrderStmt.executeQuery( null ); // a real query is
executed
here!
```

在以上示例中，因为占位符值不同或者对 Order 表执行了一些预期返回的结果集将不同的其它操作，所以在调用 executeQuery 时对 ResultSet 不使用永久名称。

数据库加密和模糊处理

可以使用 UltraLite for M-Business Anywhere 对 UltraLite 数据库进行加密或模糊处理。

有关数据库加密的详细信息，请参见“保护 UltraLite 数据库”一节 《UltraLite - 数据库管理和参考》。

加密

UltraLite 数据库可以解密，也可以进行加密或模糊处理。如果想要对数据库进行加密或模糊处理，则这种选择必须在数据库创建时进行。

UltraLite 数据库的加密使用极强的行业标准技术加密数据库中的数据。该加密是以在数据库创建时指定的密钥短语为基础的。还必须在连接到数据库时提供此密钥短语。

如果加密 UltraLite 数据库，所有到该数据库的连接必须指定正确的加密密钥，否则连接尝试会失败。

有关 EncryptionKey 属性的详细信息，请参见“ConnectionParms 类”一节第 70 页 和“changeEncryptionKey 方法”一节第 57 页。

模糊处理

模糊处理是一种非常弱的加密形式，它只是简单地掩盖数据库中的数据以阻碍通过文件或磁盘查看器程序随意查看数据库内容。要对数据库进行模糊处理，请将 creationParms.obfuscate 布尔值设置为 true。例如：

```
var create_parms = dbMgr.createCreationParms();
create_parms.obfuscate = true;
```

示例

您可以通过在 Connection 对象上指定一个新加密密钥来更改加密密钥。在调用 changeEncryptionKey 方法前，应用程序必须使用现有加密密钥连接加密的数据库。在以下示例代码中，"apricot" 是新加密密钥。

```
conn.changeEncryptionKey("apricot")
```


使用 SQL 处理数据

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

有关 Table API 的信息，请参见“使用 Table API 处理数据”一节第 20 页。

本节讲解如何使用 SQL 执行以下任务。

- 插入、删除和更新行。
- 执行查询。
- 滚动浏览结果集中的行。

本节不介绍 SQL 语言本身。有关 SQL 功能的详细信息，请参见 SQL Anywhere 服务器 - SQL 参考。

数据操作：INSERT、UPDATE 和 DELETE

使用 UltraLite 可以执行 SQL 数据操作语言操作和 DDL 操作。这些操作是使用 ExecuteStatement 方法（PreparedStatement 类的一个成员）执行的。

有关 PreparedStatement 类的详细信息，请参见“PreparedStatement 类”一节第 86 页。

预准备语句中的参数标记
UltraLite 使用 ? 参数标记处理变量值。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 对象。

```
var PrepStmt;
```

2. 给预准备语句对象指派 INSERT 语句。在以下代码中，TableName 和 ColumnName 是表和列的名称。

```
PrepStmt = conn.prepareStatement(
    "INSERT into TableName(ColumnName) values (?)", null );
```

空参数指示该语句没有永久名称。

3. 为该语句指派参数值。

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
```

4. 执行该语句。

```
PrepStmt.executeStatement( null );
```


♦ **更新一行**

1. 声明 PreparedStatement 对象。

```
var PrepStmt;
```

2. 给预准备语句对象指派 UPDATE 语句。在以下代码中，TableName 和 ColumnName 是表和列的名称。

```
PrepStmt = conn.prepareStatement(
  "UPDATE TableName SET ColumnName = ? WHERE ID = ?", null);
```

空参数指示该语句没有永久名称。

3. 使用适合于数据类型的方法为该语句指派参数值。

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
PrepStmt.setIntParameter(2, 6);
```

4. 执行该语句

```
PrepStmt.executeStatement( );
```

♦ **删除一行**

1. 声明 PreparedStatement 对象。

```
var PrepStmt;
```

2. 给预准备语句对象指派 DELETE 语句。

```
PrepStmt = conn.prepareStatement(
    "DELETE FROM customer WHERE ID = ?", null );
```

空参数指示该语句没有永久名称。

3. 为该语句指派参数值。

```
var IDValue;
IDValue = 6;
PrepStmt.setIntParameter( 1, IDValue );
```

4. 执行该语句。

```
PrepStmt.executeStatement( );
```

数据检索：SELECT

执行 SELECT 语句时，PreparedStatement.executeQuery 方法返回一个 ResultSet 对象。ResultSet 类包含在结果集中导航的方法和使用 ResultSet 更新数据的方法。

有关 ResultSet 对象的详细信息，请参见“ResultSet 类”一节第 96 页。

示例

在以下代码中，以 ResultSet 的形式访问查询的结果。在第一次指派时，ResultSet 会放置在第一行之前。然后，调用 ResultSet.moveFirst 方法来导航到结果集中的第一条记录。

```
var MyResultSet;
var PrepStmt;
PrepStmt = conn.prepareStatement("SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

示例

以下代码演示了如何获取当前行的列值。该示例使用字符数据，类似的方法也可用于其它数据类型。

getString 方法使用以下语法：MyResultSetName.getString(*Index*)，其中 *Index* 是 SELECT 语句中列名的序号位置。

```
if ( MyResultSet.getRowCount() == 0 ) {
} else {
  alert( MyResultSet.getString(1) );
  alert( MyResultSet.getString(2) );
  MyResultSet.moveRelative(0);
}
```

有关导航结果集的详细信息，请参见“使用 SQL 导航”一节第 19 页。

以下过程使用 SELECT 语句来检索数据库中的信息。查询的结果会被指派给 ResultSet 对象。

♦ 执行 select 语句

1. 声明 PreparedStatement 对象。

   ```
   var OrderStmt;
   ```

2. 给您的 PreparedStatement 对象指派预准备语句。

   ```
   OrderStmt = Connection.prepareStatement(
      "SELECT order_id, disc, quant, notes, status, c.cust_id,
       cust_name, p.prod_id, prod_name, price
     FROM ULOrder o, ULCustomer c, ULProduct p
     WHERE o.cust_id = c.cust_id
     AND o.prod_id = p.prod_id
     ORDER BY der_id", "order_query_stmt" );
   ```

   第二个参数是一个永久名称，它为跨页 JavaScript 对象提供了持久性。

3. 执行该查询。

   ```
   OrderResultSet = OrderStmt.executeQuery( "order_query" );
   ```

有关如何使用查询的详细信息，请参见 *samples-dir\UltraLiteForMBusinessAnywhere\CustDB\custdb.js* 中的 CustDB 示例代码。

使用 SQL 导航

UltraLite for M-Business Anywhere 为您执行各种各样的导航任务提供了许多在结果集中导航的方法。

使用 ResultSet 对象的以下方法，您可以在结果集中导航：

- **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** 相对于当前行移动一定数量的行。正索引值在结果集中向前移动，负索引值在结果集中向后移动，零不移动游标。如果要重新填充行缓冲区，可以使用零。

示例

以下代码段演示了如何使用 moveFirst 方法在结果集中导航。

```
PrepStmt = conn.prepareStatement(
    "SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

所有的 move 方法都采用这样一种方式。

有关这些导航方法的详细信息，请参见“ResultSet 类”一节第 96 页。

ResultSetSchema 对象

使用 ResultSet.schema 属性，您可以检索有关查询中列的信息。

以下示例演示了如何使用 ResultSetSchema 来捕获模式信息。

```
var I;
var MySchema = rs.schema ;
for ( I = 1; I <= MySchema.columnCount; I++) {
  colname = MySchema.getColumnName(I);
  coltype = MySchema.getColumnSQLType(colname).toString();
  alert ( colname + " " + coltype );
}
```


使用 Table API 处理数据

UltraLite 应用程序可以使用 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 SQL 的信息，请参见“使用 SQL 处理数据”一节第 16 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。

使用 Table API 导航

UltraLite for M-Business Anywhere 为您执行各种各样的导航任务提供了许多在表中导航的方法。

使用 ULTable 对象的以下方法，您可以在结果集中导航：

- **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** 相对于当前行移动一定数量的行。正索引值在表中向前移动，负索引值在表中向后移动，零不移动游标。如果要重新填充行缓冲区，可以使用零。

示例

以下代码打开 customer 表并在其行间滚动。然后，它会显示一个警告，其中包含每个客户的姓氏。

```
var tCustomer;
tCustomer = conn.getTable( "customer", null );
tCustomer.open();
tCustomer.moveBeforeFirst();
While (tCustomer.moveNext()) {
  alert( tCustomer.getString(3) );
}
```

指定索引

打开表对象时，应用程序可以访问表中的行。缺省情况下，这些行按主键值顺序访问，但您可以指定索引，以便以某种顺序访问行。

示例

以下代码会移动到按 ix_name 索引排序的 customer 表的第一行。

```
tCustomer = conn.getTable("customer", null );
tCustomer.openWithIndex("ix_name");
tCustomer.moveFirst();
```

访问当前行中的值

任何时候，ULTable 对象都位于以下位置之一。

- 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 ULTable 对象放置在行上，您可以使用 ULTable get 方法之一获取当前行每一列的值。

示例

以下代码段会从 tCustomer ULTable 对象中检索出三列的值，并在文本框中显示这些值。

```
var colID, colFirstName, colLastName;
colID = tCustomer.schema.getColumnID( "id" );
colFirstName = tCustomer.schema.getColumnID( "fname" );
colLastName = tCustomer.schema.getColumnID( "lname" );
alert( tCustomer.getInt( colID ) );
alert( tCustomer.getString( colFirstName ) );
alert( tCustomer.getString( colLastName ) );
```

也可以使用 ULTable 的方法来设置值。

```
tCustomer.setString( colLastName, "Kaminski" );
```

通过将值指派给这些属性，您不用在数据库中变更数据的值。

即使是在该表的第一行之前或最后一行之后，您也可以给属性指派值。但是，您不能从列中获取值。例如，以下代码段会生成一个错误。

```
tCustomer.moveBeforeFirst();
id = tCustomer.getInt( colID );
```

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，查找和查询两种模式用于搜索。ULTable 对象具有对应这些模式的方法，可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 ULTable 对象时指定的排序顺序，移动到第一个与指定的搜索值完全匹配的行。

  有关 Find 方法的详细信息，请参见“ULTable 类”一节第 152 页。

- **Lookup 方法** 按照打开 ULTable 对象时指定的排序顺序，移动到匹配或大于指定的搜索值的第一行。

  有关 lookup 方法的详细信息，请参见“ULTable 类”一节第 152 页。

♦ 搜索行

1. 进入查找或查询模式。

   调用 FindBegin 方法或 LookupBegin 方法。例如，以下代码段调用 ULTable.findBegin。

   ```
   tCustomer.findBegin();
   ```

2. 设置搜索值。

   可以通过设置当前行中的值来完成设置。设置这些值会影响缓冲区，而不影响数据库。例如，以下代码段将缓冲区中的姓氏列设置为 Kaminski。

   ```
   tCustomer.setString(3, "Kaminski" );
   ```

   对于多列索引，第一列的值是必需的，但可以忽略其它列。

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。

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

插入、更新和删除行

UltraLite 每次向应用程序提供表中的一行。ULTable 对象有一个当前位置，它可能在表中的某一行上、第一行前面或最后一行后面。

当您的应用程序改变位置时，UltraLite 会在缓冲区中制作一个该行的副本。获取或设置值的任何操作都只影响该缓冲区中的数据副本，而不会影响数据库中的数据。

示例

以下语句将缓冲区中第一列的值更改为 3。

```
tCustomer.setInt( 1 , 3 );
```

使用 UltraLite 模式

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

- **插入模式** 当调用 ULTable.insert 方法时，缓冲区中的数据将作为新行添加到表中。

- **更新模式** 当调用 ULTable.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 ULTable.find 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** 在调用某一种 ULTable.lookup 方法时，用于定位其值匹配或大于缓冲区中的数据的行。

♦ 更新一行

1. 移动到想要更新的行。

   您可以通过滚动浏览表来移动到某一行，也可以使用 Find 方法和 Lookup 方法通过搜索来移动到该行。

2. 进入更新模式。

   例如，以下指令进入表 tCustomer 的更新模式。

   ```
   tCustomer.updateBegin();
   ```

3. 为要更新的行设置新值。

   例如，以下指令将新值设置为 Elizabeth。

   ```
   tCustomer.setString( 2, "Elizabeth" );
   ```

4. 执行更新。

   ```
   tCustomer.update();
   ```

完成更新操作后，当前行就是刚才更新过的行。如果您更改了在打开 ULTable 对象时指定的索引中的列的值，则当前位置不确定。

缺省情况下，UltraLite 在 autoCommit 模式中运行，所以更新会被立即应用于永久存储中的行。如果已经禁用了 autoCommit 模式，那么只有在执行提交操作后才会应用更新。有关 autoCommit 模式的详细信息，请参见“管理事务”一节第 24 页。

> **小心**
> 不要更新行的主键；而是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类似，区别在于不需要在执行插入操作前定位到表中的任何特定行。行会按打开表时指定的索引自动排序。

♦ 插入一行

1. 进入插入模式。

   例如，以下指令进入表 CustomerTable 的插入模式。

   ```
   tCustomer.insertBegin();
   ```

2. 设置新行的值。

   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，则使用 NULL。如果该列不允许 NULL 值，则使用以下缺省值：

- 对于数字列，添加零。
- 对于字符列，添加空字符串。

若要显式设置值为 NULL，请使用 setNull 方法。

```
colID = tCustomer.schema.getColumnID( "id" );
colFirstName = tCustomer.schema.getColumnID( "fname" );
colLastName = tCustomer.schema.getColumnID( "lname" );
tCustomer.setInt( colID, 42 );
tCustomer.setString( colFirstName, "Mitch" );
tCustomer.setString( colLastName, "McLeod" );
```

3. 执行该插入。

   执行 Commit 后，插入的行就会永久保存到数据库中。在 autoCommit 模式中，执行 Insert 方法本身就包含了 Commit 操作。

   ```
   tCustomer.insert();
   ```

删除行

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. 执行删除：

   ```
   tCustomer.deleteRow();
   ```

处理 BLOB 数据

可以使用 GetBytes 或 GetBytesSection 方法来为声明为 BINARY 或 LONG BINARY 的列读取 BLOB 数据。

请参见“getBytes 方法”一节第 158 页和“getBytesSection 方法”一节第 98 页。

管理事务

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，UltraLite for M-Business Anywhere 在 autoCommit 模式下运行。在 autoCommit 模式中，每次插入、更新或删除都会作为单独的事务来执行。一旦操作完成后，也就完成了对数据库的更改。

如果将 Connection.autoCommit 属性设置为 false，那么可以使用多语句事务。例如，如果您的应用程序在两个帐户之间转移资金，从源帐户减除和添加到目标帐户就构成了一次事务。

如果 autoCommit 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，否则可执行 Connection.rollback() 语句取消事务的所有操作。关闭 autoCommit 可以改善性能。

注意
同步会导致 commit，即使您将 autoCommit 设置为 false 也是如此。

访问模式信息

每个 Connection、ULTable 和 ResultSet 对象都包含一个模式属性。这些模式对象提供了有关数据库中表、列、索引和发布的信息。

- **DatabaseSchema** 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。

  若要获取 DatabaseSchema 对象，请访问 Connection.databaseSchema 属性。

- **TableSchema** 表中的列的编号和名称，以及该表的索引集合。

  若要获取 TableSchema 对象，请访问 ULTable.schema 属性。

- **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，所以没有单独的 Index 对象，而只有一个 IndexSchema 对象。

  使用 TableSchema.getIndex 方法可访问 IndexSchema 对象。

- **PublicationSchema** 发布中包含的表和列的编号和名称。发布包括 PublicationSchema 对象，但不包括 Publication 对象。

  使用 DatabaseSchema.getPublicationSchema 方法可访问 PublicationSchema 对象。

- **ResultSetSchema** 结果集中列的编号和名称。

  使用 PreparedStatement.getResultSetSchema 方法或 ResultSet.schema 属性可访问 ResultSetSchema 对象。

处理错误

在正常运行情况下，UltraLite for M-Business Anywhere 可以抛出旨在脚本环境中捕获和处理的错误。请参见“SQLError 类”一节第 117 页。

错误会表示为 SQLCODE 值，用负数来表明错误的具体种类。

UltraLite for M-Business Anywhere 仅从 DatabaseManager 和 Connection 对象中抛出错误。以下这些 DatabaseManager 的方法可以抛出错误：

- createDatabase
- dropDatabase
- openConnection

UltraLite for M-Business Anywhere 中的所有其它错误和异常都会通过 Connection 对象来路由。

您可以从 DatabaseManager 和 Connection 对象中访问错误号。请参见：

- “Connection 类”一节第 56 页
- “DatabaseManager 类”一节第 74 页

验证用户

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，因此必须首先以此初始用户的身份进行连接。

不能更改用户 ID：添加一个用户并删除现有用户。每个 UltraLite 数据库最多允许使用四个用户 ID。

有关授予或撤消连接权限的详细信息，请参见“grantConnectTo 方法”一节第 64 页和“revokeConnectFrom 方法”一节第 66 页。

♦ 添加用户或更改现有用户的口令

1. 以现有用户身份连接到数据库。
2. 授予用户具有所需口令的连接权限。

```
conn.grantConnectTo("Robert", "newPassword");
```

♦ 删除现有用户

1. 以现有用户身份连接到数据库。
2. 像下面这样撤消用户的连接权限。

```
conn.revokeConnectFrom("Robert");
```


同步数据

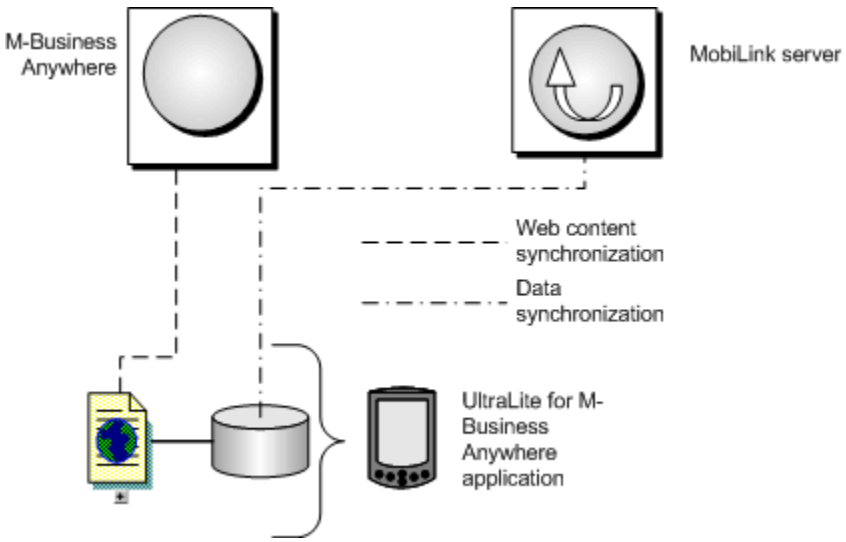
UltraLite for M-Business Anywhere 应用程序通常涉及两种同步：

- **Web 内容同步** Web 内容，包括定义应用程序本身的 HTML 页，都是通过 M-Business Anywhere 来同步的。
- **数据同步** UltraLite 数据库与 MobiLink 服务器同步。

虽然这两种同步是不同的，但可以用名为 **one-button synchronization** 的技术将其一起启动。虽然对大部分应用程序都建议使用一键同步模型，但也可能有一些情况需要保持数据和 Web 内容的同步完全独立，以下将对该技术进行讨论。

一键同步

一键同步是一种在单个操作中启动 Web 内容同步（使用 M-Business Anywhere）和 UltraLite 数据同步（使用 MobiLink）的技术。它仅在 Windows Mobile 和 Windows 上可用。一键同步的体系结构如下所示：



一键同步中事件的顺序如下所示：

1. 用户可能通过将其 Web 应用程序置于底座中来同步它们。
2. M-Business 客户端同步 Web 内容。
3. M-Business 客户端的 MBConnect 组件调用 *ulconnect.exe* 应用程序。
4. *ulconnect.exe* 启动 UltraLite 数据库同步。
5. 数据与 MobiLink 同步。

要实现一键同步，您必须执行以下步骤：

1. 在您的应用程序中，为 MobiLink 同步设置同步参数。

   如果您通过 M-Business Anywhere 进行同步，可以使用 SyncParms.setMBAServer 方法设置主机和端口同步参数。请参见“setMBAServer 方法”一节第 133 页。

   否则，使用标准方法设置同步参数。请参见“SyncParms 类”一节第 128 页。

2. 保存同步参数以使它们能被 *ulconnect.exe* 读取。

   调用 Connection.saveSyncParms 方法来保存同步参数。请参见“saveSyncParms 方法”一节第 66 页。

同步数据

对于大部分的用户来说，使用一键同步是很有用的，它同时启动了数据同步和 Web 内容同步。有关详细信息，请参见“一键同步”一节第 29 页。

本节是针对那些想要将数据同步和 Web 内容同步单独进行的用户的。

同步要求有 MobiLink 服务器和相应的许可。可以参见 CustDB 示例应用程序中的同步工作示例。

UltraLite for M-Business Anywhere 支持 TCP/IP、HTTP、HTTPS 和 HotSync 同步。同步由 UltraLite 应用程序启动。在所有情况中，您都可以使用 Connection 对象的方法和属性来控制同步。

需要单独授予许可的组成部分
ECC 加密和 FIPS 认证的加密需要单独的许可。所有高度加密技术受出口法规约束。

请参见“单独授权的组件”一节 《SQL Anywhere 11 - 简介》。

♦ 通过 TCP/IP 或 HTTP 进行同步

1. 准备同步信息。

   为 Connection.syncParms 对象的必需的属性指派值。

   有关您应该设置的属性和值的详细信息，请参见“UltraLite 客户端” 《UltraLite - 数据库管理和参考》。

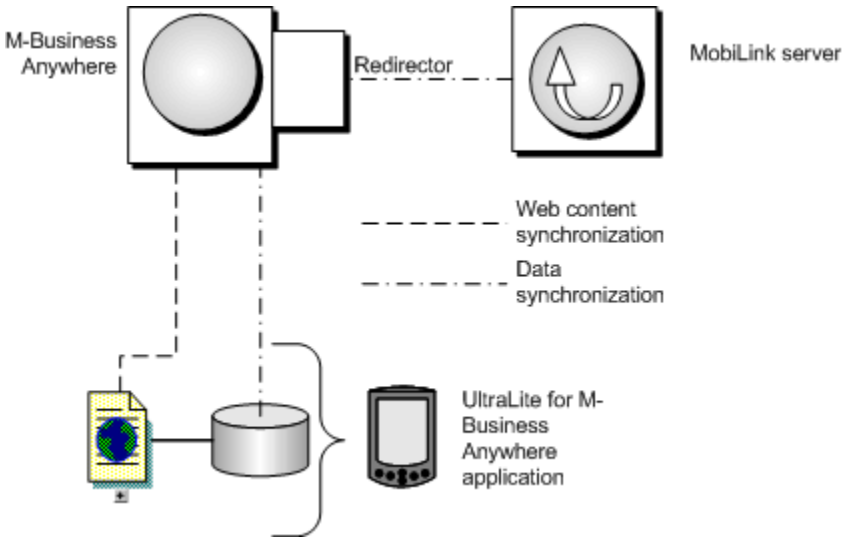
2. 执行同步。

   调用 Connection.synchronize 方法。

通过 M-Business Anywhere 同步数据

不论您是使用一键同步还是单独的数据同步，您都可以使用 MobiLink 重定向器来配置 M-Business Anywhere 服务器将数据路由到 MobiLink 服务器或从 MobiLink 服务器路由数据。对于从防火墙外部进行的同步，这样可以减少所需的外部可访问端口数。

下图显示了一键同步情况下的体系结构。



♦ **通过 M-Business Anywhere 同步数据**

1. 在服务器端，建立一个 MobiLink 重定向器在 M-Business Anywhere 和您的 MobiLink 服务器之间路由数据。请参见“M-Business Anywhere 重定向器（不建议使用）”一节 《MobiLink - 服务器管理》。
2. 在您的客户端，设置同步参数以使 UltraLite 同步定向到 M-Business Anywhere 的主机和端口号。您可以使用 SyncParms.setMBAServer 方法来执行此任务。请参见“setMBAServer 方法”一节 第 133 页。
3. 从客户端应用程序，使用一键同步或单独的数据同步来启动同步。请参见：
   - “一键同步”一节第 29 页
   - “同步数据”一节第 30 页

部署 UltraLite for M-Business Anywhere 应用程序

您在完成了应用程序或想要测试应用程序时，需要将它部署到一个设备上。这一节概要介绍了将 UltraLite 应用程序部署到设备所需的步骤。

将应用程序部署到 Windows Mobile 和 Windows 桌面操作系统

要将 UltraLite 应用程序部署到 Windows Mobile 设备上，必须执行以下步骤：

- 部署应用程序和 UltraLite 组件。请参见“UltraLite for M-Business Anywhere 快速入门”一节 第 6 页。
- 部署 UltraLite 数据库的初始副本。请参见“UltraLite for M-Business Anywhere 快速入门”一节 第 6 页。

  在许多情况下，部署 UltraLite 数据库就足够了。您可以使用同步装载数据的初始副本。

  必须将数据库放在应用程序可以找到的位置。Database On CE 连接参数为 Windows Mobile 定义了该位置。Database on Desktop 连接参数为 Windows 定义了该位置。请参见：

  - “UltraLite CE_FILE 连接参数”一节 《UltraLite - 数据库管理和参考》
  - “UltraLite DBF 连接参数”一节 《UltraLite - 数据库管理和参考》

部署使用一键同步的应用程序

一键同步需要一组文件，包括：*ulconnect.exe*、*ulconnect.udb*、*ulpod11.dll* 和 *ulrt11.dll*。对于 Windows Mobile，这些文件位于目录 *install-dirralite\UltraLiteForMBusinessAnywhere\ce\arm* 中的 *ulpod.cab* 文件中。当您将 cab 文件部署到 Windows Mobile 设备上时，它会自动将其内容安装到正确的位置。对于 Windows，所需的文件必须从目录 *install-dir\ultralite\UltraLiteForMBusinessAnywhere\win32\386* 中手工部署。

将应用程序部署到 Palm OS

要将 UltraLite 应用程序部署到 Palm OS 设备上，您必须执行以下步骤：

- 部署应用程序和 UltraLite 组件。请参见“UltraLite for M-Business Anywhere 快速入门”一节 第 6 页。
- 部署 UltraLite 数据库的初始副本。请参见“UltraLite for M-Business Anywhere 快速入门”一节 第 6 页。

  在许多情况下，只部署一个正确初始化的 UltraLite 数据库文件就足够了。然后，您就可以使用同步加载数据的初始副本。

  可以创建 *.pdb* 文件，以便利用任意 UltraLite 实用程序（包括 ulxml 和 ulinit）将其部署到 Palm OS。

必须使用正确的创建者 ID 提供一个数据库，以便您的应用程序能够找到它。Database On Palm 连接参数使用该创建者 ID 来查找数据库。请参见“UltraLite PALM_FILE 连接参数”一节《UltraLite - 数据库管理和参考》。

- 部署用于 HotSync 的 MobiLink 同步管道。

  只有在应用程序使用 HotSync 进行同步的情况下，才需要执行这一步。请参见“Palm OS 上的 HotSync”一节 《UltraLite - 数据库管理和参考》。

教程：M-Business Anywhere 示例应用程序

目录

M-Business Anywhere 开发教程简介 36
第 1 课：建立数据库 37
第 2 课：创建应用程序文件 39
第 3 课：配置 M-Business Anywhere 40
第 4 课：向应用程序添加启动代码 41
第 5 课：添加数据操作和导航 43
第 6 课：将导航添加到应用程序 46
第 7 课：将同步添加到应用程序 47
main.htm 和 tutorial.js 的列表 48

M-Business Anywhere 开发教程简介

本教程介绍如何构建跨平台 UltraLite for M-Business Anywhere 应用程序。此应用程序还可以通过统一数据库进行同步。

计时

如果您复制并粘贴代码，则此教程需要大约 60 分钟时间来完成。在本教程末尾的“main.htm 和 tutorial.js 的列表”一节第 48 页中提供有关 *main.htm* 和 *tutorial.js* 的完整的代码示例。

前提条件

本教程假定您基本了解 JavaScript 编程、在 M-Business Anywhere 环境中开发移动应用程序以及使用 Sybase Central 管理数据库。

另请参见

- “使用 [创建数据库向导] 创建数据库”一节 《UltraLite - 数据库管理和参考》
- “创建和配置 UltraLite 数据库” 《UltraLite - 数据库管理和参考》

第 1 课：建立数据库

本课介绍如何为教程创建远程数据库以及如何将同步模型部署到统一数据库。

♦ 建立数据库

1. 为本教程创建一个目录。本教程假定目录为 *c:\tutorial*。如果您创建具有不同名称的目录，请在整个教程中都使用该目录。
2. 使用以下信息通过 Sybase Central 创建 UltraLite 数据库。（有关创建远程数据库的详细信息，请参见“使用 [创建数据库向导] 创建数据库”一节 《UltraLite - 数据库管理和参考》 。）
   - **表名** Customer
   - **列**

列名	数据类型（大小）	列是否允许 NULL 值？	缺省值
ID	integer	否	autoincrement
GivenName	char(15)	否	无
Surname	char(20)	否	无
City	char(20)	是	无
Phone	char(12)	是	无
Street	char(50)	否	无

3. 为以下平台保存远程数据库文件：
   - **Windows** *c:\tutorial\WIN32_OS\tutorial.udb*
   - **Windows Mobile** *c:\tutorial\WIN32_CE\tutorial.udb*
   - **Palm** *c:\tutorial\PALM_OS\tutorial.udb*

♦ 部署同步模型

1. 从 Sybase Central 创建同步模型。有关创建同步模型的详细信息，请参见“创建模型”一节 《MobiLink - 入门》 。

   使用 *c:\tutorial* 作为同步模型文件的工作文件夹。对于 [**统一数据库模式**] 页面和 [**远程数据库模式**] 页面使用以下设置：

   a. 对于 [**统一数据库模式**]，使用 SQL Anywhere 示例数据库 *demo.db* 获取模式。

   b. 对于 [**远程数据库模式**]，使用 UltraLite 示例数据库 *tutorial.udb*（或 *.pdb*）获取模式。

   c. 为向导的所有其它页面使用缺省设置。

2. 从 Sybase Central 部署同步模型。有关部署同步模型的详细信息，请参见“部署模型”一节《MobiLink - 入门》。
   a. 对于 [**统一数据库部署目标**] 页面，使用 SQL Anywhere 示例数据库作为统一数据库。
   b. 为 [**远程数据库部署**] 页面选择 [**现有 SQL Anywhere 或 UltraLite 数据库**] 选项将同步模型部署到 UltraLite 数据库 *tutorial.udb*（或 *.pdb*）。
   c. 对于 [**现有远程数据库**] 页面，取消选中 [**连接到远程数据库以便直接应用更改**] 选项。
   d. 对于 [**MobiLink 用户**] 页面，指定以下用于连接到 MobiLink 服务器的设置：
      - **用户名** 教程
      - **口令** 教程
   e. 为向导的所有其它页面使用缺省设置。

当完成 [**部署同步模型向导**] 时应该会生成命令文件 *tutorial_mlsrv.bat*。此命令文件将用于教程后面的内容中。

第 2 课：创建应用程序文件

本课介绍如何建立应用程序文件。

♦ 创建应用程序文件

1. 创建文件 *c:\tutorial\main.htm*。

   在本教程后面的内容中，将向 *main.htm* 添加更多逻辑。目前，您只需设置此文件，以包括一个特定于平台的文件 *ul_deps.html*。

   将以下内容添加到 *main.htm*：

   ```
   <html>
   <body>
   <a href="AG_DEVICEOS/ul_deps.html"></a>
   </body>
   </html>
   ```

2. 创建特定于平台的文件 *ul_deps.html*。

   此文件为不同的操作系统引用特定的二进制，如下所示：

   - **Windows** *c:\tutorial\WIN32_OS\ul-deps.htm*

     ```
     <!-- WIN32_OS\ul_deps.html -->
     <html>
       <a href="ulpod11.dll"></a>
       <a href="tutCustomer.udb"></a>
     </html>
     ```

   - **Windows Mobile** *c:\tutorial\WINCE_OS\ul-deps.htm*

     ```
     <!-- WINCE_OS\ul_deps.html -->
     <html>
       <a href="AG_DEVICEPROCESSOR/ulpod11.dll"></a>
       <a href="tutCustomer.udb"></a>
     </html>
     ```

   - **Palm** *c:\tutorial\PALM_OS\ul-deps.htm*

     ```
     <!-- PALM_OS\ul_deps.html -->
     <html>
       <a href="ulpod11.prc"></a>
       <a href="tutCustomer.pdb"></a>
     </html>
     ```

3. 对于 Windows 和 Windows Mobile 将 UltraLite Pod 文件 *ulpod11.dll*，或者对于 Palm 将 *.prc* 复制到 *tutorial* 目录。

   - 对于 Windows 桌面操作系统，将 *ulpod11.dll* 从 *install-dir\UltraLite\UltraLiteForMBusinessAnywhere\win32\386* 复制到 *c:\tutorial\WIN32_OS*。
   - 对于 Windows Mobile，将 *ulpod11.dll* 从 *install-dir\UltraLite\UltraLiteForMBusinessAnywhere\CE\Arm* 复制到 *c:\tutorial\WINCE_OS\arm*。
   - 对于 Palm OS，将 *ulpod11.prc* 从 *install-dir\UltraLite\UltraLiteForMBusinessAnywhere\Palm\68k* 复制到 *c:\tutorial\PALM_OS*。

   现在已将所有应用程序文件放在适当位置。

第 3 课：配置 M-Business Anywhere

本课介绍如何配置 M-Business Anywhere 和同步 UltraLite 教程通道。

♦ 配置 M-Business Anywhere

1. 打开 M-Business Anywhere 管理控制台并作为 **Admin** 登录（无需口令）。
2. 创建一个名为 *tutorial* 的新用户。
3. 为该用户创建一个通道：
   a. 对此通道使用以下设置。确保为 Web 服务器替换正确的 URL：
      - **Channel Title** UltraLite 教程。
      - **Channel URL** *http://<yourwebserver>/tutorial/main.htm*.

        此位置是您的 Web 服务器提供的教程 *main.htm* 页面的 URL。
      - **Channel Size** 1000 KB。
      - **Channel Link Depth** 3.
      - **Allow Binary Distribution** 真（选中）。
      - **Hide From Users** 假（未选中）。

现在，在 M-Business Anywhere 中已设置了用户和通道。下一步是将此通道的内容与 M-Business 客户端同步。您可以从要使用的任何一个平台完成此操作。

以下过程假定您已安装了 M-Business 客户端。建议您单击 [**Tools**] » [**Options**]，然后将客户端选项设置为 [**Show JavaScript Errors**]。此设置使调试应用程序中的错误更容易。

♦ 同步设备的通道

- 同步设备的 UltraLite 通道。

  在此阶段中，没有应用程序内容，所以页面显示为空白。

有关 M-Business Anywhere 环境的详细信息，请参见 *M-Business Anywhere 应用程序开发人员指南*。

第 4 课：向应用程序添加启动代码

本课提供了连接到 UltraLite 数据库的应用程序的启动代码。

♦ 将内容添加到应用程序

1. 将以下内容添加到紧靠 <a> 标记之前的 *main.htm*：

```
<form name="form">
<br><td> ID: </td>
    <td> <input type="text" name="ID" size="10"> </td>
<br><td> Given Name: </td>
    <td> <input type="text" name="GivenName" size="15">
    </td>
<br><td> Surname: </td>
    <td> <input type="text" name="Surname" size="50"> </td>
<br><td> Street: </td>
    <td> <input type="text" name="Street" size="20"> </td>
<br><td> City: </td>
    <td> <input type="text" name="City" size="20"> </td>
<br><td> Phone: </td>
    <td> <input type="text" name="Phone" size="12"> </td>
<br>
<br>

<table>
<tr>
    <td> <input type="button" value="Insert"
         onclick="ClickInsert();"> </td>
    <td> <input type="button" value="Next"
         onclick="ClickNext();"> </td>
    <td> <input type="button" value="Prev"
         onclick="ClickPrev();"> </td>
</tr>
<tr>
    <td colspan=3>
    <input type="button" value="Synchronize"
    onclick="ClickSync();">
    </td>
</tr>
</table>
</form>
```

2. 创建一个 JavaScript 文件 *c:\tutorial\tutorial.js* 来提供应用程序逻辑。
3. 为 UltraLite Pod 对象将以下变量声明添加到 *tutorial.js*：

```
var DB_mgr;
var Connection;
var Table;
```

4. 将以下函数添加到 *tutorial.js* 以连接到教程数据库：

```
function Connect()
{
    var     dir;
    var     open_parms;
    var     browser = navigator.platform;

    DB_mgr =
CreateObject( "iAnywhere.UltraLite.DatabaseManager.Tutorial" );
```

```
    if( DB_mgr == null ) {
     alert( "Error: cannot create database manager: " + DB_mgr.sqlCode );
return;
    }
    dir = DB_mgr.directory;
    if( browser == "Palm OS" ) {
open_parms = "con=tutorial;palm_file=tutorial"
    } else {
     open_parms = "con=tutorial;" + "file_name=" + dir + "\\tutorial.udb";
    }
    try {
Connection = DB_mgr.reOpenConnection( "tutorial" );
     if( Connection == null ) {
     Connection = DB_mgr.openConnection( open_parms );
}
    } catch( ex ) {
     if( DB_mgr.sqlCode !=
         DB_mgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
         alert( "Error: cannot connect to database: " + ex.getMessage() );
     return;
     }
}
```

5. 应用程序启动时使用 onload 事件处理程序连接到数据库。按如下方式修改 *main.htm*：

   a. 通过在紧靠 <body> 标记之前添加以下一行内容来装载 *tutorial.js*：

      ```
      <script src="tutorial.js"></script>
      ```

   b. 修改 <body> 标记：

      ```
      <body onload="Connect();">
      ```

6. 测试应用程序。

   同步 UltraLite 教程通道。同步应用程序应正连接到教程数据库

第 5 课：添加数据操作和导航

本课介绍了如何用数据操作和导航逻辑填充应用程序。

♦ 初始化表

1. 通过将以下代码添加到 *tutorial.js* 中的 **Connect** 函数的末尾来初始化数据库中表示 **Customer** 表的 CustomerTable：

```
try {
  CustomerTable = Connection.getTable( "customer", null );
  CustomerTable.open();
} catch( ex3 ) {
  alert("Error: " + ex3.getMessage() );
}
```

2. 添加变量以在数据库和 Web 窗体之间移动数据。

   为客户数据将以下变量声明添加到 *tutorial.js* 的顶端：

```
var GivenName = "";
var Surname = "";
var Street = "";
var City = "";
var Phone = "";
var ID = "";
```

3. 创建函数以读取和显示客户数据。

   将以下函数添加到 *tutorial.js*。此函数读取客户数据的当前行，还确保 NULL 列显示为空字符串：

```
function Fetch()
{
    if( Table.getRowCount() == 0 ) {
 GivenName = "";
 Surname = "";
 Street = "";
 City = "";
 Phone = "";
 ID = "";
 return;
    }
    ID = Table.getString( Table.schema.getColumnID( "ID" ) );
    GivenName =
Table.getString( Table.schema.getColumnID( "GivenName" ) );
    Surname = Table.getString( Table.schema.getColumnID( "Surname" ) );
    Street = Table.getString( Table.schema.getColumnID( "Street" ) );
    if( Table.isNull( Table.schema.getColumnID( "City" ) ) ) {
 City = "";
    } else {
 City = Table.getString( Table.schema.getColumnID( "City" ) );
    }
    if( Table.isNull( Table.schema.getColumnID( "Phone" ) ) ) {
 Phone = "";
    } else {
 Phone = Table.getString( Table.schema.getColumnID( "Phone" ) );
    }
}
```


将以下内容添加到 *main.htm* 中紧靠 <script> 标记之后。**DisplayRow** 从数据库获取值，然后在 Web 窗体中显示这些值。**FetchForm** 获取 Web 窗体中的当前值，然后提供给数据库代码使用。

```
<script>
function DisplayRow() {
    Fetch();
    document.form.ID.value = ID;
    document.form.GivenName.value = GivenName;
    document.form.Surname.value = Surname;
    document.form.Street.value = Street;
    document.form.City.value = City;
    document.form.Phone.value = Phone;
}

function FetchForm() {
    GivenName = document.form.GivenName.value;
    Surname = document.form.Surname.value;
    Street = document.form.Street.value;
    City = document.form.City.value;
    Phone = document.form.Phone.value;
}
</script>
```

4. 当应用程序已装载时，调用 **DisplayCurrentRow** 以显示当前行。按如下方法修改 *main.htm* 顶端的 body 标记：

```
<body onload="Connect(); DisplayCurrentRow();">
```

虽然直到此时教程数据库中都没有任何数据，但这是同步通道以确保应用程序可以正常运行的好时机。

♦ 将代码添加到插入行

- 创建函数以插入客户数据。

  在以下过程中，调用 **InsertBegin** 可以将应用程序置于插入模式下，并将当前行中的所有值设为其缺省值。例如，ID 列收到下一个自动增量值。设置列值，然后插入新行。

  将以下函数添加到 *tutorial.js*：

```
function Insert()
{
    try {
 Table.insertBegin();
 Table.setString( Table.schema.getColumnID( "GivenName" ), GivenName );
 Table.setString( Table.schema.getColumnID( "Surname" ), Surname );
 Table.setString( Table.schema.getColumnID( "Street" ), Street );
 if( City.length > 0 ) {
     Table.setString( Table.schema.getColumnID( "City" ), City );
 }
 if( Phone.length > 0 ) {
     Table.setString( Table.schema.getColumnID( "Phone" ), Phone );
 }
 Table.insert();
 Table.moveLast();
    } catch( ex ) {
 alert( "Error: cannot insert row: " + ex.getMessage() );
    }
}
```

  将以下函数添加到 *main.htm*：


```
function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayRow();
}
```

第 6 课：将导航添加到应用程序

本课介绍用于在 **Customer** 表的各行之间前移和后移的代码。

♦ 将导航代码添加到应用程序

1. 将 **Next** 函数添加到 *tutorial.js*：

```
function Next()
{
    if( ! Table.moveNext() ) {
 Table.moveLast();
    }
}
```

2. 将 **Prev** 函数添加到 *tutorial.js*：

```
function Prev()
{
    if( ! Table.movePrevious() ) {
 Table.moveFirst();
    }
}
```

3. 将以下函数添加到 *main.htm*：

```
function ClickNext()
{
    Next();
    DisplayRow();
}

function ClickPrev()
{
    Prev();
    DisplayRow();
}
```

4. 当窗体第一次显示时，由于当前位置是在第一行的前面，所以控件是空的。在窗体显示后，单击 [**Next**] 和 [**Previous**] 按钮可在表中的行之间移动。

第 7 课：将同步添加到应用程序

以下过程实现同步。

♦ 将同步函数添加到应用程序

1. 将 **Synchronize** 函数添加到 *tutorial.js*。

```
function Synchronize()
{
    var sync_parms;

    sync_parms = Connection.syncParms;
    sync_parms.setUserName( "tutorial" );
    sync_parms.setPassword( "tutorial" );
    sync_parms.setVersion( "tutorial" );
    sync_parms.setStream( sync_parms.STREAM_TYPE_TCPIP );
    try {
 Connection.synchronize();
    } catch( ex ) {
     alert( "Error: cannot synchronize: " + ex.getMessage() );
    }
}
```

   同步参数存储在 SyncParms 对象中。例如，SyncParms.userName 属性指定 MobiLink 搜索的用户名。SyncParms.sendColumnNames 属性指定发送到 MobiLink 用于生成上载和下载脚本的列名。

   此函数使用 TCP/IP 同步流和缺省网络通信选项（流参数）。这些缺省选项假定您的同步方式为以下两种中的一种：即通过 ActiveSync 从连接到运行 MobiLink 服务器的计算机的 Windows Mobile 客户端同步，或者从 MobiLink 所在计算机上运行的 32 位 Windows 桌面操作系统客户端同步。如果不是这种情况，更改同步流类型，并将网络通信选项设置为相应的值。

   另请参见：

   - “setStream 方法”一节第 136 页
   - “setStreamParms 方法”一节第 136 页

2. 将以下函数添加到 *main.htm*：

```
function ClickSync()
{
    Synchronize();
    DisplayRow();
}
```

3. 使用 *tutorial_mlsrv.bat* 文件启动 MobiLink 服务器。

现在应下载 **Customer** 表中的数据。您应该能够在远程数据库的 **Customer** 表中的各行之间移动。

至此，本教程结束。有关 *main.htm* 和 *tutorial.js* 的完整的代码示例，请参见“main.htm 和 tutorial.js 的列表”一节第 48 页。

main.htm 和 tutorial.js 的列表

以下是供您使用的 *main.htm* 的完整列表：

```
<html>
<script src="tutorial.js"></script>

<script>
function DisplayRow() {
    Fetch();
    document.form.ID.value = ID;
    document.form.GivenName.value = GivenName;
    document.form.Surname.value = Surname;
    document.form.Street.value = Street;
    document.form.City.value = City;
    document.form.Phone.value = Phone;
}

function FetchForm() {
    GivenName = document.form.GivenName.value;
    Surname = document.form.Surname.value;
    Street = document.form.Street.value;
    City = document.form.City.value;
    Phone = document.form.Phone.value;
}

function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayRow();
}

function ClickNext()
{
    Next();
    DisplayRow();
}

function ClickPrev()
{
    Prev();
    DisplayRow();
}

function ClickSync()
{
    Synchronize();
    DisplayRow();
}
</script>

<body onload="Connect(); DisplayRow();" >

<form name="form">
<br><td> ID: </td>
    <td> <input type="text" name="ID" size="10"> </td>
<br><td> Given Name: </td>
    <td> <input type="text" name="GivenName" size="15"> </td>
<br><td> Surname: </td>
    <td> <input type="text" name="Surname" size="50"> </td>
<br><td> Street: </td>
```



```
    <td> <input type="text" name="Street" size="20"> </td>
<br><td> City: </td>
    <td> <input type="text" name="City" size="20"> </td>
<br><td> Phone: </td>
    <td> <input type="text" name="Phone" size="12"> </td>
<br>
<br>

<table>
<tr>
    <td> <input type="button" value="Insert" onclick="ClickInsert();"> </td>
    <td> <input type="button" value="Next" onclick="ClickNext();"> </td>
    <td> <input type="button" value="Prev" onclick="ClickPrev();"> </td>
</tr>
<tr>
    <td colspan=3>
    <input type="button" value="Synchronize" onclick="ClickSync();">
    </td>
</tr>
</table>
</form>

<a href="AG_DEVICEOS/ul_deps.htm"></a>
</body>
</html>
```

以下是供您参考和使用的 *tutorial.js* 的完整列表：

```
// UltraLite Tutorial

var DB_mgr;
var Connection;
var Table;

var GivenName = "";
var Surname = "";
var Street = "";
var City = "";
var Phone = "";
var ID = "";

function Connect()
{
    var     dir;
    var     open_parms;
    var     browser = navigator.platform;

    DB_mgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.Tutorial" );
    if( DB_mgr == null ) {
     alert( "Error: cannot create database manager: " + DB_mgr.sqlCode );
 return;
    }
    dir = DB_mgr.directory;
    if( browser == "Palm OS" ) {
 open_parms = "con=tutorial;palm_file=tutorial"
    } else {
     open_parms = "con=tutorial;" + "file_name=" + dir + "\\tutorial.udb";
    }
    try {
 Connection = DB_mgr.reOpenConnection( "tutorial" );
     if( Connection == null ) {
     Connection = DB_mgr.openConnection( open_parms );
 }
    } catch( ex ) {
```

```
     if( DB_mgr.sqlCode !=
DB_mgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
         alert( "Error: cannot connect to database: " + ex.getMessage() );
     return;
     }
    }
    try {
 Table = Connection.getTable( "Customer", null );
 if( Table != null ) {
     Table.open();
 }
    } catch( ex ) {
 alert( "Error: cannot open table: " + ex.getMessage() );
    }
}

function Fetch()
{
    if( Table.getRowCount() == 0 ) {
 GivenName = "";
 Surname = "";
 Street = "";
 City = "";
 Phone = "";
 ID = "";
 return;
    }
    ID = Table.getString( Table.schema.getColumnID( "ID" ) );
    GivenName = Table.getString( Table.schema.getColumnID( "GivenName" ) );
    Surname = Table.getString( Table.schema.getColumnID( "Surname" ) );
    Street = Table.getString( Table.schema.getColumnID( "Street" ) );
    if( Table.isNull( Table.schema.getColumnID( "City" ) ) ) {
 City = "";
    } else {
 City = Table.getString( Table.schema.getColumnID( "City" ) );
    }
    if( Table.isNull( Table.schema.getColumnID( "Phone" ) ) ) {
 Phone = "";
    } else {
 Phone = Table.getString( Table.schema.getColumnID( "Phone" ) );
    }
}

function Insert()
{
    try {
 Table.insertBegin();
 Table.setString( Table.schema.getColumnID( "GivenName" ), GivenName );
 Table.setString( Table.schema.getColumnID( "Surname" ), Surname );
 Table.setString( Table.schema.getColumnID( "Street" ), Street );
 if( City.length > 0 ) {
     Table.setString( Table.schema.getColumnID( "City" ), City );
 }
 if( Phone.length > 0 ) {
     Table.setString( Table.schema.getColumnID( "Phone" ), Phone );
 }
 Table.insert();
 Table.moveLast();
    } catch( ex ) {
 alert( "Error: cannot insert row: " + ex.getMessage() );
    }
}

function Next()
```



```
{
    if( ! Table.moveNext() ) {
 Table.moveLast();
    }
}

function Prev()
{
    if( ! Table.movePrevious() ) {
 Table.moveFirst();
    }
}

function Synchronize()
{
    var sync_parms;

    sync_parms = Connection.syncParms;
    sync_parms.setUserName( "tutorial" );
    sync_parms.setPassword( "tutorial" );
    sync_parms.setVersion( "tutorial" );
    sync_parms.setStream( sync_parms.STREAM_TYPE_TCPIP );
    try {
 Connection.synchronize();
    } catch( ex ) {
     alert( "Error: cannot synchronize: " + ex.getMessage() );
    }
}
```

UltraLite for M-Business Anywhere API 参考

目录

UltraLite for M-Business Anywhere 中的数据类型 54
AuthStatusCode 类 55
Connection 类 56
ConnectionParms 类 70
CreationParms 类 72
DatabaseManager 类 74
DatabaseSchema 类 78
IndexSchema 类 83
PreparedStatement 类 86
PublicationSchema 类 95
ResultSet 类 96
ResultSetSchema 类 113
SQLError 类 117
SQLType 类 126
SyncParms 类 128
SyncResult 类 138
TableSchema 类 141
ULTable 类 152
UUID 类 177

UltraLite for M-Business Anywhere 中的数据类型

JavaScript 只有一个数字数据类型和一个日期数据类型。

此 API 参考中的原型包括方法和属性说明中的多种其它数据类型。这些类型为内部 M-Business Anywhere 数据类型。在这里报告不同数字数据类型如 UInt32（无符号 32 位整数），以便用户大致了解可能提供的数据的大小和精度。报告与日期和时间相关的不同数据类型（Date、Time、Timestamp），以便您能够在必要时编写代码从提供的数据中抽取所需的信息。

AuthStatusCode 类

枚举那些可能会在 MobiLink 用户验证过程中报告的状态码。

此对象可用如下方式从 DatabaseManager 获得：

```
var authStatus = dbMgr.AuthStatusCode;
```

属性

以下常量是 AuthStatusCode 的属性

常量	值	说明
UNKNOWN	0	授权状态未知，可能是因为连接尚未执行同步。
VALID	1	用户 ID 和口令在同步时有效。
VALID_BUT_EXPIRES_SOON	2	用户 ID 和口令在同步时有效，但很快就要到期。
EXPIRED	3	用户 ID 或口令已到期；授权失败。
无效	4	用户 ID 或口令不正确；授权失败。
IN_USE	5	用户 ID 已在使用；授权失败。

toString 方法

生成授权状态码常量的字符串名称。

语法

String **toString()**;

返回值

代码的名称；如果是不可识别的代码，则返回 **unknown**。

Connection 类

表示与 UltraLite 数据库的连接。

连接可使用以下方法之一实例化：

- DatabaseManager.openConnection
- DatabaseManager.createDatabase

您必须打开一个连接，然后才可执行其它操作，而且在连接上完成所有操作之后，必须在应用程序终止前关闭该连接。

您必须首先关闭一个连接上的所有打开的表，然后才能关闭该连接。

当由于 UltraLite 数据库操作失败而抛出 JavaScript 错误时，会在 Connection 对象的 sqlCode 字段设置 SQL 错误代码。

属性

原型	说明
Boolean autoCommit	控制执行每个语句（插入、更新或删除）后是否执行提交。 如果 **autoCommit** 为假，则仅当用户调用 **commit()** 或 **rollback()** 方法时，才执行提交或回退。 缺省情况下，在每次成功执行语句后，都会执行数据库提交。如果提交失败，可选择执行其它 SQL 语句，然后再次执行提交，也可以选择执行回退语句。
String openParms（只读）	用一个以分号分隔的 [名称=值] 对列表的形式获取连接参数字符串。 请参见“UltraLite 连接参数”《UltraLite - 数据库管理和参考》。
DatabaseSchema databaseSchema（只读）	获取数据库模式。此属性仅在其连接为打开状态时才有效。
Boolean skipMBASync（读写）	控制是应该在一键同步期间同步数据库 (false)，还是应该跳过同步 (true)。 缺省值为 false。 请参见“一键同步”一节第 29 页。

原型	说明
Int32 sqlCode（只读）	获取上次在此连接上执行的操作的 SQL 代码。 SQL 代码是标准的 SQL Anywhere 代码，此连接上的任何后续 UltraLite 数据库操作都重置此代码。
SyncParms syncParms（只读）	获取此连接的同步设置。 请参见“UltraLite 的同步参数”一节 《UltraLite - 数据库管理和参考》。
SyncResult syncResult（只读）	获取此连接的最近一次同步的结果。 请参见“UltraLite 的同步参数”一节 《UltraLite - 数据库管理和参考》。
INVALID_DATABASE_ID（只读）	指示一个无效数据库的常量。

cancelGetNotification 方法

取消与给定名称匹配的所有队列上任何待执行 get-notification 调用。返回已取消事件通知的数目。

语法

UInt32 **cancelGetNotification**(String *queue_name*)

参数

- **queue_name** 事件通知队列的名称。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

changeEncryptionKey 方法

将数据库的加密密钥更改为指定的新密钥。

语法

changeEncryptionKey(String *newKey*)

参数

- **newKey** 数据库的新加密密钥。

注释

如果丢失了加密密钥，便无法打开数据库。

close 方法

关闭此连接。

语法

close()

注释

连接关闭后就无法重新打开。要重新打开连接，必须创建一个新的连接对象，然后再将其打开。

使用与已关闭的连接相关联的任何对象（如表或模式）均为错误。

在 JavaScript 中，已关闭的连接对象在关闭后不会自动设置为 NULL。建议在关闭连接后，将连接对象显式设置为 NULL。

commit 方法

将未完成的更改提交给数据库。

语法

commit()

countUploadRow 方法

返回下次进行同步时要上载的行数。

语法

UInt32 **countUploadRow(** String *pub-list*, UInt32 *threshold***)**

参数

- **pub-list** 以逗号分隔的要检查的发布的列表。

  请参见 PublicationSchema 类。

- **threshold** 用于确定要计数的最大行数的值，这样可以限制调用所花费的时间。0 值对应于最大限制。值 1 决定了是否有需要同步的行。**threshold** 必须在范围 **[0,0x0fffffff]** 内。

createNotificationQueue 方法

创建此连接的事件通知队列。

语法

void **createNotificationQueue(** String *queue_name*, String *parms*)

参数

- **queue_name** 事件通知队列的名称。
- **parms** 创建参数。目前未使用，设置为 NULL。

注释

使用提供的名称创建事件通知队列，它可用于随后的事件通知。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

declareEvent 方法

声明稍后可以进行注册和触发的事件。

语法

void **declareEvent(**String *event_name*)

参数

- **event_name** 事件的名称。

另请参见

- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

destroyNotificationQueue 方法

取消给定的事件通知队列。

语法

void **destroyNotificationQueue(**String *queue_name***)**

参数

- **queue_name** 现有事件通知队列的名称。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

executeNextSQLPassthroughScript 方法

执行下一个 SQL 直通脚本。

语法

void **executeNextSQLPassthroughScript()**

注释

执行脚本的过程中如发生错误，则抛出异常。

另请参见

- “executeSQLPassthroughScripts 方法”一节第 60 页
- “getSQLPassthroughScriptCount 方法”一节第 63 页

executeSQLPassthroughScripts 方法

执行所有可用的 SQL 直通脚本。

语法

void **executeSQLPassthroughScripts()**

注释

执行脚本的过程中如发生错误，则抛出异常。

另请参见

- “executeNextSQLPassthroughScript 方法”一节第 60 页
- “getSQLPassthroughScriptCount 方法”一节第 63 页

getDatabaseID 方法

获取当前数据库的 ID 值，该值由 setDatabaseID() 设置。

语法

UInt32 **getDatabaseID( )**

注释

如果尚未设置该值，将返回常量 Connection.INVALID_DATABASE_ID。

getGlobalAutoIncrementUsage 方法

返回已经使用的可用全局自动增量值的百分比。

语法

UInt16 **getGlobalAutoIncrementUsage( )**

注释

如果百分比接近 100，则应用程序应使用 **setDatabaseID** 为全局数据库 ID 设置一个新值。

getLastDownloadTime 方法

返回最近一次下载的时间戳。

语法

Date **getLastDownloadTime(** String *pub-list* **)**

参数

- **pub-list** 以逗号分隔的要检查的发布名列表。

注释

参数 pub-list 必须引用单个发布名。上次下载整个数据库的时间 SYNC_ALL_DB。

另请参见

- “PublicationSchema 类”一节第 95 页

getLastIdentity 方法

返回最近使用的标识值。

语法

UInt64 **getLastIdentity()**

注释

此函数等同于以下 SQL 语句：

```
SELECT @@identity
```

此函数在全局自动增量列的上下文中特别有用。返回值是数据库数据类型为 UNSIGNED BIGINT 的无符号 64 位整数。由于此语句仅确定最近指派的缺省值，所以您应在执行插入语句后尽快检索此值，以避免得到虚假结果。

有时，单个插入语句可能会包括类型为全局自动增量的多个列。在这种情况下，返回值是生成的缺省值之一，但没有可靠的方法能确定是哪一个。建议在设计数据库和编写插入语句时，应避免这种情况。

getNewUUID 方法

返回一个新的 UUID 值。

语法

UUID **getNewUUID()**

getNotification 方法

读取一个事件通知，同时返回该事件的名称。

语法

String **getNotification(** String *queue_name*, UInt32 *wait_ms***)**

参数

- **queue_name** 事件通知队列的名称；对于缺省连接队列为 NULL。
- **wait_ms** 最长等待时间，以毫秒为单位。传递 **UL_READ_WAIT_INFINITE** 可无限期等待。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

getNotificationParameter 方法

获取刚刚由 **getNotification** 读取的事件通知的参数。

语法

String **getNotificationParameter(** String *queue_name*, String *param_name***)**

参数

- **queue_name** 事件通知队列的名称。
- **param_name** 要读取的参数的名称；对于全部参数为 "*"。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

getSQLPassthroughScriptCount 方法

获取可运行的 SQL 直通脚本的数目。返回可运行的 SQL 直通脚本的数目。

语法

UInt32 **getSQLPassthroughScriptCount()**

另请参见

- “executeNextSQLPassthroughScript 方法”一节第 60 页
- “executeSQLPassthroughScripts 方法”一节第 60 页

getTable 方法

创建并返回对数据库中所请求表的引用。

语法

Table **getTable(**String *name*, String *persistName* **)**

参数

- **name** 要读取的表的名称。
- **persistName** 用于跨页 JavaScript 对象持久性的名称。如果不需要持久性（如，在应用程序只有单一 HTML 页面的情况下），则设置为空。

grantConnectTo 方法

授权某个用户 ID 使用指定口令访问 UltraLite 数据库。

语法

grantConnectTo(String *uid*, String *pwd***)**

参数

- **uid** 要授予访问权限的用户 ID。最大长度为 16 个字符。
- **pwd** 对应于用户 ID 的口令。

注释

如果指定了现有的用户 ID，则此函数更新该用户的口令。UltraLite 最多支持 4 个用户。

isOpen 方法

如果连接是打开的，则返回 true；否则返回 false。

语法

Boolean **isOpen()**;

prepareStatement 方法

将一个带有或不带有 IN 参数的 SQL 语句预编译并存储到一个 PreparedStatement 对象中。

语法

PreparedStatement **prepareStatement(**String *sql*, String *persistName***)**

参数

- **sql** 可能包含一个或多个 '?' 的 SQL 语句 IN 参数占位符。
- **persistName** 用于跨页 JavaScript 对象持久性的名称。如果不需要持久性（如，在应用程序只有单一 HTML 页面的情况下），则设置为空。

注释

此对象可用于多次有效地执行 SQL 语句。

registerForEvent 方法

注册（队列）以便接收事件的通知。

语法

void **registerForEvent(**String *event_name*, String *object_name*, String *queue_name*, Boolean *register***)**

参数

- **event_name** 事件的名称。
- **object_name** 事件适用的对象的名称（例如，表名）。
- **queue_name** 事件通知队列的名称。
- **register** TRUE 表示注册，FALSE 表示注销。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “sendNotification 方法”一节第 67 页
- “triggerEvent 方法”一节第 68 页

resetLastDownloadTime 方法

重置最近一次下载的时间。

语法

void **resetLastDownloadTime(**String *pub-list***)**

参数

- **pub-list** 以逗号分隔的要重置的发布名列表。

revokeConnectFrom 方法

撤消指定的用户 ID 对 UltraLite 数据库的访问权限。

语法

revokeConnectFrom(String *uid* **)**

参数

- **uid** 要从数据库访问中排除的用户 ID。最大长度为 16 个字符。

rollback 方法

将未完成的更改回退到数据库。

语法

rollback( )

rollbackPartialDownload 方法

回退失败的同步所做的更改。

语法

rollbackPartialDownload( )

注释

在同步的下载阶段出现通信错误时，UltraLite 可以应用已下载的更改，以便可以从同步中断的位置继续同步。如果不需要下载的更改（用户或应用程序不想从该位置继续下载），则 RollbackPartialDownload 回退失败的下载事务。

saveSyncParms 方法

保存同步参数供 HotSync 使用或在一键同步期间使用。

语法

saveSyncParms( )

注释

不要将 saveSyncParms 方法与 Connection.SyncParms 属性混淆。SyncParms 属性用于定义此连接的同步参数。saveSyncParms 方法只保存这些参数，以便 HotSync 能够使用它们。

另请参见

- “一键同步”一节第 29 页

sendNotification 方法

将通知发送到与给定名称匹配的所有队列（包括当前连接中任何此类队列）。返回已发送通知的数目（匹配队列的数目）。

语法

UInt32 **sendNotification(** String *queue_name*, String *event_name*, String *parms***)**

参数

- **queue_name** 事件通知队列的名称。
- **event_name** 事件的名称。
- **parms** 通知的参数字符串（如果有）。符合 "名称=值" 格式。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “triggerEvent 方法”一节第 68 页

setDatabaseID 方法

设置用于全局自动增量列的数据库 ID 值。

语法

setDatabaseID(UInt32 *value* **)**

参数

- **value** 数据库 ID 值。**value** 必须在 **[0,0x0fffffff]** 范围内。

startSynchronizationDelete 方法

将此连接所做的所有后续删除都标记为进行同步。

语法

startSynchronizationDelete( )

注释

调用此函数后，将再次同步所有删除操作。

stopSynchronizationDelete 方法

防止同步删除操作。

语法

stopSynchronizationDelete( )

注释

这一方法适用于从 UltraLite 数据库中删除旧信息以节省空间而又不从统一数据库中删除这些信息的情况。

synchronize 方法

使用当前的 SyncParms 对象同步数据库。

语法

synchronize( )

注释

此连接的 SyncResult 对象中将报告详细的结果状态。使用此连接的 Connection.SyncParms 对象中定义的同步属性执行同步。

synchronizeWithParm 方法

使用指定的 SyncParms 对象同步数据库。

语法

synchronizeWithParm(SyncParms *parms***)**

参数

- **parms** 用于此同步的 SyncParms 对象。

注释

此方法能够在连接间共享同步参数。

此连接的 SyncResult 对象中将报告详细的结果状态。

triggerEvent 方法

触发事件（并将通知发送到所有已注册的队列）。返回已发送事件通知的数目。

语法

UInt32 **triggerEvent(** String *event_name*, String *parms***)**

参数

- **event_name** 事件通知队列的名称。
- **parms** 附加参数。

另请参见

- “使用事件通知”一节 《UltraLite - 数据库管理和参考》
- “createNotificationQueue 方法”一节第 59 页
- “cancelGetNotification 方法”一节第 57 页
- “declareEvent 方法”一节第 59 页
- “destroyNotificationQueue 方法”一节第 60 页
- “getNotification 方法”一节第 62 页
- “registerForEvent 方法”一节第 65 页
- “sendNotification 方法”一节第 67 页

validateDatabase 方法

校验此连接上的数据库。

语法

void **validateDatabase(** UInt16 *type*, String *tablename***)**

参数

- **type** 要执行的校验的类型。请参见“属性”一节第 74 页。
- **tablename** 要校验的特定表名；如果为空，则校验整个数据库。

注释

此方法可用于校验一个特定表或整个数据库。

ConnectionParms 类

指定用于打开与 UltraLite 数据库的连接的参数。

数据库由一个已通过验证的用户 **DBA** 来创建，其初始口令为 **sql**。缺省情况下，使用用户 ID **DBA** 和口令 **sql** 打开连接。要禁用缺省用户，使用 Connection.revokeConnectFrom。要添加用户或更改用户口令，使用 Connection.grantConnectTo。

当前，任何时候都只能打开一个连接。在某一给定时间，只能有一个数据库处于活动状态。在其它连接处于打开状态时尝试打开与另一数据库的连接会导致出错。

属性

这里列出了该类的属性。

原型	说明
String additionalParms（读写）	以 **name**=*value* 对的形式指定的附加参数，参数之间用分号隔开。
String cacheSize（读写）	高速缓存的大小。CacheSize 值以字节为单位指定。对于千字节，使用后缀 k 或 K，对于兆字节，使用后缀 m 或 M。 请参见“UltraLite CACHE_SIZE 连接参数”一节《UltraLite - 数据库管理和参考》。
String connectionName（读写）	连接的名称。连接名用于在多个 Web 页间共享单一连接。 请参见“UltraLite CON 连接参数”一节 《UltraLite - 数据库管理和参考》和“跨页维护连接和应用程序状态”一节 第 11 页。
String creatorIdOnPalm	Palm 设备上的 UltraLite 数据库的创建者 ID。
String databaseOnCE（读写）	Windows Mobile 上数据库的文件名。 请参见“UltraLite CE_FILE 连接参数”一节 《UltraLite - 数据库管理和参考》。
String databaseOnDesktop（读写）	部署到 Windows 的数据库的文件名。 请参见“UltraLite DBF 连接参数”一节 《UltraLite - 数据库管理和参考》。
String databaseOnPalm（读写）	Palm 上 UltraLite 数据库的文件名。 请参见“UltraLite PALM_FILE 连接参数”一节 《UltraLite - 数据库管理和参考》。

原型	说明
String encryptionKey（读写）	用于数据库加密的密钥。OpenConnection 必须使用在数据库创建期间指定的同一密钥。对密钥的建议如下： 1. 选择任意长度的字符串 2. 选择由多个数字、字母和特殊字符组成的字符串，以减少破解密钥的机会。 请参见“UltraLite DBKEY 连接参数”一节 《UltraLite - 数据库管理和参考》。
String password（读写）	已通过验证用户的口令。数据库由一个已通过验证的用户 (**DBA**) 使用口令 **sql** 初始创建。如果数据库不区分大小写，则口令也不区分大小写；反之，如果数据库区分大小写，则口令也区分大小写。缺省值为 *sql*。 请参见“UltraLite PWD 连接参数”一节 《UltraLite - 数据库管理和参考》。
String userID（读写）	数据库的已通过验证用户。数据库由一个已通过验证用户 DBA 初始创建。UserID 不区分大小写。缺省值为 *DBA*。 请参见“UltraLite UID 连接参数”一节 《UltraLite - 数据库管理和参考》。

toString 方法

生成授权状态码常量的字符串名称。

语法

String **toString()**;

返回值

代码的名称；如果是不可识别的代码，则返回 **unknown**。

CreationParms 类

定义可在创建 UltraLite 数据库时指定的参数。

某些 UltraLite 数据库选项必须在创建数据库时设置。以下参数可在使用 createDatabase 方法创建数据库时提供。请参见“createDatabase 方法”一节第 75 页。

属性

这里列出了该类的属性。有关相应说明的详细信息，请参见“为 UltraLite 选择数据库创建参数”一节 《UltraLite - 数据库管理和参考》。

原型	说明
Boolean caseSensitive	将 UltraLite 数据库中的字符串比较设置成区分大小写。
UInt32 checksumLevel	设置数据库中校验和校验的级别。 缺省值为 0。
String dateFormat	设置从数据库中检索日期所用的缺省字符串格式。
String dateOrder	控制对年、月、日的日期顺序的解释。
UInt32 maxHashSize	设置用于散列 UltraLite 索引的最大字节数。 缺省值为 0。
UInt32 nearestCentury	控制字符串到日期的转换中对两位的年份的解释。
Boolean obfuscate	控制是否对数据库中的数据进行模糊处理。模糊处理是一种简单加密方式。
UInt32 pageSize	定义数据库页面大小。有效值是： 1024, 2048, 4096, 8192, 16384. 缺省值为 4096。
UInt32 precision	指定任意数字算术的结果中数字的最大位数。
UInt32 scale	指定算术结果被截断为最大精度值时小数点后的最小位数。
String timeFormat	为从数据库中检索得到的时间设置格式。
String timestampFormat	确定 UltraLite 中时间戳的格式设置。
String timestampIncrement	确定 UltraLite 中时间戳是如何截断的。

原型	说明
Boolean utf8Encoding	使用 UTF-8 格式（Unicode 的 8 位多字节编码）对数据进行编码。

DatabaseManager 类

管理与 UltraLite 数据库的连接。

您必须打开一个连接，然后才可执行其它操作，而且在连接上完成所有操作之后，必须在应用程序终止前关闭该连接。您必须首先关闭一个连接上的所有打开的表，然后才能关闭该连接。

属性

这里列出了该类的属性。

属性	说明
AuthStatusCode **AuthStatusCode**（只读）	获取与最近一次同步相关联的 AuthStatusCode 对象。
String **directory**（只读）	在其中运行 M-Business Anywhere 的目录。 在 Palm OS 上，此属性为 NULL。
UInt32 **runtimeType**（只读）	运行时类型：UltraLite 运行时（独立）库或 UltraLite 数据库引擎。此值是枚举，为以下值之一： ● DatabaseManager.UL_STANDALONE ● DatabaseManager.UL_ENGINE_CLIENT
Int32 **sqlCode**（只读）	获取与最近一次操作相关联的 SQL 代码值。
SQLError **SQLError**（只读）	获取 SQLError 对象。
SQLType **SQLType**（只读）	获取 SQLType 对象。
PODSUInt32 **UL_STANDALONE**（只读）	指示运行时类型为 UltraLite 运行时库的常量。
PODSUInt32 **UL_ENGINE_CLIENT**（只读）	指示运行时类型为 UltraLite 数据库引擎的常量。
UInt16 **VALIDATE_EXPRESS**（只读）	用于通过 validateDatabase 方法指定快速校验（比 **VALIDATE_FULL** 校验速度快，但不彻底）的常量。

属性	说明
UInt16 **VALIDATE_FULL**（只读）	用于通过 validateDatabase 方法指定完全校验（校验表、索引和数据库页）的常量。

createDatabase 方法

创建一个数据库，并打开与 **access_parms** 指定的数据库的连接。

语法

```
Connection createDatabase(
String access_parms ,
PODSArray *coll_bytes,
String create_parms
)
```

参数

- **access_parms** 用于连接到数据库的参数。access_parms 用于指定连接参数（包括数据库文件名和位置）以及打开连接。

  请参见“连接到 UltraLite 数据库” 《UltraLite - 数据库管理和参考》。

- **coll_bytes** 定义要创建的数据库使用的数据库归类的字节数组。多个源文件以 JavaScript 源文件 (*.js*) 的形式随 UltraLite 提供，它们位于 *install-dir\UltraLite\Collations\js* 中，文件名的形式为 **coll_XXXXX.js**，其中 XXXXX 代表归类名。例如，*coll_1250LATIN2.js*。

  归类文件必须包括到主 html 文件中并位于数据库逻辑之前。字节数组变量在 coll_XXXXX.js 文件中定义。

- **create_parms** 用于创建数据库的参数。参数关键字是不区分大小写的，而大多数值是区分大小写的。create_parms 用于指定仅在创建数据库时才可以指定的参数。

  请参见“为 UltraLite 选择数据库创建参数”一节 《UltraLite - 数据库管理和参考》。

返回值

无返回值。

注释

如果该数据库已经存在，则抛出 SQLE_DATABASE_NOT_CREATED 异常。

在某一给定时间，只能有一个数据库处于活动状态。如果与不同数据库的连接处于打开状态，在尝试打开数据库连接时会导致出错。

dropDatabase 方法

删除指定的数据库。

语法

void **dropDatabase(**String *parms***)**

参数

- **parms** 用于标识数据库的参数。

注释

parms 是以分号分隔的 "关键字=值" 对的列表 (**"param1=value1;param2=value2"**)。参数关键字不区分大小写，而大多数值是区分大小写的。

不能删除具有已打开连接的数据库。

getDatabaseOptions 方法

语法

Connection **openConnection(** String *parms***)**

openConnection 方法

打开与 **parms** 指定的数据库的连接。

语法

Connection **openConnection(** String *parms***)**

参数

- **parms** 保存用于打开连接的参数、形式为一组 "关键字=值" 对的字符串。参数关键字不区分大小写，而大多数值是区分大小写的。

返回值

一个打开的连接。

注释

如果该数据库不存在，则抛出错误。您可以在错误捕获代码中检查 Connection.sqlCode 以便标识错误的原因。

在某一给定时间，只能有一个数据库处于活动状态。在其它连接处于打开状态时尝试打开与另一数据库的连接会导致出错。

reOpenConnection 方法

返回一个打开的 Connection 对象。

语法

Connection **reOpenConnection(** String *connectionName* **)**

参数

- **connectionName** 要重新打开的连接的名称，在 ConnectionParms.connectionName 属性中指定。

返回值

此方法用于在多个 Web 页间维护连接。

validateDatabase 方法

在当前未连接到 UltraLite 数据库的情况下校验该数据库。

语法

void **validateDatabase(**
String *start_parms*;
UInt16 *type***)**

参数

- **start_parms** 连接到数据库的参数。
- **type** 要执行的校验的类型。请参见 DatabaseManager 类“属性”一节第 74 页中的 **VALIDATE_EXPRESS** 和 **VALIDATE_FULL** 属性。

DatabaseSchema 类

表示 UltraLite 数据库的模式。**DatabaseSchema** 对象会附加到连接上，并且只有在该连接处于打开状态时才有效。

常量

常量	说明
SYNC_ALL_DB	同步数据库中所有的表（除了定义为不同步的表）。
SYNC_ALL_PUBS	同步数据库中的所有发布。

这里列出了该类的成员。

getCollationName 方法

返回标识此数据库中使用的字符集和排序顺序的字符串。

语法

String **getCollationName()**

getDatabaseProperty 方法

返回指定数据库属性的值。

语法

String **getDatabaseProperty(**String *name***)**

参数

- **name** 数据库属性的名称。

注释

识别的属性有：

- **"date_format"** 数据库进行字符串转换所用的日期格式。
- **"date_order"** 数据库用于字符串转换的日期顺序。
- **"nearest_century"** 数据库用于字符串转换的最近一个世纪。
- **"precision"** 数据库用于字符串转换的浮点精度。

- **"scale"** 数据库进行字符串转换的过程中，算术结果按最大精度值截断时小数点后的最少位数。
- **"time_format"** 数据库用于字符串转换的时间格式。
- **"timestamp_format"** 数据库用于字符串转换的时间戳格式。
- **"timestamp_increment"** 两个唯一时间戳之间的最小差异，以微秒（百万分之一秒）为单位。

getDateFormat 方法

返回进行字符串转换所用的日期格式。

语法

String **getDateFormat()**

getDateOrder 方法

返回进行字符串转换所用的日期顺序。

语法

String **getDateOrder()**

getNearestCentury 方法

返回进行字符串转换所用的最近一个世纪。

语法

String **getNearestCentury()**

getPrecision 方法

返回用于字符串转换的浮点精度。

语法

String **getPrecision( )**

getPublicationCount 方法

返回数据库中的发布数目。

语法

UInt16 **getPublicationCount( )**

注释

发布 ID 的范围是 1 到 **getPublicationCount()**（含边界值）。

注意：发布 ID 和计数在模式升级过程中可能发生变化。为了正确地识别发布，可按名称访问它，或者在模式升级后刷新任何高速缓存中的 ID 和计数。

getPublicationName 方法

返回由指定的发布 ID 标识的发布的名称。

语法

String **getPublicationName(** UInt16 *pubID* **)**

参数

- **pubID** 发布 ID。**pubID** 必须在范围 **[1,getPublicationCount()]** 内（含边界值）。

注释

注意：发布 ID 和计数在模式升级过程中可能发生变化。为了正确地识别发布，可按名称访问它，或者在模式升级后刷新任何高速缓存中的 ID 和计数。

getPublicationSchema 方法

返回与指定发布对应的发布模式。

语法

PublicationSchema **getPublicationSchema(** String *name* **)**

参数

- **name** 发布的名称。

getSignature 方法

返回此数据库的签名。

语法

String **getSignature( )**

getTableAGDBSet 方法

使用特定的表名绑定到 AGDBSet 对象。返回 TableAGDBSet 对象的一个实例。

语法

TableAGDBSet **getTableAGDBSet(** String *tablename***)**

参数

- **tablename** 要绑定到 AGDBSet 的表的名称。

注释

有关 AGDB 对象的文档，请参见 Sybase API Reference for M-Business Anywhere。

getTableCount 方法

返回数据库中的表的数目。

语法

UInt16 **getTableCount( )**

返回值

表的数目；如果连接未打开，则为 0。

注释

表 ID 的范围是 1 到 **getTableCount()**（含边界值）。

getTableName 方法

返回由指定的表 ID 所标识的表的名称。

语法

String **getTableName(** UInt16 *tableID***)**

参数

- **tableID** 表 ID。**tableID** 必须在 **[1,getTableCount()]** 内（含边界值）。

注释

注意：表 ID 在模式升级过程中可发生变化。为了正确地标识表，请按名称访问它，或者在模式升级后刷新高速缓存中的 ID。

getTimeFormat 方法

返回进行字符串转换所用的时间格式。

语法

String **getTimeFormat()**

getTimestampFormat 方法

返回进行字符串转换所用的时间戳格式。

语法

String **getTimestampFormat()**

isCaseSensitive 方法

如果数据库区分大小写，则返回 true；否则返回 false。

语法

Boolean **isCaseSensitive()**

isOpen 方法

如果数据库模式是打开的，则返回 true；否则返回 false。

语法

Boolean **isOpen()**

IndexSchema 类

表示 UltraLite 表索引的模式。

无法直接实例化此对象。索引模式是使用 TableSchema.getPrimaryKey、TableSchema.getIndex 和 TableSchema.getOptimalIndex 方法创建的。

getColumnCount 方法

返回此索引中的列数。

语法

UInt16 **getColumnCount( )**

注释

在索引中，列 ID 的范围是从 1 到 getColumnCount()（包括 1 和 getColumnCount()）。

getColumnName 方法

返回此索引中 *colIDInIndex* 列的名称。

语法

String **getColumnName(**UInt16 *colIDInIndex* **)**

参数

- **colIDInIndex** 该列的此索引中的 ID。colIDInIndex 必须在 [1, getColumnCount()] 范围内。

getName 方法

返回此索引的名称。

语法

String **getName()**

getReferencedIndexName 方法

返回此索引为外键时引用的主索引的名称。

语法

String **getReferencedIndexName()**

getReferencedTableName 方法

返回索引为外键时引用的主表的名称。

语法

String **getReferencedTableName()**

isColumnDescending 方法

如果按降序使用列，则返回 true；如果按升序使用列，则返回 false。

语法

Boolean **isColumnDescending(**String *name***)**

参数

- **name** 列的名称。

isForeignKey 方法

如果索引是外键，则返回 true；如果索引不是外键，则返回 false。

语法

Boolean **isForeignKey()**

注释

外键中的列可以引用另一个表的非空唯一索引。

isForeignKeyCheckOnCommit 方法

如果在提交时检查参照完整性，则返回 true；如果在插入和更新时检查参照完整性，则返回 false。

语法

Boolean **isForeignKeyCheckOnCommit()**

isForeignKeyNullable 方法

如果此外键可为空，则返回 true；如果此外键不可为空，则返回 false。

语法

Boolean **isForeignKeyNullable()**

isPrimaryKey 方法

如果索引是主键，则返回 true；如果索引不是主键，则返回 false。

语法

Boolean **isPrimaryKey()**

注释

主键中的列不可以为空值。

isUniqueIndex 方法

如果索引是唯一索引，则返回 true；否则返回 false。

语法

Boolean **isUniqueIndex()**

注释

唯一索引中的列可以为空值。

isUniqueKey 方法

如果索引是唯一键，则返回 true；如果索引不是唯一键，则返回 false。

语法

Boolean **isUniqueKey()**

注释

唯一键中的列不可以为空值。

PreparedStatement 类

表示具有或不具有 IN 参数的预编译的 SQL 语句。在运行时使用 Connection.prepareStatement 创建。

此对象之后可用于多次有效地执行此语句。

在关闭一个预准备语句时，所有与该语句关联的 ResultSet 和 ResultSetSchema 对象也将被关闭。出于资源管理原因，最好在使用完预准备语句后显式关闭这些语句。

appendBytesParameter 方法

将指定的字节数组的指定子集添加到指定的 SQLType.LONGBINARY 列的新值中。

语法

```
appendBytesParameter(
  UInt16 parameterID,
  Array value,
  UInt32 srcOffset,
  UInt32 count
)
```

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 要添加到参数的当前新值中的值。
- **srcOffset** 源数组中的起始位置。
- **count** 要复制的字节数。

注释

数组 **value** 从 srcOffset（从 0 开始）位置到 `srcOffset+count-1` 位置的字节会添加到指定参数的值中。在插入时，**insertBegin** 将新值初始化为参数的缺省值。

如果以下任何一个条件为 true，则抛出代码为 SQLError.SQLE_INVALID_PARAMETER 的错误，并且不会修改目标：

- **value** 参数为空值。
- **srcOffset** 参数为负值。
- **count** 参数为负值。
- **srcOffset+count** 大于源数组的长度 **value.length**。

appendStringChunkParameter 方法

将此字符串添加到指定的 SQLType.LONGVARCHAR 的新值中。

语法

appendStringChunkParameter(
UInt16 *parameterID*,
String *value*,
)

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 要添加到参数的当前新值中的值。

示例

以下语句将字符串 **XYZ** 的一百个实例添加到第一个参数中：

```
for ( I = 0; I < 100; I++ ){
  stmt.appendStringChunkParameter( 1, "XYZ" );
}
```

close 方法

关闭预准备语句。

语法

close( )

注释

在关闭一个预准备语句时，所有与该语句关联的 ResultSet 和 ResultSetSchema 对象也将被关闭。

建议在关闭 preparedStatement 对象后，立即将其设置为空值。

executeQuery 方法

执行 SQL SELECT 语句并返回结果集。

语法

ResultSet **executeQuery(** String *persistName* **)**

参数

- **persistName** 用于跨页 JavaScript 对象持久性的名称。如果不需要持久性（如，在应用程序只有单一 HTML 页面的情况下），则设置为空。

返回值

该查询的结果集（一组行的形式）。

executeStatement 方法

执行一个不返回结果集的语句，如 SQL INSERT、DELETE 或 UPDATE 语句。

语法

Int32 **executeStatement( )**

返回值

受该语句影响的行的数目。

注释

如果 Connection.autoCommit 为 true，此语句只在一行或多行受其影响时才提交。

getPlan 方法

返回描述 UltraLite 将用来执行查询的访问计划的字符串。

语法

String **getPlan( )**

注释

此方法主要供开发期间使用。

另请参见

- “UltraLite 中的执行计划”一节 《UltraLite - 数据库管理和参考》

getResultSetSchema 方法

返回描述此查询语句的结果集的模式。

语法

ResultSetSchema **getResultSetSchema()**

hasResultSet 方法

如果执行此语句时生成结果集，则返回 true；如果执行此语句时不生成结果集，则返回 false。

语法

Boolean **hasResultSet( )**

isOpen 方法

如果预准备语句是打开的，则返回 true；否则返回 false。

语法

Boolean **isOpen( )**

setBooleanParameter 方法

使用布尔型设置指定参数的值。

语法

setBooleanParameter(UInt16 *parameterID*, Boolean *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句设置第一个参数的值：

```
stmt.setBooleanParameter(1, false);
```

setBytesParameter 方法

使用字节数组设置指定参数的值。

语法

setBytesParameter(UInt16 *parameterID*, Array *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

注释

仅适用于 SQLType.BINARY 或 SQLType.LONGBINARY 类型的列。

示例

以下语句设置第一个参数的值：

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0;
```



```
blob[ 2 ] = 68;
stmt.setBytesParameter( 1, blob );
```

setDateParameter 方法

使用日期设置指定的 SQLType.DATE 类型参数的值。

语法

setDateParameter(UInt16 *parameterID*, Date *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

注释

仅与 Date 对象的年、月和日字段相关。

示例

以下语句将第一个参数的值设置为 2004/09/27：

```
stmt.setDateParameter(
    1, new Date( 2004,9,27,0,0,0,0 )
);
```

setDoubleParameter 方法

使用 **double** 设置指定参数的值。

语法

setDoubleParameter(UInt16 *parameterID*, Double *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句设置第一个参数的值：

```
stmt.setDoubleParameter( 1, Number.MAX_VALUE );
```

setFloatParameter 方法

设置指定的 SQLType.REAL 参数的值。

语法

setFloatParameter(UInt16 *parameterID*, Float *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句设置第一个参数的浮点值：

```
stmt.setFloatParameter( 1,
    (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

setIntParameter 方法

使用 UInt16 型设置指定参数的值。

语法

setIntParameter(UInt16 *parameterID*, UInt16 *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句将第一个参数的值设置为 **2147483647**：

```
stmt.setIntParameter( 1, 2147483647 );
```

setLongParameter 方法

设置指定参数的值。

语法

setLongParameter(UInt16 *parameterID*, Int64 *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句将第一个参数的值设置为 **9223372036854770000**：

```
stmt.setLongParameter( 1, 9223372036854770000 );
```

setNullParameter 方法

将指定参数设置为 SQL NULL 值。

语法

setNullParameter(UInt16 *parameterID* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。

setShortParameter 方法

设置指定参数的值。

语法

setUInt16Parameter(UInt16 *parameterID*, UInt16 *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句将第一个参数的值设置为 **32767**：

```
stmt.setShortParameter( 1, 32767 );
```

setStringParameter 方法

设置指定参数的值。

语法

setStringParameter(UInt16 *parameterID*, String *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句将第一个参数的值设置为 **ABC**：


```
stmt.setStringParameter( 1, "ABC" );
```

setTimeParameter 方法

使用日期设置指定的 SQLType.TIME 类型参数的值。

语法

setTimeParameter(UInt16 *parameterID*, Date *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

注释

仅与 Date 对象的小时、分钟和秒字段相关。

示例

以下语句将第一个参数的值设置为 18:02:13:0000：

```
stmt.setTimeParameter(
    1, new Date( 1966,4,1,18,2,13,0 )
);
```

setTimestampParameter 方法

使用 **Timestamp** 设置指定参数的值。

语法

setTimestampParameter(UInt16 *parameterID*, Date *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

示例

以下语句将第一个参数的值设置为 1966/04/01 18:02:13:0000：

```
stmt.setTimestampParameter(
    1, new Date( 1966,4,1,18,2,13,0 )
);
```


setULongParameter 方法

使用被视为无符号值的 Double 型设置指定参数的值。

语法

setULongParameter(UInt16 *parameterID*, UInt64 *value* **)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。使用 Double 型表示 64 位无符号整数的值。

注释

请参见类 Unsigned64。

示例

以下语句设置第一个参数的值：

```
stmt.setLongParameter( 1, 9223372036854770000 * 4096 );
```

setUUIDParameter 方法

使用 **UUID** 设置指定参数的值。

语法

setUUIDParameter(UInt16 *parameterID*, UUID *value***)**

参数

- **parameterID** 参数 ID 号。结果集中第一个参数的 ID 值为 1。
- **value** 参数的新值。

PublicationSchema 类

表示 UltraLite 发布的模式。

无法直接实例化此类。发布模式是使用 DatabaseSchema.getPublicationSchema 方法创建的。

发布由名称来标识。一些方法需要提供以逗号分隔的发布名列表。

DatabaseSchema 对象可提供两个特殊的常量值。**SYNC_ALL_DB** 对应整个数据库。**SYNC_ALL_PUBS** 对应所有发布。

getName 方法

返回此发布的名称。

语法

String **getName()**

ResultSet 类

表示 UltraLite 数据库中的结果集。在运行时使用 PreparedStatement.executeQuery 创建。

属性

这里列出了该类的属性。

属性	说明
ResultSetSchema schema（只读）	此结果集的模式。只有在其预准备语句处于打开的情况下此属性才有效。
NULL_TIMESTAMP_VAL	指示时间戳值为 NULL 的常量。

appendBytes 方法

将指定的字节数组的指定子集添加到指定的 SQLType.LONGBINARY 列的新值中。

语法

```
appendBytes(
  UInt16 columnID,
  Array value,
  UInt32 srcOffset,
  UInt32 count
)
```

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。
- **srcOffset** 要附加到列的当前新值中的值。
- **count** 要复制的字节数。

注释

数组 **value** 从 srcOffset（从 0 开始）位置到 **srcOffset+count-1** 位置的字节会添加到指定列的值中。在插入时，insertBegin 将新值初始化为列的缺省值。直到执行 **insert** 之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

如果以下任何一个条件为 true，则抛出代码为 SQLCode.SQLE_INVALID_PARAMETER 的错误，并且不会修改目标：

- **value** 参数为空值。
- **srcOffset** 参数为负值。

- **count** 参数为负值。
- **srcOffset+count** 大于源数组的长度 **value.length**。

对于其它错误，抛出具有相应错误代码的 **SQLException**。

appendStringChunk 方法

将指定字符串添加到指定的 SQLType.LONGVARCHAR 列的新值中。

语法

appendStringChunk(
 UInt16 *columnID*,
 String *value*
)

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

示例

以下语句将字符串 **XYZ** 的一百个实例添加到第一列的值中：

```
for ( I = 0; I < 100; I++ ){
  t.AppendStringChunk( 1, "XYZ" );
}
```

close 方法

释放与此对象关联的所有资源。

语法

close()

deleteRow 方法

删除当前行。

语法

deleteRow( )

注释

在每次执行 deleteRow 之前必须先调用 updateBegin。

getAGDBSet 方法

绑定到结果集中的 AGDBSet 对象。返回 TableAGDBSet 对象的一个实例。

语法

AGDBSet **getAGDBSet()**

注释

有关 AGDB 对象的文档，请参见 Sybase API Reference for M-Business Anywhere。

getBoolean 方法

以布尔值形式返回指定列的值。

语法

Boolean **getBoolean(** UInt16 *index* **)**

参数

- **index** 列的 ID 号。结果集中第一列的 ID 为 1。

getBytes 方法

以字节数组形式返回指定列的值。

语法

Array **getBytes(** UInt16 *index* **)**

参数

- **index** 列的 ID 号。结果集中第一列的 ID 为 1。

注释

仅对 SQLType.BINARY 或 SQLType.LONGBINARY 类型的列有效。

getBytesSection 方法

从指定的源偏移量开始，将指定的 SQLType.LONGBINARY 或 SQLType.BINARY 列的内容的子集复制到目标字节数组的指定偏移量处。

语法

UInt32 **getBytesSection(**
  UInt16 *index*,
  UInt32 *srcOffset*,


```
  Array  dst,
  UInt32 dstOffset,
  UInt32 count
)
```

参数

index　包含二进制数据的列从 1 开始的序号。

srcOffset　源字节数组的零相对偏移。源偏移必须大于或等于 0，否则将引发 SQLE_INVALID_PARAMETER 错误。另外，还允许大于 64K 的缓冲区。

dst　目标字节数组。

dstOffset　目标字节数组的零相对偏移。目标偏移必须大于或等于 0，否则将引发 SQLE_INVALID_PARAMETER 错误。另外，还允许大于 64K 的缓冲区。

count　要移动的字节数。计数必须大于或等于 0。

返回值

读取的字节数。

注释

位于源数组的 `srcOffset`（从 0 开始）位置到 `srcOffset+count-1` 位置的字节会分别复制到目标数组的 dstOffset 到 dstOffset+count-1 的位置。如果在复制完指定的字节数之前遇到源值结尾，则目标数组的剩余部分保持不变。

如果以下任何一个条件为 true，则抛出错误，SQLError 代码会设置为 SQLE_INVALID_PARAMETER，并且不会修改目标：

- dst 参数为空值。
- srcOffset 参数为负。
- dstOffset 参数为负。
- count 参数为负。
- dstOffset + count 大于 dst.length（目标数组的长度）。

错误集

SQLE_CONVERSION_ERROR　如果列数据类型不是 BINARY 或 LONG BINARY，则会出现此错误。

SQLE_INVALID_PARAMETER　如果列数据类型为 BINARY 并且偏移不是 0 或 1，或者数据长度小于 0，则会出现此错误。

如果列数据类型为 LONG BINARY，而偏移小于 1，也会出现此错误。

getDate 方法

以 Date 的形式返回值。

语法

Date **getDate(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getDouble 方法

以 Double 的形式返回值。

语法

Double **getDouble(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getFloat 方法

返回指定列的值。

语法

Float **getFloat(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getInt 方法

返回指定列的值。

语法

UInt32 **getInt(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getLong 方法

返回指定列的值。

语法

Int64 **getLong(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getRowCount 方法

返回结果集中的行数。

语法

UInt32 **getRowCount( )**

getRowCountWithThreshold 方法

返回结果集中的行数，最多不超过行的指定阈值数。

语法

UInt32 **getRowCount(** UInt32 *threshold***)**

参数

threshold 此值为行计数操作指定一个限定值。如果行数大于阈值，则结果为阈值。当存在大量行时，计数行是一项高开销操作。一些应用程序只需要知道是否有多于指定数目的行存在（例如，确定是否为用户提供一个选项用于请求更多行），而不需要精确的行计数。如果此值为零，则计数所有行。

getShort 方法

以 Int16 的形式返回值。

语法

Int16 **getShort(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getString 方法

以 String 的形式返回值。

语法

String **getString(** UInt16 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

getStringChunk 方法

从指定的偏移量开始，将指定的 SQLType.LONGVARCHAR 列的值的子集复制到 String 对象。

语法

String **getStringChunk(**
  UInt16 *index*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

参数

- **index**　要获取的结果集中从 1 开始的序号
- **srcOffset**　字符串值的从 0 开始的起始位置。
- **count**　要复制的字符数。

返回值

复制了指定字符的字符串。

getTime 方法

以 Date 的形式返回值。

语法

Date **getTime(** UInt16 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

getTimestamp 方法

以 Date 的形式返回值。

语法

Date **getTimestamp(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getULong 方法

以 64 位无符号整数的形式返回值。

语法

UInt64 **getULong(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getUUID 方法

以 UUID 的形式返回列的值。

语法

UUID **getUUID(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

注释

此列必须为 SQLType.BINARY 类型，且长度为 16。

isBOF 方法

如果当前行位置在第一行之前，则返回 **true**，否则返回 **false**。

语法

Boolean **isBOF( )**

isEOF 方法

如果当前行位置在最后一行之后，则返回 **true**，否则返回 **false**。

语法

Boolean **isEOF( )**

isNull 方法

如果值为空，则返回 **true**，否则返回 **false**。

语法

Boolean **isNull(** UInt16 *index* **)**

参数

index　列索引值。

isOpen 方法

如果 ResultSet 已打开，则返回 **true**，否则返回 **false**。

语法

Boolean **isOpen( )**

moveAfterLast 方法

移至 ULResultSet 的最后一行之后的位置。

语法

moveAfterLast( )

moveBeforeFirst 方法

移至第一行之前的位置。

语法

moveBeforeFirst( )

moveFirst 方法

移至第一行。

语法

Boolean **moveFirst( )**

返回值

如果成功，则返回 **True**。

如果失败，则返回 **False**。例如，如果没有行，此方法失败。

moveLast 方法

移至最后一行。

语法

Boolean **moveLast( )**

返回值

如果成功，则返回 **True**。

如果失败，则返回 **False**。例如，如果没有行，此方法失败。

moveNext 方法

移至下一行。

语法

Boolean **moveNext( )**

返回值

如果成功，则返回 **True**。

如果失败，则返回 **False**。例如，如果没有行，此方法失败。

movePrevious 方法

移至上一行。

语法

Boolean **movePrevious( )**

返回值

如果成功，则返回 **true**。

如果失败，则返回 **false**。例如，如果没有行，此方法失败。

moveRelative 方法

相对于当前行移动一定数量的行。

语法

Boolean **moveRelative(** Int32 *index* **)**

参数

index　要移动的行数。该值可以为正数、负数或零。

返回值

如果成功，则返回 **true**。

如果失败，则返回 **false**。例如，如果没有行，此方法失败。

注释

相对于游标当前在结果集中的位置，正索引值在结果集中向前移动，负索引值在结果集中向后移动，零不移动游标。

setBoolean 方法

使用 **boolean** 设置指定列的值。

语法

setBoolean(short *columnID*, boolean *value*)

参数

- **columnID**　列的 ID 号。表中第一列的 ID 值为 1。
- **value**　列的新值。

注释

直到执行 **update** 之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setBytes 方法

使用 **byte** 数组设置指定列的值。

语法

setBytes(UInt16 *columnID*, Array *value* **)**

参数

- **columnID**　列的 ID 号。表中第一列的 ID 值为 1。
- **value**　列的新值。

注释

仅适用于 **SQLType.BINARY** 或 **SQLType.LONGBINARY** 类型的列。直到执行 **update** 之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setDate 方法

使用 **Date** 设置指定列的值。

语法

setDate(UInt16 *columnID*, Date *value***)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setDateTime 方法

使用 **Date** 设置指定列的值。

语法

setDateTime(UInt16 *columnID*, Date *value***)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setDouble 方法

使用 **double** 设置指定列的值。

语法

setDouble(UInt16 *columnID*, Double *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setFloat 方法

使用 **float** 设置指定列的值。

语法

setFloat(UInt16 *columnID*, Float *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setInt 方法

使用 Integer 型设置指定列的值。

语法

setInt(UInt16 *columnID*, Int32 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setLong 方法

使用 Int64 设置指定列的值。

语法

setLong(UInt16 *columnID*, Int64 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setNull 方法

将一列设置为 SQL NULL。

语法

setNull(UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

直到执行更新之后，才实际更改数据，而且在提交之前该更改不是永久更改。

setShort 方法

使用 UInt16 型设置指定列的值。

语法

setShort(UInt16 *columnID*, Int16 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setString 方法

使用 String 型设置指定列的值。

语法

setString(UInt16 *columnID*, String *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setTime 方法

使用 Date 型设置指定列的值。

语法

setTime(UInt16 *columnID*, Date *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setTimestamp 方法

使用 Date 型设置指定列的值。

语法

setTimestamp(UInt16 *columnID*, Date *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setULong 方法

使用被视为无符号值的 64 位整数设置指定列的值。

语法

setULong(UInt16 *columnID*, UInt64 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

setUUID 方法

使用 UUID 设置指定列的值。

语法

setUUID(UInt16 *columnID*, UUID *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。仅对类型为 SQLType.BINARY 且长度为 16 的列有效。

另请参见

- “使用 UUID”一节 《MobiLink - 服务器管理》

update 方法

用当前列值（使用 set 方法指定）来更新当前行。

语法

update()

注释

在每次更新之前必须先调用 updateBegin。

updateBegin 方法

准备更新此结果集中的当前行。

语法

updateBegin()

注释

列值通过调用相应的 set*Type* 方法进行修改。

直到执行更新之后，才实际更改数据，而且在提交之前该更改不是永久更改。

ResultSetSchema 类

表示 UltraLite 结果集的模式。

getColumnCount 方法

返回此游标中的列数。

语法

UInt16 **getColumnCount()**;

注释

列 ID 的范围是从 1 到 getColumnCount（含 1 和 getColumnCount）。

列 ID 和计数在模式升级过程中可能发生变化。为了正确地标识列，请按名称访问它，或者在模式升级后刷新高速缓存中的 ID 和计数。

getColumnID 方法

返回指定列的列 ID。

语法

UInt16 **getColumnID(**String *name***)**

参数

- **name** 列的名称。

注释

列 ID 的范围是从 1 到 getColumnCount()（含 1 和 getColumnCount()）。

列 ID 和计数在模式升级过程中可能发生变化。为了正确地标识列，请按名称访问它，或者在模式升级后刷新高速缓存中的 ID 和计数。

getColumnName 方法

返回由指定的列 ID 标识的列的名称。

语法

String **getColumnName(**UInt16 *columnID***)**

参数

- **columnID** 列 ID。**columnID** 必须在 **[1,getColumnCount()]** 范围内。

注释

列 ID 和计数在模式升级过程中可能发生变化。为了正确地标识列，请按名称访问它，或者在模式升级后刷新高速缓存中的 ID 和计数。

getColumnPrecision 方法

返回指定列的精度。

语法

Int32 **getColumnPrecision(**String *name***)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnPrecisionByColID 方法

返回列的精度。

语法

Int32 **getColumnPrecisionByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。结果集中第一列的 ID 值为 1。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnScale 方法

返回列的小数位数。

语法

Int32 **getColumnScale(**String *name***)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnScaleByColID 方法

返回列的小数位数。

语法

Int32 **getColumnScaleByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。结果集中第一列的 ID 值为 1。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnSize 方法

返回指定列的大小。

语法

UInt32 **getColumnSize(**String *name***)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnSizeByColID 方法

返回列的大小。

语法

UInt32 **getColumnSizeByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。结果集中第一列的 ID 值为 1。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnSQLType 方法

返回指定列的 SQL 数据类型。

语法

UInt16 **getColumnSQLType(**String *name***)**

参数

- **name** 列的名称。

getColumnSQLTypeByColID 方法

以 SQLType 枚举的整数的形式返回列的 SQLType。

语法

UInt16 **getColumnSQLTypeByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。结果集中第一列的 ID 值为 1。

isOpen 方法

如果结果集已打开，则返回 **true**，否则返回 **false**。

语法

Boolean **isOpen()**;

SQLError 类

枚举 UltraLite for M-Business Anywhere 可能会报告的 SQL 代码。此类提供静态常量并且无法直接实例化。

成员	说明
SQLE_AGGREGATES_NOT_ALLOWED	请参见“集合函数用法无效”一节 《错误消息》。
SQLE_ALIAS_NOT_UNIQUE	请参见“别名 '%1' 不唯一”一节 《错误消息》。
SQLE_ALIAS_NOT_YET_DEFINED	请参见“别名 '%1' 的定义必须显示在其第一个引用之前”一节 《错误消息》。
SQLE_AMBIGUOUS_INDEX_NAME	请参见“索引名 '%1' 不明确”一节 《错误消息》。
SQLE_BAD_ENCRYPTION_KEY	请参见“加密密钥不正确或遗失”一节 《错误消息》。
SQLE_BAD_PARAM_INDEX	请参见“输入参数索引超出范围”一节 《错误消息》。
SQLE_CANNOT_ACCESS_FILESYSTEM	请参见“无法访问设备上的文件系统”一节 《错误消息》。
SQLE_CANNOT_CHANGE_USER_NAME	请参见“如果上次上载状态未知，则无法更改同步 user_name”一节 《错误消息》。
SQLE_CANNOT_CONVERT	请参见“无效的数据转换”一节 《错误消息》。
SQLE_CANNOT_EXECUTE_STMT	请参见“语句无法执行”一节 《错误消息》。
SQLE_CANNOT_MODIFY	请参见“无法修改表 '%2' 中的列 '%1'”一节 《错误消息》。
SQLE_CLIENT_OUT_OF_MEMORY	请参见“客户端内存不足”一节 《错误消息》。
SQLE_COLUMN_AMBIGUOUS	请参见“在多个表中找到列 '%1' -- 需要相关名”一节 《错误消息》。
SQLE_COLUMN_CANNOT_BE_NULL	请参见“表 '%2' 中的列 '%1' 不能为 NULL”一节 《错误消息》。
SQLE_COLUMN_IN_INDEX	请参见“不能变更索引中的列”一节 《错误消息》。

成员	说明
SQLE_COLUMN_NOT_FOUND	请参见“未找到列 '%1'”一节 《错误消息》。
SQLE_COLUMN_NOT_INDEXED	请参见“列 '%1' 不是其所在表中的任何索引的一部分”一节 《错误消息》。
SQLE_COLUMN_NOT_STREAMABLE	请参见“因为列 '%1' 的类型不支持流操作，所以该操作失败”一节 《错误消息》。
SQLE_COMMUNICATIONS_ERROR	请参见“通信错误”一节 《错误消息》。
SQLE_CONNECTION_ALREADY_EXISTS	请参见“此连接已存在”一节 《错误消息》。
SQLE_CONNECTION_NOT_FOUND	请参见“未找到连接”一节 《错误消息》。
SQLE_CONNECTION_RESTORED	请参见“UltraLite 连接已恢复”一节 《错误消息》。
SQLE_CONSTRAINT_NOT_FOUND	请参见“未找到约束 '%1'”一节 《错误消息》。
SQLE_CONVERSION_ERROR	请参见“无法将 %1 转换为 %2”一节 《错误消息》。
SQLE_COULD_NOT_FIND_FUNCTION	请参见“无法在动态库 '%2' 中找到 '%1'”一节 《错误消息》。
SQLE_COULD_NOT_LOAD_LIBRARY	请参见“无法装载动态库 '%1'”一节 《错误消息》。
SQLE_CURSOR_ALREADY_OPEN	请参见“游标已打开”一节 《错误消息》。
SQLE_CURSOR_NOT_OPEN	请参见“游标未打开”一节 《错误消息》。
SQLE_CURSOR_RESTORED	请参见“UltraLite 游标（或结果集或表）已恢复”一节 《错误消息》。
SQLE_CURSOROP_NOT_ALLOWED	请参见“试图进行非法游标操作”一节 《错误消息》。
SQLE_DATABASE_ERROR	请参见“内部数据库错误 %1 -- 事务已回退”一节 《错误消息》。

成员	说明
SQLE_DATABASE_NAME_REQUIRED	请参见“启动服务器需要数据库名”一节 《错误消息》。
SQLE_DATABASE_NOT_CREATED	请参见“数据库创建失败: %1”一节 《错误消息》。
SQLE_DATATYPE_NOT_ALLOWED	请参见“表达式有不受支持的数据类型”一节 《错误消息》。
SQLE_DBSPACE_FULL	请参见“某一 dbspace 已达到其文件大小的最大值”一节 《错误消息》。
SQLE_DESCRIBE_NONSELECT	请参见“仅能描述一条 SELECT 语句”一节 《错误消息》。
SQLE_DIV_ZERO_ERROR	请参见“除以零”一节 《错误消息》。
SQLE_DOWNLOAD_CONFLICT	请参见“由于与现有行发生冲突，下载失败”一节 《错误消息》。
SQLE_DOWNLOAD_RESTART_FAILED	请参见“无法重试下载，因为上载没有完成”一节 《错误消息》。
SQLE_DROP_DATABASE_FAILED	请参见“试图删除数据库 '%1' 失败”一节 《错误消息》。
SQLE_DUPLICATE_CURSOR_NAME	请参见“游标名 '%1' 已经存在”一节 《错误消息》。
SQLE_DUPLICATE_FOREIGN_KEY	请参见“表 '%2' 的外键 '%1' 与现有外键重复”一节 《错误消息》。
SQLE_DUPLICATE_OPTION	请参见“选项 '%1' 被指定多次”一节 《错误消息》。
SQLE_DYNAMIC_MEMORY_EXHAUSTED	请参见“动态内存已耗尽”一节 《错误消息》。
SQLE_ENCRYPTION_INITIALIZATION_FAILED	请参见“无法初始化加密 DLL: '%1'”一节 《错误消息》。
SQLE_ENGINE_ALREADY_RUNNING	请参见“数据库服务器已在运行”一节 《错误消息》。
SQLE_ENGINE_NOT_MULTIUSER	请参见“数据库服务器未在多用户模式下运行”一节 《错误消息》
SQLE_ERROR	请参见“运行时 SQL 错误 -- %1”一节 《错误消息》。

成员	说明
SQLE_ERROR_CALLING_FUNCTION	请参见“无法分配资源来调用外部函数”一节 《错误消息》。
SQLE_ERROR_IN_ASSIGNMENT	请参见“赋值出错”一节 《错误消息》。
SQLE_EXPRESSION_ERROR	请参见“'%1' 附近的表达式无效”一节 《错误消息》。
SQLE_FEATURE_NOT_ENABLED	请参见“未对您的应用程序启用您曾尝试调用的方法”一节 《错误消息》。
SQLE_FILE_BAD_DB	请参见“无法启动指定的数据库:'%1' 不是有效的数据库文件”一节 《错误消息》。
SQLE_FILE_IN_USE	请参见“指定的数据库文件已在使用”一节 《错误消息》。
SQLE_FILE_NOT_DB	请参见“无法启动指定的数据库:'%1' 不是数据库”一节 《错误消息》。
SQLE_FILE_VOLUME_NOT_FOUND	请参见“找不到为数据库 '%1' 指定的文件系统卷”一节 《错误消息》。
SQLE_FILE_WRONG_VERSION	请参见“无法启动指定的数据库:'%1' 由本软件的另一版本创建”一节 《错误消息》。
SQLE_FOREIGN_KEY_NAME_NOT_FOUND	请参见“未找到外键名 '%1'”一节 《错误消息》。
SQLE_IDENTIFIER_TOO_LONG	请参见“标识符 '%1' 过长”一节 《错误消息》。
SQLE_INCORRECT_VOLUME_ID	请参见“'%1' 的卷 ID 不正确”一节 《错误消息》。
SQLE_INDEX_NAME_NOT_UNIQUE	请参见“索引名 '%1' 不唯一”一节 《错误消息》。
SQLE_INDEX_NOT_FOUND	请参见“无法找到名为 '%1' 的索引”一节 《错误消息》。
SQLE_INDEX_NOT_UNIQUE	请参见“表 '%2' 的索引 '%1' 将不唯一”一节 《错误消息》。
SQLE_INTERRUPTED	请参见“语句被用户中断”一节 《错误消息》。
SQLE_INVALID_CONSTRAINT_REF	请参见“对约束 '%1' 的引用或操作无效”一节 《错误消息》。

成员	说明
SQLE_INVALID_DESCRIPTOR_INDEX	请参见“无效的描述符索引”一节 《错误消息》。
SQLE_INVALID_DESCRIPTOR_NAME	请参见“无效的 SQL 描述符名”一节 《错误消息》。
SQLE_INVALID_DISTINCT_AGGREGATE	请参见“分组查询包含多个不同的集合函数”一节 《错误消息》。
SQLE_INVALID_FOREIGN_KEY	请参见“表 '%2' 中的外键 '%1' 没有主键值”一节 《错误消息》。
SQLE_INVALID_FOREIGN_KEY_DEF	请参见“外键的列 '%1' 与主键定义不同”一节 《错误消息》。
SQLE_INVALID_GROUP_SELECT	请参见“对 '%1' 的函数或列引用还必须出现在 GROUP BY 中”一节 《错误消息》。
SQLE_INVALID_LOGON	请参见“无效的用户 ID 或口令”一节 《错误消息》。
SQLE_INVALID_OPTION_SETTING	请参见“选项 '%1' 的设置无效”一节 《错误消息》。
SQLE_INVALID_OPTION_VALUE	请参见“'%1' 是 '%2' 的无效值”一节 《错误消息》。
SQLE_INVALID_ORDER	请参见“ORDER BY 说明无效”一节 《错误消息》。
SQLE_INVALID_PARAMETER	请参见“无效的参数”一节 《错误消息》。
SQLE_INVALID_PARSE_PARAMETER	请参见“分析错误: %1”一节 《错误消息》。
SQLE_INVALID_SQL_IDENTIFIER	请参见“无效的 SQL 标识符”一节 《错误消息》。
SQLE_INVALID_UNION	请参见“UNION、INTERSECT 或 EXCEPT 中的 SELECT 列表长度不匹配”一节 《错误消息》。
SQLE_KEYLESS_ENCRYPTION	请参见“无法执行请求的操作，因为此数据库使用无密钥加密”一节 《错误消息》。
SQLE_LOCKED	请参见“用户 '%1' 锁定了 '%2' 中的行”一节 《错误消息》。
SQLE_MEMORY_ERROR	请参见“内存错误 -- 事务已回退”一节 《错误消息》。

成员	说明
SQLE_NAME_NOT_UNIQUE	请参见“项 '%1' 已经存在”一节 《错误消息》。
SQLE_NO_COLUMN_NAME	请参见“派生表 '%1' 没有列 %2 的名称”一节 《错误消息》。
SQLE_NO_CURRENT_ROW	请参见“没有当前的游标行”一节 《错误消息》。
SQLE_NO_INDICATOR	请参见“未给 NULL 结果提供指示符变量”一节 《错误消息》。
SQLE_NO_MATCHING_SELECT_ITEM	请参见“派生表 '%1' 的选择列表没有与 '%2' 匹配的表达式”一节 《错误消息》。
SQLE_NO_PRIMARY_KEY	请参见“表 '%1' 无主键”一节 《错误消息》。
SQLE_NOERROR	SQLE_NOERROR(0) - 此代码表示没有错误或警告。
SQLE_NON_UPDATEABLE_COLUMN	请参见“不能更新表达式”一节 《错误消息》。
SQLE_NON_UPDATEABLE_CURSOR	请参见“为 READ ONLY 游标指定了不正确的 FOR UPDATE”一节 《错误消息》。
SQLE_NOT_IMPLEMENTED	请参见“未实现功能 '%1'”一节 《错误消息》。
SQLE_NOT_SUPPORTED_IN_ULTRALITE	请参见“在 UltraLite 中功能不可用”一节 《错误消息》。
SQLE_NOTFOUND	请参见“未找到行”一节 《错误消息》。
SQLE_ONLY_ONE_TABLE	请参见“游标上的 INSERT/DELETE 只能修改一个表”一节 《错误消息》。
SQLE_OVERFLOW_ERROR	请参见“值 %1 超出了目标的范围”一节 《错误消息》。
SQLE_PAGE_SIZE_INVALID	请参见“无效的数据库页面大小”一节 《错误消息》。
SQLE_PARTIAL_DOWNLOAD_NOT_FOUND	请参见“未找到部分下载”一节 《错误消息》。
SQLE_PERMISSION_DENIED	请参见“权限被拒绝: %1”一节 《错误消息》。
SQLE_PRIMARY_KEY_NOT_UNIQUE	请参见“表 '%1' 的主键不唯一:主键值 ('%2')”一节 《错误消息》。
SQLE_PRIMARY_KEY_TWICE	请参见“表不能有两个主键”一节 《错误消息》。

成员	说明
SQLE_PRIMARY_KEY_VALUE_REF	请参见“表 '%1' 中行的主键被表 '%3' 中的外键 '%2' 引用”一节 《错误消息》。
SQLE_PUBLICATION_NOT_FOUND	请参见“未找到发布 '%1'”一节 《错误消息》。
SQLE_PUBLICATION_PREDICATE_IGNORED	请参见“发布谓语未被评估”一节 《错误消息》。
SQLE_RESOURCE_GOVERNOR_EXCEEDED	请参见“超出 '%1' 的资源调控器”一节 《错误消息》。
SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY	请参见“为了保持参照完整性，已从表 %1 中删除行”一节 《错误消息》。
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	请参见“当前不允许模式升级”一节 《错误消息》。
SQLE_SERVER_SYNCHRONIZATION_ERROR	请参见“由于服务器出错，同步失败: %1”一节 《错误消息》。
SQLE_START_STOP_DATABASE_DENIED	请参见“启动/停止数据库的请求被拒绝”一节 《错误消息》。
SQLE_STATEMENT_ERROR	请参见“SQL 语句错误”一节 《错误消息》。
SQLE_STRING_RIGHT_TRUNCATION	请参见“字符串数据右截断”一节 《错误消息》。
SQLE_SUBQUERY_SELECT_LIST	请参见“子查询只允许一个选择列表项”一节 《错误消息》。
SQLE_SYNC_INFO_INVALID	请参见“同步信息不完整或无效，请检查 '%1'”一节 《错误消息》。
SQLE_SYNC_INFO_REQUIRED	请参见“未提供同步信息”一节 《错误消息》。
SQLE_SYNC_NOT_REENTRANT	请参见“同步进程无法重新进入同步”一节 《错误消息》。
SQLE_SYNC_STATUS_UNKNOWN	请参见“上次同步上载的状态未知”一节 《错误消息》。
SQLE_SYNTAX_ERROR	请参见“第 %2 行的 '%1' 附近有语法错误”一节 《错误消息》。

成员	说明
SQLE_TABLE_ALREADY_INCLUDED	请参见“已包括表 '%1'”一节 《错误消息》。
SQLE_TABLE_IN_USE	请参见“表正在使用”一节 《错误消息》。
SQLE_TABLE_NOT_FOUND	请参见“未找到表 '%1'”一节 《错误消息》。
SQLE_TOO_MANY_BLOB_REFS	请参见“对 BLOB 的引用太多”一节 《错误消息》。
SQLE_TOO_MANY_CONNECTIONS	请参见“超出数据库服务器连接限制”一节 《错误消息》。
SQLE_TOO_MANY_PUBLICATIONS	请参见“为操作指定的发布过多”一节 《错误消息》。
SQLE_TOO_MANY_TEMP_TABLES	请参见“连接中临时表过多”一节 《错误消息》。
SQLE_TOO_MANY_USERS	请参见“数据库中的用户太多”一节 《错误消息》。
SQLE_ULTRALITE_DATABASE_NOT_FOUND	请参见“未找到数据库 '%1'”一节 《错误消息》。
SQLE_ULTRALITE_OBJ_CLOSED	请参见“在已关闭的对象上的操作无效”一节 《错误消息》。
SQLE_ULTRALITE_WRITE_ACCESS_DENIED	请参见“写访问被拒绝”一节 《错误消息》。
SQLE_UNABLE_TO_CONNECT	请参见“无法启动数据库 -- %1”一节 《错误消息》。
SQLE_UNABLE_TO_START_DATABASE	请参见“无法启动指定的数据库: %1”一节 《错误消息》。
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	请参见“无法启动指定的数据库:必须升级服务器才能启动数据库 %1”一节 《错误消息》。
SQLE_UNCOMMITTED_TRANSACTIONS	请参见“不能同步或升级未提交的事务”一节 《错误消息》。
SQLE_UNKNOWN_FUNC	请参见“未知函数 '%1'”一节 《错误消息》。
SQLE_UNKNOWN_OPTION	请参见“'%1' 是未知选项”一节 《错误消息》。
SQLE_UNKNOWN_USERID	请参见“用户 ID '%1' 不存在”一节 《错误消息》。

成员	说明
SQLE_UNRECOGNIZED_OPTION	请参见“无法识别选项 '%1'”一节 《错误消息》。
SQLE_UPLOAD_FAILED_AT_SERVER	请参见“同步服务器无法提交上载”一节 《错误消息》。
SQLE_VALUE_IS_NULL	请参见“无法将 NULL 结果作为所需数据类型返回”一节 《错误消息》。
SQLE_VARIABLE_INVALID	请参见“无效的主机变量”一节 《错误消息》。
SQLE_WRONG_NUM_OF_INSERT_COLS	请参见“INSERT 的值数目错误”一节 《错误消息》。
SQLE_WRONG_PARAMETER_COUNT	请参见“函数 '%1' 的参数数目错误”一节 《错误消息》。

SQLType 类

此枚举以常量形式列出用作表列类型的可用 UltraLite SQL 数据库类型。

常量	UltraLite 数据库类型
BAD_INDEX	
S_LONG	INT
U_LONG	UNSIGNED INT
S_SHORT	SMALLINT
U_SHORT	UNSIGNED SMALLINT
S_BIG	BIGINT
U_BIG	UNSIGNED BIGINT
TINY	TINYINT
BIT	BIT
TIMESTAMP	TIMESTAMP
DATE	DATE
TIME	TIMESTAMP
DOUBLE	DOUBLE
REAL	REAL
NUMERIC	NUMERIC
BINARY	BINARY
CHAR	CHAR 或 VARCHAR
LONGVARCHAR	LONG VARCHAR
LONGBINARY	LONG BINARY
MAX_INDEX	

toString 方法

返回指定的 SQL 列类型常量的字符串名称；如果类型不可识别，则返回 BAD_SQL_TYPE。

语法

String **toString(**UInt16 *code***)**

参数

- **code** SQL 列类型常量。

SyncParms 类

表示同步参数，这些参数用于定义如何同步 UltraLite 数据库。每个连接都有自己的 SyncParms 实例。

常量

常量	值	说明
STREAM_TYPE_TCPIP	0	TCP/IP 流
STREAM_TYPE_HTTP	1	HTTP 流
STREAM_TYPE_HTTPS	2	HTTPS 同步
STREAM_TYPE_TLS	3	TLS 同步
STREAM_TYPE_HOTSYNC	4	用于 HotSync 同步

getAdditionalParms 方法

返回名称值对的字符串。

语法

String **getAdditionalParms()**

另请参见

- “Additional Parameters 同步参数”一节 《UltraLite - 数据库管理和参考》

getAuthenticationParms 方法

返回提供给自定义用户验证脚本的参数，如果没有指定参数，则返回空值。

语法

Array **getAuthenticationParms()**

getDownloadOnly 方法

如果禁用了上载，则返回 true；如果启用了上载，则返回 false。

语法

Boolean **getDownloadOnly()**

getKeepPartialDownload 方法

如果要保存部分下载，则返回 true；如果要回退部分下载，则返回 false。

语法

Boolean getKeepPartialDownload()

getNewPassword 方法

返回在下次同步后与 MobiLink 用户相关联的新口令。

语法

String **getNewPassword()**

getPartialDownloadRetained 方法

如果由于通信错误而导致下载失败并且保留了部分下载，则返回 true；如果下载没有中断或者如果回退了部分下载，则返回 false。

语法

Boolean **getPartialDownloadRetained( )**

getPassword 方法

返回用 setUserName 指定的用户的 MobiLink 口令。

语法

String **getPassword()**;

getPingOnly 方法

如果客户端仅 ping 服务器，则返回 true；如果客户端执行同步，则返回 false。

语法

Boolean **getPingOnly()**

getResumePartialDownload 方法

如果要恢复以前的部分下载，则返回 true；如果要回退以前的部分下载，则返回 false。

语法

Boolean **getResumePartialDownload( )**

getSendColumnNames 方法

如果客户端在同步过程中向 MobiLink 服务器发送列名，则返回 true；如果客户端不发送列名，则返回 false。

语法

Boolean **getSendColumnNames()**

getSendDownloadAck 方法

如果客户端向 MobiLink 服务器提供下载确认，则返回 true；如果客户端不提供下载确认，则返回 false。

语法

Boolean **getSendDownloadAck()**

getStream 方法

返回用于同步的 MobiLink 同步流的类型。

语法

UInt16 **getStream()**;

getStreamParms 方法

返回包含用于同步流的所有网络协议选项的一个字符串。

语法

String **getStreamParms()**;

getUploadOnly 方法

如果禁用了下载，则返回 true；如果启用了下载，则返回 false

语法

Boolean **getUploadOnly()**

getUserName 方法

返回 MobiLink 用户名。

语法

String **getUserName()**

getVersion 方法

返回指示要使用哪一个同步脚本的版本字符串。

语法

String **getVersion()**

setAdditionalParms 方法

指定 "关键字=值" 对的字符串。此 "关键字=值" 对的列表通常包含不经常使用的同步参数。

语法

getAdditionalParms(String *additional_params***)**

参数

- **additional_params** "关键字=值" 对的字符串。

另请参见

- “Additional Parameters 同步参数”一节 《UltraLite - 数据库管理和参考》

setAuthenticationParms 方法

为自定义用户验证脚本（MobiLink authenticate_parameters 连接事件）指定参数。

语法

setAuthenticationParms(Array *value* **)**

参数

- **value** 字符串数组，每个字符串都包含一个验证参数（空数组项将导致同步错误）。

注释

只使用前 255 个字符串并且每个字符串的长度不应超过 128 个字符（多余的字符串在发送到 MobiLink 时会被截断）。

setDownloadOnly 方法

指定是否在同步时禁用或启用上载。

语法

setDownloadOnly(Boolean *value* **)**

参数

- **value** 如果禁用上载，则设置为 true；如果启用上载，则设置为 false。

setKeepPartialDownload 方法

指定在同步时是禁用还是启用部分下载。

语法

setKeepPartialDownload(Boolean *value* **)**

参数

- **value** 如果同步时启用对部分下载的保存，则设置为 true；如果放弃部分下载，则设置为 false。

注释

UltraLite 能够重新启动由于通信错误而失败的下载。UltraLite 在接收下载的过程中对其进行处理。如果下载被中断，则部分下载事务仍保留在数据库中，并且可在下一次同步过程中恢复。

要指示 UltraLite 应保存部分下载，请指定 Connection.syncParms.setKeepPartialDownload(true)；否则当发生错误时将回退下载。

如果保留了部分下载，则当 connection.synchronize 退出时，输出字段 connection.SyncResult.getPartialDownloadRetained 将设置为 true。如果设置了 getPartialDownloadRetained，则可以继续下载。为此，使用 connection.syncParms.setResumePartialDownload(true) 调用 connection.synchronize。可能仍需要将 KeepPartialDownload 设置为 true，以免发生其它通信错误。如果跳过某一下载，将不执行任何上载。

您在恢复下载期间接收的下载将与下载最初开始时一样旧。如果您需要最新数据，则可以在专门的恢复下载完成后再立即进行一次下载。

在恢复下载时，许多 SyncParms 字段是无关的字段。您会收到在最初下载时请求的发布。只需要设置字段 setResumePartialDownload(boolean) 和 setUserName(String)。如果您愿意，也可以设置字段

setKeepPartialDownload(boolean)、setDownloadOnly(boolean) 和 setDisableConcurrency(boolean)，三个字段都会正常工作。

如果不再需要现有的部分下载，可以调用 Connection.rollbackPartialDownload 回退失败的下载事务。此外，如果您尝试再次同步但没有指定 ResumePartialDownload，则部分下载在下次同步开始之前将回退。

请参见“如何处理同步失败”一节 《MobiLink - 入门》。

setMBAServer 方法

提供一种快速方法将 MobiLink 主机和端口同步参数设置为 M-Business 客户端使用的 M-Business Anywhere 服务器同步参数。

语法

setMBAServer(String *host*, String *port*, String *url_suffix* **)**

参数

- **host** M-Business Anywhere 服务器的主机或 IP 值。如果主机为空值，UltraLite 会将它设置为当前的 M-Business Anywhere 主机。
- **port** M-Business Anywhere 服务器监听的端口。如果端口为空值，UltraLite 会将它设置为当前的 M-Business Anywhere 端口值。
- **url_suffix** 它与 M-Business Anywhere 的 *sync.conf* 文件中设置的 url_suffix 参数对应。

注释

使用 M-Business Anywhere 的 MobiLink 重定向器向 MobiLink 服务器和从 MobiLink 服务器路由数据。

如果您使用的是一键式同步，您必须使用 Connection.saveSyncParms 保存同步参数。

有关配置 M-Business 服务器以通过 M-Business Anywhere 重定向器路由 HTTP 数据库通信的信息，请参见“M-Business Anywhere 重定向器（不建议使用）”一节 《MobiLink - 服务器管理》。

setMBAServerWithMoreParms 方法

提供一种快速方法将 MobiLink 主机和端口同步参数设置为 M-Business 客户端使用的 M-Business Anywhere 服务器同步参数。

语法

setMBAServerWithMoreParms(String *host*, String *port*, String *url_suffix*, String *additional***)**

参数

- **host** M-Business Anywhere 服务器的主机或 IP 值。如果主机为空值，UltraLite 会将它设置为当前的 M-Business Anywhere 主机。

- **port** M-Business Anywhere 服务器监听的端口。如果端口为空值，UltraLite 会将它设置为当前的 M-Business Anywhere 端口值。
- **url_suffix** 它与 M-Business Anywhere 的 *sync.conf* 文件中设置的 url_suffix 参数对应。
- **additional** 此参数可能包含先前的参数未包括的附加流参数（如代理主机、代理端口或与安全相关的参数）。如果您需要指定主机、端口或 url_suffix 信息，您可以使用上一节介绍的 setMBAServer 方法。

注释

使用 M-Business Anywhere 的 MobiLink 重定向器向 MobiLink 服务器和从 MobiLink 服务器路由数据。

此方法允许用户用附加参数指定其它参数，从而扩展了 setMBAServer 提供的功能。

如果您使用的是一键式同步，您必须使用 Connection.saveSyncParms 保存同步参数。

有关配置 M-Business 服务器以通过 M-Business Anywhere 重定向器路由 HTTP 数据库通信的信息，请参见“M-Business Anywhere 重定向器（不建议使用）”一节 《MobiLink - 服务器管理》。

setNewPassword 方法

为用 setUserName 指定的用户设置一个新的 MobiLink 口令。

语法

setNewPassword(String *value* **)**

参数

- **value** MobiLink 用户的新口令。

注释

新口令将在下次同步之后生效。

setPassword 方法

设置用 setUserName 指定的用户的 MobiLink 口令。

语法

setPassword(String *value* **)**

参数

- **value** MobiLink 用户的口令。

注释

此用户名和口令独立于任何数据库用户 ID 和口令，它们用于向 MobiLink 服务器标识和验证应用程序。

setPingOnly 方法

指定客户端是否应当仅 ping MobiLink 服务器，而不实际执行同步。

语法

setPingOnly(Boolean *value* **)**;

参数

- **value** 如果仅 ping MobiLink 服务器，则设置为 true；如果执行同步，则设置为 false。

setSendColumnNames 方法

指定客户端是否应在同步过程中向 MobiLink 服务器发送列名。

语法

setSendColumnNames(Boolean *value* **)**

参数

- **value** 设置为 true 时，发送列名称；设置为 false 时，不发送列名称。

注释

此参数用于直接行处理。

setSendDownloadAck 方法

指定客户端是否应在同步过程中向 MobiLink 服务器发送下载确认消息。

语法

setSendDownloadAck(Boolean *value* **)**

参数

- **value** 如果发送下载确认（正数或负数），则设置为 true；如果告知服务器不发送下载确认，则设置为 false。

注释

下载确认会在下载在远程完全应用并提交后发送（正确认），或者是在下载失败后发送（负确认）。

如果客户端确要发送下载确认，则 MobiLink 服务器数据库工作线程必须等待客户端应用并提交下载。如果客户端不发送下载确认，则可更快地释放 MobiLink 服务器以进行下一次同步。

setStream 方法

设置用于同步的 MobiLink 同步流。

语法

setStream(UInt16 *value* **)**

参数

- **value** 用于同步的 MobiLink 同步流的类型。有关有效选择的列表，请参见“常量”一节 第 128 页。

注释

大多数同步流都需要用一些参数来标识 MobiLink 服务器地址和控制其它行为。这些参数是随 **setStreamParms()** 方法提供的。

缺省流类型是 STREAM_TYPE_TCPIP。

setStreamParms 方法

设置用于配置同步流的参数。

语法

setStreamParms(String *value* **)**

参数

- **value** 包含用于同步流的所有网络协议选项的字符串。选项是以以分号分隔的 [名称=值] 对列表 (**"param1=value1;param2=value2"**) 的形式指定的。

注释

有关配置特定流类型的信息，请参见“UltraLite 同步流的网络协议选项”一节 《UltraLite - 数据库管理和参考》。

setUploadOnly 方法

指定在同步时是禁用还是启用下载。

语法

setUploadOnly(Boolean *value* **)**

参数

- **value** 如果禁用下载，则设置为 true；如果启用下载，则设置为 false。

setUserName 方法

设置向 MobiLink 服务器唯一标识 MobiLink 客户端的用户名。

语法

setUserName(String *value* **)**

参数

- **value** MobiLink 用户名。

注释

MobiLink 使用此值确定下载内容、记录同步状态并在同步期间发生中断时进行恢复。此用户名和口令独立于任何数据库用户 ID 和口令，它们用于向 MobiLink 服务器标识和验证应用程序。

setVersion 方法

指定要使用哪一个同步脚本版本。

语法

setVersion(String *value* **)**

参数

- **value** 脚本版本字符串。

注释

统一数据库中的每个同步脚本都带有版本字符串标记。例如，可能有两个不同的 download_cursor 脚本，分别用不同的版本字符串来标识。版本字符串使 UltraLite 应用程序可以从一组同步脚本中进行选择。

SyncResult 类

表示上一次同步时 UltraLite for M-Business Anywhere 方法的状态。每个连接都有自己的 SyncResult 实例。

无法直接实例化此类。

getAuthStatus 方法

返回上次同步尝试的授权状态码。

语法

UInt16 **getAuthStatus()**

getIgnoredRows 方法

如果最近一次同步过程中忽略了任何已上载的行，则返回 true；如果未忽略已上载的行，则返回 false。

语法

Boolean **getIgnoredRows()**

参数

- **return** 如果忽略了任何已上载的行，则返回真；如果未忽略已上载的行，则返回假。

getPartialDownloadRetained 方法

如果下载中断并且保留了部分下载，则返回 true；如果下载没有中断或者如果回退了部分下载，则返回 false。

语法

Boolean getPartialDownloadRetained()

getStreamErrorCode 方法

返回一个整数，该整数表示由同步流处理报告的错误代码。

语法

UInt16 **getStreamErrorCode( )**

参数

- **return** 同步流报告的错误代码。

注释

请参见“MobiLink 通信错误消息” 《错误消息》。

getStreamErrorParameters 方法

返回以逗号分隔的流错误参数列表。

语法

String **StreamErrorParameters ( )**

参数

- **return** 由同步流处理报告的错误参数列表（以逗号分隔）。

另请参见

- “getStreamErrorCode 方法”一节第 138 页

getStreamErrorSystem 方法

返回流错误的系统特定代码。

语法

UInt16 **getStreamErrorSystem( )**

参数

- **return** 系统特定的错误代码。

getTimestamp 方法

返回最近一次同步的时间戳。

语法

Date **getTimestamp( )**

getUploadOK 方法

如果上次上载同步成功，则返回 true；如果上次上载同步不成功，则返回 false。

语法

Boolean **getUploadOK()**

TableSchema 类

表示 UltraLite 表的模式。

getColumnCount 方法

返回此表中从 1 开始的列号。

语法

UInt16 **getColumnCount( )**

注释

列 ID 的范围是从 1 到 getColumnCount()。

getColumnDefaultValue 方法

返回指定列的缺省值；如果缺省值为 **null**，则返回空值。

语法

String **getColumnDefaultValue(** String *name* **)**

参数

- **name** 列的名称。

getColumnDefaultValueByColID 方法

返回列的缺省值；如果缺省值为 **null**，则返回空值。

语法

String **getColumnDefaultValueByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

getColumnID 方法

返回指定列从 1 开始的 ID。

语法

UInt16 **getColumnID(** String *name* **)**

参数

- **name** 列的名称。

getColumnName 方法

返回指定列的名称。

语法

String **getColumnName(** UInt16 *colID* **)**

参数

- **colID** 列从 1 开始的列 ID。

getColumnPartitionSize 方法

返回列的全局自动增量分区大小（由 Double 型表示的 64 位无符号数字形式）。

语法

UInt64 **getColumnPartitionSize(** String *name* **)**

参数

- **name** 列的名称。

注释

给定表中的所有全局自动增量列都共享同一全局自动增量分区。

getColumnPartitionSizeByColID 方法

返回列的全局自动增量分区大小（由 Double 型表示的 64 位无符号数字形式）。

语法

UInt64 **getColumnPartitionSizeByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

给定表中的所有全局自动增量列都共享同一全局自动增量分区。

getColumnPrecision 方法

返回列的精度。

语法

Int32 **getColumnPrecision(** String *name* **)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnPrecisionByColID 方法

返回列的精度。

语法

Int32 **getColumnPrecisionByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

此列必须为 SQLType.NUMERIC 类型

getColumnScale 方法

返回列的小数位数。

语法

Int32 **getColumnScale(** String *name* **)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnScaleByColID 方法

返回列的小数位数。

语法

Int32 **getColumnScaleByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

此列必须为 SQLType.NUMERIC 类型。

getColumnSize 方法

返回列的大小。

语法

UInt32 **getColumnSize(** String *name* **)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.BINARY 或 SQLType.CHAR 类型。

getColumnSizeByColID 方法

返回列的大小。

语法

UInt32 **getColumnSizeByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

此列必须为 SQLType.BINARY 或 SQLType.CHAR 类型。

getColumnSQLType 方法

以 SQLType 枚举的整数的形式返回列的 SQLType。

语法

Int16 **getColumnSQLType(** String *name* **)**

参数

- **name** 列的名称。

getColumnSQLTypeByColID 方法

以 SQLType 枚举的整数的形式返回列的 SQLType。

语法

Int16 **getColumnSQLTypeByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

getIndex 方法

返回指定索引的索引模式。

语法

IndexSchema **getIndex(** String *name* **)**

参数

- **name** 索引的名称。

getIndexCount 方法

返回此表的索引数。

语法

UInt16 **getIndexCount( )**

注释

索引 ID 的范围是 1 到 **getIndexCount()**（含边界值）。

注意：索引 ID 和计数在模式升级过程中可能发生变化。为了正确地标识索引，请按名称访问它，或者在模式升级后刷新高速缓存中的 ID 和计数。

getIndexName 方法

返回由指定的索引 ID 标识的索引的名称。

语法

String **getIndexName(** UInt16 *indexID* **)**

参数

- **indexID** 索引 ID。**indexID** 必须在 **[1,getIndexCount()]** 范围内。

注释

注意：索引 ID 和计数在模式升级过程中可能发生变化。为了正确地标识索引，请按名称访问它，或者在模式升级后刷新高速缓存中的 ID 和计数。

getName 方法

返回此表的名称。

语法

String **getName( )**

getOptimalIndex 方法

返回用于使用指定列搜索表的最佳索引。

语法

IndexSchema **getOptimalIndex(** String *name* **)**

参数

- **name** 列的名称。

注释

指定的列是索引中的第一列，而索引可以有多列。

getPrimaryKey 方法

返回此表的主键的索引模式。

语法

IndexSchema **getPrimaryKey( )**

getUploadUnchangedRows 方法

如果表被标记为上载所有行，则返回 true；如果表未被标记为上载所有行，则返回 false。

语法

Boolean **getUploadUnchangedRows( )**

注释

当对表进行同步时，此方法为其返回 true 的表将始终上载未更改和已更改的行。这些表有时称为 "全部同步" 表。

isColumnAutoIncrement 方法

如果列自动增量，则返回 true；否则返回 false。

语法

Boolean **isColumnAutoIncrement(** String *name* **)**

参数

- **name** 列的名称。

isColumnAutoIncrementByColID 方法

如果列自动增量，则返回 true；否则返回 false。

语法

Boolean **isColumnAutoIncrementByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

isColumnCurrentDate 方法

如果列缺省为当前日期，则返回 true；否则返回 false。

语法

Boolean **isColumnCurrentDate(** String *name* **)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.DATE 类型。

isColumnCurrentDateByColID 方法

如果列缺省为当前日期，则返回 true；否则返回 false。

语法

Boolean **isColumnCurrentDateByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

此列必须为 SQLType.DATE 类型。

isColumnCurrentTime 方法

如果列缺省为当前时间，则返回 true；否则返回 false。

语法

Boolean **isColumnCurrentTime(** String *name* **)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.TIME 类型。

isColumnCurrentTimeByColID 方法

如果列缺省为当前时间，则返回 true；否则返回 false。

语法

Boolean **isColumnCurrentTimeByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

此列必须为 SQLType.TIME 类型。

isColumnCurrentTimestamp 方法

如果列缺省为当前时间戳，则返回 true；否则返回 false。

语法

Boolean **isColumnCurrentTimestamp(** String *name* **)**

参数

- **name** 列的名称。

注释

此列必须为 SQLType.TIMESTAMP 类型。

isColumnCurrentTimestampByColID 方法

如果列缺省为当前时间戳，则返回 true；否则返回 false。

语法

Boolean **isColumnCurrentTimestampByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

此列必须为 SQLType.TIME 类型。

isColumnGlobalAutoIncrement 方法

如果列缺省为全局自动增量，则返回 true；否则返回 false。

语法

Boolean **isColumnGlobalAutoIncrement(** String *name* **)**

参数

- **name** 列的名称。
- **return** 如果列全局自动递增，则返回 true；如果不全局自动递增，则返回 false。

isColumnGlobalAutoincrementByColID 方法

如果列缺省为全局自动增量，则返回 true；否则返回 false。

语法

Boolean **isColumnGlobalAutoincrementByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

isColumnNewUUID 方法

如果列缺省为一个新的 UUID，则返回 true；否则返回 false。

语法

Boolean **isColumnNewUUID(** String *name* **)**

参数

- **name** 列的名称。

isColumnNewUUIDByColID 方法

如果列缺省为一个新的 UUID，则返回 true；否则返回 false。

语法

Boolean **isColumnNewUUIDByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

isColumnNullable 方法

如果列可以为空，则返回 true；否则返回 false。

语法

Boolean **isColumnNullable(** String *name* **)**

参数

- **name** 列的名称。

isColumnNullableByColID 方法

如果列可以为空，则返回 true；否则返回 false。

语法

Boolean **isColumnNullableByColID(** UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

isInPublication 方法

如果表在发布中，则返回 True；如果表不在发布中，则返回 false。

语法

Boolean **isInPublication(** String *pubName* **)**

参数

- **pubName** 发布的名称。

isNeverSynchronized 方法

如果表被标记为永不同步，则返回 true；如果表未被标记为永不同步，则返回 false。

语法

Boolean **isNeverSynchronized( )**

注释

此方法为其返回 true 的表将永不同步，即使这些表包含在发布中。这些表有时称为 "不同步" 表。

ULTable 类

表示一个 UltraLite 表。

属性

这里列出了该类的属性。

属性	说明
TableSchema schema（只读）	此结果集的模式。只有在其预准备语句处于打开的情况下此属性才有效。
NULL_TIMESTAMP_VAL	指示时间戳值为 NULL 的常量。

appendBytes 方法

将指定的字节数组的指定子集添加到指定的 SQLType.LONGBINARY 列的新值中。

语法

```
appendBytes(
  UInt16 columnID,
  Array value,
  UInt32 srcOffset,
  UInt32 count
)
```

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。
- **srcOffset** 要附加到列的当前新值中的值。
- **count** 要复制的字节数。

注释

数组 **value** 从 srcOffset（从 0 开始）位置到 **srcOffset+count-1** 位置的字节会添加到指定列的值中。在插入时，insertBegin 将新值初始化为列的缺省值。直到执行 **insert** 之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

如果以下任何一个条件为 true，则抛出代码为 SQLCode.SQLE_INVALID_PARAMETER 的错误，并且不会修改目标：

- **value** 参数为空值。
- **srcOffset** 参数为负值。

- **count** 参数为负值。
- **srcOffset+count** 大于源数组的长度 **value.length**。

对于其它错误，抛出具有相应错误代码的 **SQLException**。

appendStringChunk 方法

将指定字符串添加到指定的 SQLType.LONGVARCHAR 列的新值中。

语法

appendStringChunk(
  UInt16 *columnID*,
  String *value*
)

参数

- **columnID**　列的 ID 号。表中第一列的 ID 值为 1。
- **value**　列的新值。

示例

以下语句将字符串 **XYZ** 的一百个实例添加到第一列的值中：

```
for ( I = 0; I < 100; I++ ){
  t.appendStringChunk( 1, "XYZ" );
}
```

deleteRow 方法

删除当前行。

语法

deleteRow( )

deleteAllRows 方法

删除表中的所有行。

语法

deleteAllRows( )

注释

在某些应用程序中，它可用于在将一组新数据下载到表中之前删除表中的所有行。使用 Connection.startSynchronizationDelete 方法可以将行从 UltraLite 数据库删除，但不从统一数据库删除。

findBegin 方法

准备在此表上执行新的查找。

语法

findBegin( )

注释

要搜索的值是通过对用于打开此表的索引中的列调用相应的 set*Type* 方法来指定的。

findFirst 方法

在表中从开头往前移动，查找与当前索引中的一个值或整个值集完全匹配的行。

语法

Boolean **findFirst( )**

返回值

如果成功，则返回 **true**；否则，返回 **false**

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在第一个与索引值完全匹配的行上。失败时，游标位置在最后一行之后 (isEOF)。

每次搜索之前必须首先调用 findBegin 方法。

另请参见

- “findBegin 方法”一节第 154 页
- “isEOF 方法”一节第 103 页

findFirstForColumns 方法

在表中从开头往前移动，查找与当前索引中的一个值或部分值集完全匹配的行。

语法

Boolean **findFirstForColumns(**
UInt16 *numColumns*
)

参数

- **numColumns** 对于复合索引，是指在查找中使用的列数。例如，如果有一个三列的索引，而需要查找的值仅基于第一列进行匹配，则应为第一列设置值，然后将 **numColumns** 的值设置为 **1**。

返回值

如果成功，则返回 **true**；否则，返回 **false**

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在第一个与索引值完全匹配的行上。失败时，游标位置在最后一行之后 (isEOF)。

每次搜索之前必须首先调用 findBegin 方法。

另请参见

- “findBegin 方法”一节第 154 页
- “isEOF 方法”一节第 103 页

findLast 方法

在表中从末尾向后移动，查找与当前索引中的一个值或整个值集完全匹配的行。

语法

Boolean **findLast( )**

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在所找到的第一个与索引值完全匹配的行上。失败时，游标位置在第一行之前 (isBOF)。

每次搜索之前必须首先调用 findBegin 方法。

另请参见

- “findBegin 方法”一节第 154 页
- “isBOF 方法”一节第 103 页

findLastForColumns 方法

在表中从末尾向后移动，查找与当前索引中的一个值或部分值集完全匹配的行。

语法

Boolean **findLastForColumns(** UInt16 *numColumns* **)**

参数

- **numColumns** 对于复合索引，是指在查找中使用的列数。例如，如果有一个三列的索引，而需要查找的值仅基于第一列进行匹配，则应为第一列设置值，然后将 **numColumns** 的值设置为 **1**。

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在所找到的第一个与索引值完全匹配的行上。失败时，游标位置在第一行之前 (isBOF)。

每次搜索之前必须首先调用 findBegin 方法。

另请参见

- “findBegin 方法”一节第 154 页
- “isBOF 方法”一节第 103 页

findNext 方法

继续执行 findFirst 搜索，从表中的当前位置向前移动，查看下一行是否与当前索引中的某个值或整个值集完全匹配。

语法

Boolean **findNext( )**

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

如果下一行与索引值完全匹配，则游标将停留在该行上。失败时，游标位置在最后一行之后 (isEOF)。

如果在行更新过程中修改了要搜索的列值，则 findNext 方法的行为处于不确定状态。

findNextForColumns 方法

继续执行 findFirst 搜索，从表中的当前位置向前移动，查看下一行是否与当前索引中的某个值或部分值集完全匹配。

语法

Boolean **findNextForColumns(** UInt16 *numColumns*)

参数

- **numColumns** 对于复合索引，是指在查找中使用的列数。例如，如果有一个三列的索引，而需要查找的值仅基于第一列进行匹配，则应为第一列设置值，然后将 **numColumns** 的值设置为 **1**。

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

如果下一行与索引值完全匹配，则游标将停留在该行上。失败时，游标位置在最后一行之后 (isEOF)。

如果在行更新过程中修改了要搜索的列值，则 findNext 方法的行为处于不确定状态。

findPrevious 方法

继续执行 findLast 搜索，从表中的当前位置向后移动，查看上一行是否与当前索引中的某个值或整个值集完全匹配。

语法

Boolean **findPrevious( )**

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

如果上一行与索引值完全匹配，则游标将停留在该行上。失败时，游标位置在第一行之前 (isBOF)。

如果在行更新过程中修改了要搜索的列值，则 findPrevious 方法的行为处于不确定状态。

findPreviousForColumns 方法

继续执行 findLast 搜索，从表中的当前位置向后移动，查看上一行是否与当前索引中的某个值或部分值集完全匹配。

语法

Boolean **findPreviousForColumns(**
UInt16 *numColumns*
)

参数

- **numColumns** 对于复合索引，是指在查找中使用的列数。例如，如果有一个三列的索引，而需要查找的值仅基于第一列进行匹配，则应为第一列设置值，然后将 **numColumns** 的值设置为 **1**。

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

如果上一行与索引值完全匹配，则游标将停留在该行上。失败时，游标位置在第一行之前 (isBOF)。

如果在行更新过程中修改了要搜索的列值，则 findPrevious 方法的行为处于不确定状态。

getBoolean 方法

以布尔值形式返回指定列的值。

语法

Boolean **getBoolean(** UInt16 *index* **)**

参数

- **index** 列的 ID 号。结果集中第一列的 ID 为 1。

getBytes 方法

以字节数组形式返回指定列的值。

语法

Array **getBytes(** UInt16 *index* **)**

参数

- **index** 列的 ID 号。结果集中第一列的 ID 为 1。

注释

仅对 SQLType.BINARY 或 SQLType.LONGBINARY 类型的列有效。

getBytesSection 方法

从指定的偏移量开始，将指定的 SQLType.LONGBINARY 列的内容的子集复制到目标字节数组的指定偏移量处。

语法

```
UInt32 getBytesSection(
  UInt16 index
  UInt32 srcOffset,
  Array  dst,
  UInt32 dstOffset,
  UInt32 count
)
```

参数

index　包含二进制数据的列从 1 开始的序号。

srcOffset　列值的起始位置。该值从零开始。

dst　目标数组。

dstOffset　目标数组的起始位置。

count　要复制的字节数

返回值

读取的字节数。

注释

位于源列的 srcOffset（从 0 开始）到 srcOffset+count-1 处之间的字节会分别复制到目标数组的 dstOffset 到 dstOffset+count-1 位置。如果在复制完计数的字节之前遇到值尾，则目标数组的剩余部分保持不变。

如果以下任何一个条件为 true，则会抛出错误，Connection.sqlCode 设置为 SQLError.SQLE_INVALID_PARAMETER，并且不会修改目标：

- dst 参数为空值
- srcOffset 参数为负
- dstOffset 参数为负
- count 参数为负
- dstOffset + count 大于目标数组的长度 dst.length。

getDate 方法

以 Date 的形式返回值。

语法

```
Date getDate( UInt16 index )
```

参数

index 要获取的结果集中从 1 开始的序号。

getDouble 方法

以 Double 的形式返回值。

语法

Double **getDouble(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getFloat 方法

返回指定列的值。

语法

Float **getFloat(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getInt 方法

返回指定列的值。

语法

Int32 **getInt(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getLong 方法

返回指定列的值。

语法

Int64 **getLong(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getRowCount 方法

返回结果集中的行数。

语法

UInt32 **getRowCount( )**

getRowCountWithThreshold 方法

返回表中的行数，最多不超过行的指定阈值数。

语法

UInt32 **getRowCount(** UInt32 *threshold***)**

参数

threshold 此值为行计数操作指定一个限定值。如果行数大于阈值，则结果为阈值。当存在大量行时，计数行是一项高开销操作。一些应用程序只需要知道是否有多于指定数目的行存在（例如，确定是否为用户提供一个选项用于请求更多行），而不需要精确的行计数。如果此值为零，则计数所有行。

getShort 方法

以 Int16 的形式返回值。

语法

Int16 **getShort(** UInt16 *index* **)**

参数

index 要获取的结果集中从 1 开始的序号。

getString 方法

以 String 的形式返回值。

语法

String **getString(** UInt32 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

getStringChunk 方法

从指定的偏移量开始，将指定的 SQLType.LONGVARCHAR 列的值的子集复制到 String 对象。

语法

String **getStringChunk(**
  UInt16 *index*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

参数

- **index**　要获取的结果集中从 1 开始的序号
- **srcOffset**　字符串值的从 0 开始的起始位置。
- **count**　要复制的字符数。

返回值

复制了指定字符的字符串。

getTime 方法

以 Date 的形式返回值。

语法

Date **getTime(** UInt16 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

getTimestamp 方法

以 Date 的形式返回值。

语法

Date **getTimestamp(** UInt16 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

getULong 方法

以 64 位无符号整数的形式返回值。

语法

UInt64 **getULong(** UInt16 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

getUUID 方法

以 UUID 的形式返回列的值。

语法

UUID **getUUID(** UInt16 *index* **)**

参数

index　要获取的结果集中从 1 开始的序号。

注释

此列必须为 SQLType.BINARY 类型，且长度为 16。

insert 方法

插入含当前列值（使用 set 方法来指定）的新行。

语法

insert()

注释

在每次插入之前必须先调用 insertBegin。

insertBegin 方法

通过将所有当前列值设置为其缺省值来准备在此表中插入新行。

语法

insertBegin( )

注释

调用相应的 Set*Type* 方法，以指定要插入的非缺省值。

直到执行 insert 方法之后，才实际插入行和更改行中的数据；而且在提交之前该更改不是永久更改。

lookupBackward 方法

在表中从末尾向后移动，查找匹配或小于当前索引中某个值或整个值集的行。

语法

Boolean **lookupBackward( )**

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在所找到的第一个与索引值匹配或小于该索引值的行上。失败时（即没有任何行小于要查找的值），游标位置在第一行之前 (isBOF)。

每次搜索之前必须首先调用 lookupBegin 方法。

lookupBackwardForColumns 方法

在表中从开头向后移动，查找匹配或小于当前索引中某个值或部分值集的行。

语法

Boolean **lookupBackwardForColumns(** UInt16 *numColumns* **)**

参数

- **numColumns** 对于复合索引，是指在查找中使用的列数。例如，如果有一个三列的索引，而需要查找的值仅基于第一列进行匹配，则应为第一列设置值，然后将 **numColumns** 的值设置为 **1**。

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在所找到的第一个与索引值匹配或小于该索引值的行上。失败时（即没有任何行小于要查找的值），游标位置在第一行之前 (isBOF)。

每次搜索之前必须首先调用 lookupBegin 方法。

lookupBegin 方法

准备在此表中执行新的查找。

语法

lookupBegin( )

注释

要搜索的值是通过对用于打开此表的索引中的列调用相应的 set*Type* 方法来指定的。

lookupForward 方法

在表中从开头往前移动，查找匹配或大于当前索引中某个值或整个值集的行。

语法

Boolean **lookupForward( )**

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在第一个与索引值匹配或大于索引值的行上。失败时（即没有任何行大于要查找的值），游标位置在最后一行之后 (isEOF)。

每次搜索之前必须首先调用 lookupBegin 方法。

lookupForwardForColumns 方法

在表中从开头往前移动，查找匹配或大于当前索引中某个值或部分值集的行。

语法

Boolean **lookupForwardForColumns(** UInt16 *numColumns* **)**

参数

- **numColumns** 对于复合索引，是指在查找中使用的列数。例如，如果有一个三列的索引，而需要查找的值仅基于第一列进行匹配，则应为第一列设置值，然后将 **numColumns** 的值设置为 **1**。

返回值

如果成功，则返回 **true**；否则，返回 **false**。

注释

若要指定要搜索的值，请为索引中的每一列设置列值。游标停留在第一个与索引值匹配或大于索引值的行上。失败时（即没有任何行大于要查找的值），游标位置在最后一行之后 (isEOF)。

每次搜索之前必须首先调用 lookupBegin 方法。

isBOF 方法

如果成功，则返回 **true**；否则，返回 **false**。

语法

Boolean **isBOF( )**

isEOF 方法

如果成功，则返回 **true**；否则，返回 **false**。

语法

Boolean **isEOF( )**

isNull 方法

如果值为空，则返回 **true**，否则返回 **false**。

语法

Boolean **isNull(** UInt16 *index* **)**

参数

index　列索引值。

isOpen 方法

如果 ResultSet 已打开，则返回 **true**，否则返回 **false**。

语法

Boolean **isOpen( )**

moveAfterLast 方法

移至 ULResultSet 的最后一行之后的位置。

语法

Boolean **moveAfterLast( )**

返回值

如果成功，则返回 **true**。

如果失败，则返回 **false**。例如，如果没有行，此方法失败。

moveBeforeFirst 方法

移至第一行之前的位置。

语法

Boolean **moveBeforeFirst( )**

返回值

如果成功，则返回 **true**。

如果失败，则返回 **false**。例如，如果没有行，此方法失败。

moveFirst 方法

移至第一行。

语法

Boolean **moveFirst( )**

返回值

如果成功，则返回 **True**。

如果失败，则返回 **False**。例如，如果没有行，此方法失败。

moveLast 方法

移至最后一行。

语法

Boolean **moveLast( )**

返回值

如果成功，则返回 **True**。

如果失败，则返回 **False**。例如，如果没有行，此方法失败。

moveNext 方法

移至下一行。

语法

Boolean **moveNext( )**

返回值

如果成功，则返回 **True**。

如果失败，则返回 **False**。例如，如果没有行，此方法失败。

movePrevious 方法

移至上一行。

语法

Boolean **movePrevious( )**

返回值

如果成功，则返回 **true**。

如果失败，则返回 **false**。例如，如果没有行，此方法失败。

moveRelative 方法

相对于当前行移动一定数量的行。

语法

Boolean **moveRelative(** Int32 *index* **)**

参数

index　要移动的行数。该值可以为正数、负数或零。

返回值

如果成功，则返回 **true**。

如果失败，则返回 **false**。例如，如果没有行，此方法失败。

注释

相对于游标当前在结果集中的位置，正索引值在结果集中向前移动，负索引值在结果集中向后移动，零不移动游标。

open 方法

打开此表，以便使用其主键访问数据。

语法

open()

openWithIndex 方法

打开此表，以便使用指定的索引访问数据。

语法

openWithIndex(String *index*)

参数

- **index** 打开表所使用的索引的名称。如果为空值，则使用主键。

setBoolean 方法

使用 **boolean** 设置指定列的值。

语法

setBoolean(short *columnID*, boolean *value*)

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行 **insert** 或 **update** 之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 **false**：

```
t.setBoolean( 1, false );
```

setBytes 方法

使用 **byte** 数组设置指定列的值。

语法

setBytes(UInt16 *columnID*, Array *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

仅适用于 **SQLType.BINARY** 或 **SQLType.LONGBINARY** 类型的列。直到执行 **insert** 或 **update** 之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

示例

以下语句设置第一列的值：

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0'
blob[ 2 ] = 68;
t.setBytes( 1, blob );
```

setDate 方法

使用 **Date** 设置指定列的值。

语法

setDate(UInt16 *columnID*, Date *value***)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 2004/09/27：

```
t.setDate(
  1, new Date( 2004,9,27,0,0,0,0 )
);
```

setDouble 方法

使用 **double** 设置指定列的值。

语法

setDouble(UInt16 *columnID*, Double *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下示例设置第一列的值：

```
t.setDouble( 1, Number.MAX_VALUE );
```

setFloat 方法

使用 Float 型设置指定列的值。

语法

setFloat(UInt16 *columnID*, Float *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句设置第一列的值：

```
t.setFloat(
   1,
   (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

setInt 方法

使用 Integer 型设置指定列的值。

语法

setInt(UInt16 *columnID*, Int32 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 2147483647：

```
t.setInt( 1, 2147483647 );
```

setLong 方法

使用 Int64 设置指定列的值。

语法

setLong(UInt16 *columnID*, Int64 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 9223372036854770000：

```
t.setLong( 1, 9223372036854770000 );
```

setNull 方法

将一列设置为 SQL NULL。

语法

setNull(UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

直到执行插入或更新之后，才实际更改数据，而且在提交之前该更改不是永久更改。

setShort 方法

使用 UInt16 型设置指定列的值。

语法

setShort(UInt16 *columnID*, Int16 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 32767：

```
t.setShort( 1, 32767 );
```

setString 方法

使用 String 型设置指定列的值。

语法

setString(UInt16 *columnID*, String *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 **abc**。

```
t.setString( 1, "abc" );
```

setTime 方法

使用 Date 型设置指定列的值。

语法

setTime(UInt16 *columnID*, Date *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 18:02:13:0000：

```
t.setTime(
  1, new Date( 1966,4,1,18,2,13,0 )
);
```

setTimestamp 方法

使用 Date 型设置指定列的值。

语法

setTimestamp(UInt16 *columnID*, Date *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句将第一列的值设置为 1966/04/01 18:02:13:0000：

```
t.setTimestamp(
  1, new Date( 1966,4,1,18,2,13,0 )
);
```

setToDefault 方法

将指定列的值设置为其缺省值。

语法

setToDefault(UInt16 *columnID* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

setULong 方法

使用被视为无符号值的 64 位整数设置指定列的值。

语法

setULong(UInt16 *columnID*, UInt64 *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。

示例

以下语句设置第一列的值：

```
t.setULong( 1, 9223372036854770000 * 4096 );
```

setUUID 方法

使用 setUUID 设置指定列的值。

语法

setUUID(UInt16 *columnID*, UUID *value* **)**

参数

- **columnID** 列的 ID 号。表中第一列的 ID 值为 1。
- **value** 列的新值。

注释

直到执行插入或更新之后，才实际更改行中的数据；而且在提交之前该更改不是永久更改。仅对类型为 SQLType.BINARY 且长度为 16 的列有效。

示例

以下语句为表中的第一列设置新 UUID 值：

```
t.setUUID( 1, conn.getNewUUID(); );
```

另请参见

- “使用 UUID”一节 《MobiLink - 服务器管理》

truncate 方法

在临时激活 STOP SYNCHRONIZATION DELETE 语句时，删除表中的所有行。

语法

truncate()

update 方法

用当前列值（使用 set 方法指定）来更新当前行。

语法

update()

注释

在每次更新之前必须先调用 updateBegin。

updateBegin 方法

准备更新此表中的当前行。

语法

updateBegin()

注释

列值通过调用相应的 set*Type* 方法进行修改。

直到执行更新之后，才实际更改行中的数据，而且在提交之前该更改不是永久更改。

如果修改用于打开表的索引中的列，将会对所有活动搜索产生不可预知的影响。无法更新表主键中的列。

UUID 类

表示一个 UUID。UUID（通用唯一标识符，Universally Unique Identifier）或 GUID（全局唯一标识符，Globally Unique Identifier）是确保在所有计算机和数据库中的唯一性的生成值。UUID 作为 SQLType.BINARY(16) 值存储在 UltraLite 数据库中，可用于对行进行唯一标识。UUID 类可存储不可变的 UUID。

UUID 与创建它的连接相关联，并且在该连接关闭后，不能再转换为字符串。

equals 方法

如果此 UUID 与其它参数相同，则返回 **true**；否则返回 **false**。

语法

Boolean **equals(** UUID *other* **)**

参数

- **other** 用于比较的 UUID。

toString 方法

返回此 UUID 的字符串表示形式。

语法

String **toString()**

注释

此字符串的格式为 XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX，其中 X 是十六进制数字；如果与 UUID 相关联的连接关闭，则 X 为空值。

术语表

术语表 181

术语表

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。在版本 10.0.0 中，Adaptive Server Anywhere 更名为 SQL Anywhere 服务器，SQL Anywhere Studio 更名为 SQL Anywhere。

另请参见： “SQL Anywhere”一节第 198 页

包

Java 中相关类的集合。

被引用对象

一种对象（如表），该对象在另一个对象（如视图）的定义中被直接引用。

另请参见： “主键”一节第 208 页

编码

也称作字符编码，编码是一种方法，通过该方法可以将字符集中的每个字符映射到一个或多个字节的信息，这些信息通常以十六进制数字表示。编码的一个例子是 UTF-8。

另请参见：

- “字符集”一节第 208 页
- “代码页”一节第 183 页
- “归类”一节第 187 页

标识符

用于引用数据库对象（如表或列）的字符串。标识符可以包含 A 到 Z、a 到 z、0 到 9、下划线 (_)、at 符号 (@)、数字符号 (#) 或美元符号 ($) 中的任何字符。

并发

同时执行两个或更多个独立并且可能存在竞争关系的进程。SQL Anywhere 会自动使用锁定来隔离事务，并确保每个并发应用程序看到的数据集均一致。

另请参见：

- “事务”一节第 195 页
- “隔离级别”一节第 186 页

参考数据库

MobiLink 中一种用于 UltraLite 客户端开发的 SQL Anywhere 数据库。在开发过程中，可以将一个 SQL Anywhere 数据库同时作为参考数据库和统一数据库使用。通过其它产品建立的数据库无法用作参考数据库。

参照完整性

遵守数据一致性控制规则（具体而言，不同表中主键值与外键值之间的关系）。若要实现参照完整性，每个外键中的值必须与被引用表中行的主键值相符。

另请参见：

- “主键”一节第 208 页
- “外键”一节第 200 页

策略

QAnywhere 中指定应在何时进行消息传输的方式。

插件模块

Sybase Central 中一种用于访问和管理产品的方法。当您安装相应的产品时，插件通常会自动安装并注册 Sybase Central。通常，插件在 Sybase Central 主窗口中作为顶级容器出现，并且使用产品本身的名称，如 SQL Anywhere。

另请参见： “Sybase Central”一节第 199 页

查询

一条或一组 SQL 语句，用于访问和/或操作数据库中的数据。

另请参见： “SQL”一节第 198 页

冲突解决

在 MobiLink 中，冲突解决是指一种逻辑，它指定当两个用户修改不同远程数据库上同一行时的处理方法。

重定向器

一种 Web 服务器插件，用于为客户端与 MobiLink 服务器之间的请求和响应选择发送路径。此插件还实现了负荷平衡和故障转移机制。

抽取

SQL Remote 复制中从统一数据库卸载相应结构和数据的行为。此信息用于初始化远程数据库。

另请参见： “复制”一节第 185 页

触发器

一种特殊形式的存储过程，用户运行修改数据的查询时会自动执行该存储过程。

另请参见：

- “行级触发器”一节第 187 页
- “语句级触发器”一节第 205 页
- “完整性”一节第 201 页

传输规则

QAnywhere 中用于确定何时进行消息传输、传输哪些消息以及应在何时删除消息的逻辑。

窗口

作为分析功能执行对象的行组。一个窗口可以包含一行、多行或所有行的数据，这些数据已根据窗口定义中提供的分组规格进行了分区。窗口会进行移动，以包括为输入中的当前行执行计算所需的行数或行范围。窗口结构的主要优点是，不需要执行附加查询就可以有机会对结果进行分组和分析。

创建者 ID

UltraLite Palm OS 应用程序中一种在创建应用程序时指派的 ID。

存储过程

存储过程是数据库中存储的一组 SQL 指令，用于在数据库服务器上执行一组操作或查询。

代理表

一种本地表，它所包含的元数据可以像访问本地表一样访问远程数据库服务器上的表。

另请参见： “元数据”一节第 206 页

代理 ID

另请参见： “客户端消息存储库 ID”一节第 191 页

代码页

代码页是一种将字符集的字符映射到数字表示的编码，数字表示通常是 0 到 255 之间的一个整数。例如，Windows 代码页 1252 就是一个代码页。就本文档而言，代码页和编码这两个术语可以互换。

另请参见：

- “字符集”一节第 208 页
- “编码”一节第 181 页
- “归类”一节第 187 页

DBA 权限

使用户能够在数据库中执行管理活动的权限级别。DBA 用户在缺省情况下具有 DBA 权限。

另请参见： “数据库管理员 (DBA)”一节第 197 页

dbspace

用于创建更多数据存储空间的附加数据库文件。一个数据库可以包含在最多 13 个独立的文件（一个初始文件和 12 个 dbspace）中。每个表及其索引必须包含在单个数据库文件中。SQL 命令 CREATE DBSPACE 可将新文件添加到数据库中。

另请参见： “数据库文件”一节第 198 页

动态 SQL

执行前由程序以编程方式生成的 SQL。UltraLite 动态 SQL 是一种专用于小型设备的 SQL 变体。

对象树

Sybase Central 中数据库对象的层次。对象树的顶层显示您的 Sybase Central 版本所支持的全部产品。每种产品展开后会显示其自己的对象子树。

另请参见： “Sybase Central”一节第 199 页

EBF

快速错误修正软件。快速错误修正软件是含有一个或多个错误修正软件的软件子集。错误修正软件列在更新程序的发行说明中。错误修正软件更新可能只适用于具有相同版本号的已安装软件。已对该软件执行了一些测试，但该软件尚未进行完全测试。除非您自己已验证了软件的适用性，否则不要随应用程序分发这些文件。

发布

MobiLink 或 SQL Remote 中一种用于标识将要同步的数据的数据库对象。在 MobiLink 中，发布仅存在于客户端。一个发布包括多个项目。SQL Remote 用户可以通过预订发布来接收发布。MobiLink 用户可以通过创建发布的同步预订来同步发布。

另请参见：

- “复制”一节第 185 页
- “项目”一节第 203 页
- “发布更新”一节第 184 页

发布更新

SQL Remote 复制中对一个数据库中的一个或多个发布所做更改的列表。发布更新将作为复制消息的一部分定期发送到远程数据库。

另请参见：

- “复制”一节第 185 页
- “发布”一节第 184 页

发布者

SQL Remote 复制中数据库内可以与其它复制数据库交换复制消息的单个用户。

另请参见： “复制”一节第 185 页。

FILE

SQL Remote 复制中一种使用共享文件来交换复制消息的消息系统。它对测试以及在没有显式消息传送系统的情况下进行的安装很有用。

另请参见“复制”一节第 185 页。

分析树

查询的代数表示。

服务

在 Windows 操作系统上，服务是在运行应用程序的用户 ID 未登录时的应用程序运行方式。

服务器管理请求

一种 QAnywhere 消息，其格式设置为 XML 并发送到 QAnywhere 系统队列，作为一种管理服务器消息存储库或监控 QAnywhere 应用程序的方法。

服务器启动的同步

一种从 MobiLink 服务器启动 MobiLink 同步的方式。

服务器消息存储库

QAnywhere 中在消息传输到客户端消息存储库或 JMS 系统之前服务器上用于临时存储消息的关系数据库。消息通过服务器消息存储库在各客户端之间进行交换。

复制

在物理上不相同的数据库之间共享数据。Sybase 有三种复制技术：MobiLink、SQL Remote 和复制服务器。

复制代理

请参见： “LTM”一节第 192 页

复制服务器

Sybase 的一种基于连接的复制技术，用于与 SQL Anywhere 和 Adaptive Server Enterprise 一起使用。它专用于在一些数据库之间进行接近实时的复制。

另请参见： “LTM”一节第 192 页

复制频率

SQL Remote 复制中一项针对每个远程用户的设置，它决定发布者的消息代理向该远程用户发送复制消息的频率应为多少。

另请参见： “复制”一节第 185 页。

复制消息

SQL Remote 或复制服务器中一种在发布数据库与预订数据库之间发送的通信。消息包含复制系统所需的数据、直通语句及信息。

另请参见：

- “复制”一节第 185 页
- “发布更新”一节第 184 页

隔离级别

一个事务中的操作对其它并发事务中的操作的可见程度。隔离级别有四级，编号依次为 0 至 3。第 3 级提供最高级别的隔离。级别 0 为缺省设置。SQL Anywhere 还支持以下三个快照隔离级别：快照、语句快照和只读语句快照。

另请参见： “快照隔离”一节第 191 页

个人服务器

与客户端应用程序在同一台计算机上运行的数据库服务器。个人数据库服务器通常由单个用户在一台计算机上使用，但它可以支持来自该用户的几个并发连接。

工作表

一种内部存储区域，用于在查询优化过程中存储中间结果。

故障切换

在活动服务器、系统或网络出现故障或意外终止时切换到冗余或备用的服务器、系统或网络。故障转移会自动进行。

关系数据库管理系统 (RDBMS)

一种以相关表的形式存储数据的数据库管理系统。

另请参见： “数据库管理系统 (DBMS)”一节第 197 页

规范化

对数据库模式的改进，目的在于按照基于关系数据库理论的规则消除冗余并改善组织。

归类

定义数据库中文本属性的字符集与排序顺序的组合。对于 SQL Anywhere 数据库，缺省归类取决于运行服务器时所使用的操作系统和语言；例如，英语 Windows 系统上的缺省归类为 1252LATIN1。归类（也称作归类序列）用于对字符串进行比较和排序。

另请参见：

- “字符集”一节第 208 页
- “代码页”一节第 183 页
- “编码”一节第 181 页

行级触发器

每更改一行即执行一次的触发器。

另请参见：

- “触发器”一节第 183 页
- “语句级触发器”一节第 205 页

回退日志

对在每个未提交的事务执行过程中所做更改的记录。当收到 ROLLBACK 请求或者系统出现故障时，未提交的事务会从数据库中回退，将数据库返回其原先的状态。每个事务都有一个单独的回退日志，事务完成时日志会被删除。

另请参见： “事务”一节第 195 页

iAnywhere JDBC 驱动程序

iAnywhere JDBC 驱动程序提供了一个 JDBC 驱动程序，与纯 Java jConnect JDBC 驱动程序相比，该驱动程序拥有一些性能优势和功能优点，但它不是纯 Java 解决方案。建议在大多数情况下使用 iAnywhere JDBC 驱动程序。

另请参见：

- “JDBC”一节第 188 页
- “jConnect”一节第 188 页

InfoMaker

一种报告和数据维护工具，它用于创建复杂的表格、报告、图形、交叉表和表，并创建将这些报告用作构件块的应用程序。

Interactive SQL

一种 SQL Anywhere 应用程序，用于查询和更改数据库中的数据以及修改数据库的结构。Interactive SQL 不但提供了一个用于输入 SQL 语句的窗格，还提供了一些用于返回有关查询处理过程的信息和结果集的窗格。

JAR 文件

Java 档案文件。一种压缩的文件格式，由一个或多个用于 Java 应用程序的包的集合组成。它将安装和运行 Java 程序所需的全部资源都放在一个压缩文件中。

Java 类

Java 中的主要代码结构单元。它是组合在一起的过程和变量的集合，将过程和变量组合在一起的原因是它们都与某个特定的可识别类别有关。

jConnect

JavaSoft JDBC 标准的 Java 实现。它为 Java 开发人员提供多层和异类环境中的本地数据库访问。但在大多数情况下，iAnywhere JDBC 驱动程序是首选的 JDBC 驱动程序。

另请参见：

- “JDBC”一节第 188 页
- “iAnywhere JDBC 驱动程序”一节第 187 页

JDBC

Java 数据库连接。一种 SQL 语言编程接口，它允许 Java 应用程序访问关系数据。首选的 JDBC 驱动程序是 iAnywhere JDBC 驱动程序。

另请参见：

- “jConnect”一节第 188 页
- “iAnywhere JDBC 驱动程序”一节第 187 页

基表

永久性的数据表。有时为区别于临时表和视图，会将这种表称作**基表**。

另请参见：

- “临时表”一节第 191 页
- “视图”一节第 195 页

基于会话的同步

一种同步类型，这种同步会使数据表示在统一数据库和远程数据库都一致。MobiLink 基于会话。

基于脚本的上载

MobiLink 中一种将上载过程自定义为使用日志文件的替代方法的方式。

基于 SQL 的同步

MobiLink 中一种使用 MobiLink 事件将表数据与支持 MobiLink 的统一数据库进行同步的方式。对于基于 SQL 的同步，可以直接使用 SQL，也可以使用面向 Java 和 .NET 平台的 MobiLink 服务器 API 返回 SQL。

基于文件的下载

在 MobiLink 中同步数据的一种方式，其中下载以文件的方式进行分发，从而支持脱机分发同步更改。

集成登录

一种登录功能，它允许将同一个用户 ID 和口令用于操作系统登录、网络登录和数据库连接。

监听器

一个程序 (dblsn)，用于 MobiLink 服务器启动的同步。监听器安装在远程设备上，它们被配置为在接收到来自通告程序的信息时启动针对设备的操作。

另请参见： “服务器启动的同步”一节第 185 页

检查点

将对数据库的所有更改都保存到数据库文件中的时间点。在其它时间，所提交的更改仅保存到事务日志中。

检查约束

对列或列集强制实施指定条件的一种限制。

另请参见：

- “约束”一节第 207 页
- “外键约束”一节第 201 页
- “主键约束”一节第 208 页
- “唯一约束”一节第 202 页

脚本

MobiLink 中为处理 MobiLink 事件而编写的代码。脚本通过编程方式控制数据交换，以满足业务需要。

另请参见： “事件模型”一节第 195 页

脚本版本

MobiLink 中为创建同步而一起应用的一组同步脚本。

校验

测试数据库、表或索引是否受到特定类型的文件损坏。

校验和

随数据库页本身一起记录的计算出的数据库页位数。校验和能够确保数据库页写入磁盘时位数相符，因此数据库管理系统可以通过它来验证数据库页的完整性。如果计数相符，即认为数据库页已成功写入。

镜像日志

另请参见： “事务日志镜像”一节第 196 页

角色

概念性数据库建模中从一个角度描述某种关系的动词或短语。您可以用两个角色来描述每种关系。例如，"包含" 和 "隶属于" 便是角色。

角色名

外键的名称。由于它命名外表和主表之间的关系，因此称作角色名。缺省情况下，角色名就是表名，除非其它外键已经使用该名称（在这种情况下，缺省的角色名是表名后接一个三位的唯一数字）。也可以自己创建角色名。

另请参见： “外键”一节第 200 页

局部临时表

一种临时表，仅在复合语句执行期间或连接结束之前存在。当您只需要将数据集装载一次时，局部临时表非常有用。缺省情况下，行会在提交时被删除。

另请参见：

- “临时表”一节第 191 页
- “全局临时表”一节第 194 页

客户端/服务器

一种软件体系结构，在这种体系结构中，一个应用程序（客户端）从另一个应用程序（服务器）获取信息并向该应用程序发送信息。这两个应用程序常位于通过网络连接的不同计算机上。

客户端消息存储库

QAnywhere 中一种用于在远程设备上存储消息的 SQL Anywhere 数据库。

客户端消息存储库 ID

QAnywhere 中一种对客户端消息存储库进行唯一标识的 MobiLink 远程 ID。

快照隔离

一种为发出读请求的事务返回数据的已提交版本的隔离级别。SQL Anywhere 提供了以下三种快照隔离级别：快照、语句快照和只读语句快照。使用快照隔离时，读操作不会阻塞写操作。

另请参见： “隔离级别”一节第 186 页

连接

关系系统中的一种基本操作，它通过比较指定列中的值将两个或更多个表中的行链接在一起。

连接 ID

用于标识客户端应用程序与数据库之间给定连接的唯一编号。可以使用以下 SQL 语句来确定当前连接 ID：

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

连接类型

SQL Anywhere 提供了四种类型的连接：交叉连接、键连接、自然连接和使用 ON 子句的连接。

另请参见： “连接”一节第 191 页

连接配置

连接到数据库所需的一组参数，如用户名、口令和服务器名称，它们在存储后即可方便地使用。

连接启动的同步

一种 MobiLink 服务器启动的同步，在这种同步下，连接发生变化时会启动同步。

另请参见： “服务器启动的同步”一节第 185 页

连接条件

一种影响连接结果的限制。您可以通过紧跟在连接语句的后面插入 ON 子句或 WHERE 子句来指定连接条件。对于自然连接和关键连接，SQL Anywhere 会生成连接条件。

另请参见：

- “连接”一节第 191 页
- “生成的连接条件”一节第 196 页

临时表

为临时存储数据而创建的表。有两种类型：全局临时表和局部临时表。

另请参见：

- “局部临时表”一节第 190 页
- “全局临时表”一节第 194 页

LTM

日志传送管理器（Log Transfer Manager，简称 LTM）也称作复制代理。LTM 是一个与 Replication Server 一起使用的程序，它读取数据库事务日志并将提交的更改发送到 Sybase 复制服务器。

请参见： “复制服务器”一节第 186 页

轮询

在 MobiLink 服务器启动的同步中，轻量级轮询器（例如 MobiLink 监听器）从通告程序请求推式通知的方式。

另请参见： “服务器启动的同步”一节第 185 页

逻辑索引

指向物理索引的引用（指针）。磁盘上不存储逻辑索引的索引结构。

命令文件

包含 SQL 语句的文本文件。命令文件可以手工建立，也可以通过数据库实用程序自动建立。例如，dbunload 实用程序会创建一个命令文件，其中包含重新创建给定数据库所需的 SQL 语句。

MobiLink

一种基于会话的同步技术，其设计用途是将 UltraLite 和 SQL Anywhere 远程数据库与统一数据库同步。

另请参见：

- “统一数据库”一节第 199 页
- “同步”一节第 199 页
- “UltraLite”一节第 200 页

MobiLink 服务器

运行 MobiLink 同步的计算机程序，即 mlsrv11。

MobiLink 监控器

一种用于监控 MobiLink 同步的图形化工具。

MobiLink 客户端

有两种 MobiLink 客户端。对于 SQL Anywhere 远程数据库，MobiLink 客户端是 dbmlsync 命令行实用程序。对于 UltraLite 远程数据库，MobiLink 客户端内置于 UltraLite 运行时库中。

MobiLink 系统表

MobiLink 同步所需的系统表。它们由 MobiLink 安装程序脚本安装到 MobiLink 统一数据库中。

MobiLink 用户

MobiLink 用户用于与 MobiLink 服务器进行连接。在远程数据库上创建 MobiLink 用户，然后在统一数据库中注册该用户。MobiLink 用户名完全独立于数据库用户名。

模式

数据库的结构，其中包括表、列和索引以及它们之间的关系。

内连接

一种连接，在这种连接中，仅当两个表都满足连接条件时行才会出现在结果集中。内连接是缺省设置。

另请参见：

- “连接”一节第 191 页
- “外连接”一节第 201 页

ODBC

开放式数据库连接。一种用于与数据库管理系统连接的标准 Windows 接口。ODBC 是 SQL Anywhere 所支持的几种接口之一。

ODBC 管理器

一种随 Windows 操作系统提供的 Microsoft 程序，用于设置 ODBC 数据源。

ODBC 数据源

用户要通过 ODBC 访问的数据的规范以及获取该数据时所需的信息。

PDB

Palm 数据库文件。

PowerDesigner

一种数据库建模应用程序。PowerDesigner 为设计数据库或数据仓库提供了结构化的方法。SQL Anywhere 包括 PowerDesigner 的 Physical Data Model 组件。

PowerJ

一种 Sybase 产品，用于开发 Java 应用程序。

QAnywhere

应用程序到应用程序的消息传递（包括移动设备到移动设备和移动设备与企业之间的消息传递），它使在移动或无线设备上运行的自定义程序能够与处在中央位置的服务器应用程序进行通信。

QAnywhere 代理

QAnywhere 中一种运行在客户端设备上的进程，用于监控客户端消息存储库和确定应在何时传输消息。

嵌入式 SQL

一种 C 语言程序编程接口。SQL Anywhere 嵌入式 SQL 是 ANSI 和 IBM 标准的实现。

轻量级轮询器

在 MobiLink 服务器启动的同步中，轮询来自 MobiLink 服务器的推式通知的设备应用程序。

另请参见： “服务器启动的同步”一节第 185 页

全局临时表

一种临时表，在被显式地删除之前，其数据定义对所有用户都可见。全局临时表允许用户各自打开一个表的相同实例。缺省情况下，行在提交时被删除，并且始终是在连接结束时被删除。

另请参见：

- “临时表”一节第 191 页
- “局部临时表”一节第 190 页

日志文件

SQL Anywhere 所维护的事务日志。该日志文件用于确保在出现系统或介质故障时可以恢复数据库、提高数据库性能以及使用 SQL Remote 实现数据复制。

另请参见：

- “事务日志”一节第 195 页
- “事务日志镜像”一节第 196 页
- “完全备份”一节第 201 页

散列

散列是一种将索引条目转化为键的索引优化。索引散列旨在通过将足够的行实际数据与其行 ID 包括在一起，以避免进行先查找行、后装载行然后再将行解出才能得出索引值的高开销操作。

上载

同步过程的一个阶段，在此阶段数据从远程数据库传送到统一数据库。

设备跟踪

在 MobiLink 服务器启动的同步中，允许使用标识设备的 MobiLink 用户名来对消息进行寻址的功能。

另请参见： “服务器启动的同步”一节第 185 页

实例化视图

实例化视图是指已计算并已存储在磁盘上的视图。实例化视图同时具有视图的特征（使用查询说明进行定义）和表的特征（可以对其执行大多数表操作）。

另请参见：

- “基表”一节第 188 页
- “视图”一节第 195 页

世代号

MobiLink 中的一种机制，用于强制远程数据库先上载数据，然后再应用任何其它下载文件。

另请参见： “基于文件的下载”一节第 189 页

事件模型

MobiLink 中组成同步的事件（如 begin_synchronization 和 download_cursor）序列。如果为事件创建了脚本，则会调用事件。

视图

一种作为对象存储在数据库中的 SELECT 语句。它使用户能够看到一个或多个表中的行子集或列子集。每当用户使用特定表或表组合的视图时，都将利用存储在这些表中的信息重新计算视图。视图对确保安全以及定制数据库信息的外观来使数据访问简单明了有帮助。

事务

组成一个逻辑工作单元的 SQL 语句序列。事务要么全部得到处理，要么根本不做处理。SQL Anywhere 支持事务处理，并内置了锁定功能，使并发事务能够访问数据库而又不损坏数据。事务要么以 COMMIT 语句结束，该语句使对数据的更改成为永久性更改；要么以 ROLLBACK 语句结束，该语句撤消在事务执行过程中所做的全部更改。

事务日志

一种按进行更改的顺序存储对数据库所做全部更改的文件。它会提高性能并支持在数据库文件损坏时恢复数据。

事务日志镜像

同时维护的事务日志文件的完全相同副本（可选）。每当数据库更改写入事务日志文件时，也会同时写入事务日志镜像文件。

镜像文件应该与事务日志保留在不同的设备上，这样在任意设备出现故障时，日志的其它副本会确保数据可以安全地恢复。

另请参见： “事务日志”一节第 195 页

事务完整性

MobiLink 中对整个同步系统事务的有保证维护。要么同步整个事务，要么不对事务的任何部分进行同步。

生成的连接条件

一种自动生成的对连接结果的限制。有两种类型：关键和自然。指定 KEY JOIN 或指定关键字 JOIN 但不使用关键字 CROSS、NATURAL 或 ON 时，会生成关键连接。对于关键连接，所生成的连接条件取决于表之间的外键关系。指定 NATURAL JOIN 时会生成自然连接；所生成的连接条件基于两个表中的公用列名。

另请参见：

- “连接”一节第 191 页
- “连接条件”一节第 191 页

受保护的功能

数据库服务器启动时由 -sf 选项指定的功能，该数据库服务器上运行的任何数据库都无法使用该功能。

授权选项

一种权限级别，它允许用户向其他用户授予权限。

数据操作语言 (DML)

用于操作数据库中数据的 SQL 语句子集。DML 语句可以检索、插入、更新和删除数据库中的数据。

数据定义语言 (DDL)

用于定义数据库中数据结构的 SQL 语句子集。DDL 语句可以创建、修改和删除数据库对象（如表和用户）。

数据类型

数据的格式，如 CHAR 或 NUMERIC。在 ANSI SQL 标准中，数据类型也可以包括对大小、字符集和归类的限制。

另请参见： “域”一节第 205 页

数据立方体

一种多维结果集，每一维都以不同的方式对相同的结果进行分组和排序。数据立方体提供了有关数据的综合性信息，如果不使用数据立方体，要获得同样的信息就必须进行自连接查询和相关子查询。数据立方体是 OLAP 功能的一部分。

数据库

通过主键和外键关联的表的集合。表包含数据库中的信息。表和键一起定义数据库的结构。数据库管理系统会访问此信息。

另请参见：

- “外键”一节第 200 页
- “主键”一节第 208 页
- “数据库管理系统 (DBMS)”一节第 197 页
- “关系数据库管理系统 (RDBMS)”一节第 186 页

数据库对象

包含或接收信息的数据库组件。表、索引、视图、过程和触发器便是数据库对象。

数据库服务器

对所有针对数据库信息的访问进行管理的计算机程序。SQL Anywhere 提供了两种类型的服务器：网络服务器和个人服务器。

数据库管理系统 (DBMS)

用于创建和使用数据库的程序的集合。

另请参见： “关系数据库管理系统 (RDBMS)”一节第 186 页

数据库管理员 (DBA)

具有维护数据库所需权限的用户。DBA 通常负责对数据库模式的所有更改以及管理用户和组。数据库管理员角色自动内置于数据库中，其用户 ID 为 DBA，口令是 sql。

数据库连接

客户端应用程序与数据库之间的通信渠道。必须具有有效的用户 ID 和口令才能建立连接。为用户 ID 授予的特权决定了在连接过程中可以执行的操作。

数据库名称

服务器装载数据库时为数据库指定的名称。缺省数据库名是初始数据库文件的文件名（不含扩展名）。

另请参见： “数据库文件”一节第 198 页

数据库所有者 (dbo)

一种特殊的用户，他拥有不归 SYS 所有的系统对象。

另请参见：

- “数据库管理员 (DBA)”一节第 197 页
- “SYS”一节第 199 页

数据库文件

数据库保存在一个或多个数据库文件中。其中一个是初始文件，后面的文件称作 dbspace。每个表（包括其索引）都必须包含在单个数据库文件中。

另请参见： “dbspace”一节第 184 页

死锁

一组事务会进入的一种特殊状态，在该状态下这些事务都不能继续执行。

SQL

用于与关系数据库进行通信的语言。ANSI 定义了 SQL 的标准，其最新标准是 SQL-2003。SQL 的非官方全称是结构化查询语言。

SQL Anywhere

SQL Anywhere 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。SQL Anywhere 也是包含 SQL Anywhere RDBMS、UltraLite RDBMS、MobiLink 同步软件和其它组件的软件包的名称。

SQL Remote

一种基于消息的数据复制技术，用于在统一数据库与远程数据库之间进行双向复制。统一数据库和远程数据库必须是 SQL Anywhere。

SQL 语句

包含用于将指令传递给 DBMS 的 SQL 关键字的字符串。

另请参见：

- “模式”一节第 193 页
- “SQL”一节第 198 页
- “数据库管理系统 (DBMS)”一节第 197 页

锁定

一种在同时执行多个事务的过程中保护数据完整性的并发控制机制。SQL Anywhere 会自动应用锁以防止两个连接同时更改同一数据，并防止其它连接读取正接受更改的数据。

您可以通过设置隔离级别来控制锁定。

另请参见：

- “隔离级别”一节第 186 页
- “并发”一节第 181 页
- “完整性”一节第 201 页

索引

一组已排序的、与基表中的一个或多个列关联的键和指针。在表中一个或多个列上设置索引可以提高性能。

Sybase Central

一种数据库管理工具，通过图形用户界面提供 SQL Anywhere 数据库设置、属性和实用程序。Sybase Central 也可用于管理其它 Sybase 产品，其中包括 MobiLink。

SYS

一种拥有大多数系统对象的特殊用户。无法以 SYS 身份登录。

统一数据库

在分布式数据库环境中，是指用于存储数据主副本的数据库。出现冲突或差异时，将把统一数据库视为具有数据的主副本。

另请参见：

- “同步”一节第 199 页
- “复制”一节第 185 页

通信流

MobiLink 中 MobiLink 客户端与 MobiLink 服务器之间进行通信时所使用的网络协议。

通告程序

一种由 MobiLink 服务器启动的同步使用的程序。通告程序集成在 MobiLink 服务器中。它们会检查统一数据库是否有推式请求，并发送推式通知。

另请参见：

- “服务器启动的同步”一节第 185 页
- “监听器”一节第 189 页

同步

利用 MobiLink 技术在数据库之间复制数据的过程。

在 SQL Remote 中，同步专指以初始数据集初始化远程数据库的过程。

另请参见：

- “MobiLink”一节第 192 页
- “SQL Remote”一节第 198 页

推式请求

在 MobiLink 服务器启动的同步中，通告程序通过检查它来确定推式通知是否需要发送到设备的结果集中的一行值。

另请参见： “服务器启动的同步”一节第 185 页

推式通知

QAnywhere 中一种从服务器传送到 QAnywhere 客户端的特殊消息，用于提示客户端启动消息传输。在 MobiLink 服务器启动的同步中，从通告程序传送到包含推式请求数据和内部信息的设备的特殊消息。

另请参见：

- “QAnywhere”一节第 194 页
- “服务器启动的同步”一节第 185 页

UltraLite

一种针对小型设备、移动设备和嵌入式设备进行了优化的数据库。所面向的平台包括手机、传呼机和个人记事本。

UltraLite 运行时

一种过程中关系数据库管理系统，其中包括一个内置 MobiLink 同步客户端。每个 UltraLite 编程接口使用的库以及 UltraLite 引擎中都包括 UltraLite 运行时。

外表

包含外键的表。

另请参见： “外键”一节第 200 页

外部登录

与远程服务器通信时使用的替代登录名和口令。缺省情况下，SQL Anywhere 每次代表其客户端连接到远程服务器时都会使用这些客户端的名称和口令。但是，您可以通过创建外部登录来替换这一缺省设置。外部登录是指与远程服务器通信时使用的替代登录名和口令。

外键

一个表中复制另一个表中主键值的一个或多个列。外键建立表间的关系。

另请参见：

- “主键”一节第 208 页
- “外表”一节第 200 页

外键约束

对单个列或一组列的限制，指定表中的数据与某个其它表中数据的关系。对列集施加外键约束可使这些列成为外键。

另请参见：

- “约束”一节第 207 页
- “检查约束”一节第 189 页
- “主键约束”一节第 208 页
- “唯一约束”一节第 202 页

外连接

一种保留表中所有行的连接。SQL Anywhere 支持左、右和完全外连接。左外连接保留表中位于连接运算符左侧的行，当右表中的行不满足连接条件时，它将返回空值。完全外连接保留两个表中的所有行。

另请参见：

- “连接”一节第 191 页
- “内连接”一节第 193 页

完全备份

对整个数据库和事务日志（可选）的备份。完全备份包含数据库中的所有信息，因此可以在系统或介质出现故障时提供保护。

另请参见： “增量备份”一节第 207 页

完整性

遵守完整性规则的情况，完整性规则确保数据正确并准确，而且数据库的关系结构保持不变。

另请参见： “参照完整性”一节第 182 页

网关

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关如何发送用于服务器启动同步的消息的信息。

另请参见： “服务器启动的同步”一节第 185 页

网络服务器

从共享公共网络的计算机接受连接的数据库服务器。

另请参见： “个人服务器”一节第 186 页

网络协议

通信类型，如 TCP/IP 或 HTTP。

维护版本

维护版本是一套完整的软件，它升级已安装的具有相同主版本号的较早版本的软件（版本号格式是 *major.minor.patch.build*）。升级程序的发行说明中列出了错误修正软件和其它更改。

唯一约束

对某个列或一组列的限制，它要求所有非空值都各不相同。一个表可以有多个唯一约束。

另请参见：

- “外键约束”一节第 201 页
- “主键约束”一节第 208 页
- “约束”一节第 207 页

谓语

一种条件表达式，可以选择性地将其与逻辑运算符 AND 和 OR 组合在一起，以组成 WHERE 或 HAVING 子句中的条件集。在 SQL 中，求值结果为 UNKNOWN 的谓语将解释为 FALSE。

位数组

位数组是一种用于有效率地存储位序列的数组数据结构。位数组与字符串类似，不同的是其各个部分由 0（零）和 1（一）而不是字符组成。位数组通常用于保存一串布尔值。

Windows

Microsoft Windows 操作系统系列，如 Windows Vista、Windows XP 和 Windows 200x。

Windows CE

请参见“Windows Mobile”一节第 202 页。

Windows Mobile

Microsoft 为移动设备制造的操作系统系列。

文件定义数据库

MobiLink 中一种用于创建下载文件的 SQL Anywhere 数据库。

另请参见： “基于文件的下载”一节第 189 页

物理索引

索引存储在磁盘上的实际索引结构。

系统表

一种表，由 SYS 或 dbo 拥有，用于保存元数据。系统表也称作数据字典表，由数据库服务器创建并维护。

系统对象

由 SYS 或 dbo 拥有的数据库对象。

系统视图

存在于每一个数据库中的一种视图，它以易于理解的格式表示系统表中包含的信息。

下载

同步过程的一个阶段，在此阶段数据从统一数据库传送到远程数据库。

相关名

查询的 FROM 子句中使用的表或视图的名称—要么是表或视图的原始名称，要么是在 FROM 子句中定义的替代名称。

项目

在 MobiLink 或 SQL Remote 中，项目是表示整个表或表中行和列子集的数据库对象。项目在发布中组合在一起。

另请参见：

- “复制”一节第 185 页
- “发布”一节第 184 页

消息存储库

QAnywhere 中客户端和服务器设备上存储消息的数据库。

另请参见：

- “客户端消息存储库”一节第 190 页
- “服务器消息存储库”一节第 185 页

消息类型

SQL Remote 复制中指定远程用户与统一数据库发布者通信方式的数据库对象。一个统一数据库可能定义了几种消息类型，这样一来，不同的远程用户就可以使用不同的消息系统与统一数据库进行通信。

另请参见：

- “复制”一节第 185 页
- “统一数据库”一节第 199 页

消息日志

可存储来自数据库服务器或 MobiLink 服务器等应用程序的消息的日志。此类信息还可以出现在消息窗口中或记录到文件中。消息日志包括信息性消息、错误、警告以及来自 MESSAGE 语句的消息。

消息系统

SQL Remote 复制中用于在统一数据库与远程数据库之间交换消息的协议。SQL Anywhere 包括对以下消息系统的支持：FILE、FTP 和 SMTP。

另请参见：

- “复制”一节第 185 页
- “FILE”一节第 185 页

卸载

卸载数据库时会将数据库的结构和/或数据导出到文本文件（如果是结构，则导出到 SQL 命令文件中；如果是数据，则导出到 ASCII 逗号分隔文件中）。使用卸载实用程序来卸载数据库。

此外，您也可以使用 UNLOAD 语句卸载数据的选定部分。

性能统计

反映数据库系统性能的值。例如，CURRREAD 统计表示数据库服务器已发出但尚未完成的文件读取次数。

业务规则

基于实际要求的准则。通常，业务规则通过检查约束、用户定义数据类型以及事务的正确使用来实现。

另请参见：

- “约束”一节第 207 页
- “用户定义数据类型”一节第 205 页

引用对象

一种对象（如视图），其定义直接引用数据库中的另一个对象（如表）。

另请参见： “外键”一节第 200 页

用户定义数据类型

请参见“域”一节第 205 页。

游标

指向结果集的已命名链接，用于通过编程接口访问和更新行。在 SQL Anywhere 中，游标支持在查询结果中进行向前和向后移动。游标由两部分组成：游标结果集（通常由 SELECT 语句定义）和游标位置。

另请参见：

- “游标结果集”一节第 205 页
- “游标位置”一节第 205 页

游标结果集

与游标关联的查询所得到的行集。

另请参见：

- “游标”一节第 205 页
- “游标位置”一节第 205 页

游标位置

指向游标结果集中一个行的指针。

另请参见：

- “游标”一节第 205 页
- “游标结果集”一节第 205 页

语句级触发器

在整个触发语句完成后执行的触发器。

另请参见：

- “触发器”一节第 183 页
- “行级触发器”一节第 187 页

域

内置数据类型的别名，其中包括适用的精度值和小数位值，还可以选择是否包括 DEFAULT 值和 CHECK 条件。SQL Anywhere 中预定义了一些域，如货币数据类型。也称作用户定义数据类型。

另请参见： “数据类型”一节第 196 页

预订

MobiLink 同步中发布与 MobiLink 用户之间的客户端数据库中的一个链接，它使发布所描述的数据能够得到同步。

SQL Remote 复制中发布与远程用户之间的一种链接，它使用户能够与统一数据库交换该发布上的更新。

另请参见：

- “发布”一节第 184 页
- “MobiLink 用户”一节第 193 页

元数据

数据的数据。元数据描述其它数据的性质和内容。

另请参见： “模式”一节第 193 页

原子事务

保证成功完成或保证根本不予完成的事务。如果错误使原子事务的一部分无法完成，则将回退事务以防止数据库处于不一致的状态。

REMOTE DBA 特权

在 SQL Remote 中，消息代理 (dbremote) 所需的权限级别。MobiLink 中 SQL Anywhere 同步客户端 (dbmlsync) 所需的权限级别。当消息代理或同步客户端作为具有该权限的用户建立连接时，它将具有完全的 DBA 访问权。如果不是通过消息代理或同步客户端进行连接，则该用户 ID 将不具有附加权限。

另请参见： “DBA 权限”一节第 184 页

远程 ID

SQL Anywhere 和 UltraLite 数据库中一种由 MobiLink 使用的唯一标识符。远程 ID 初始情况下设置为 NULL，在数据库第一次同步期间将设置为 GUID。

远程数据库

MobiLink 或 SQL Remote 中一种与统一数据库交换数据的数据库。远程数据库可以共享统一数据库中的全部或部分数据。

另请参见：

- “同步”一节第 199 页
- “统一数据库”一节第 199 页

约束

对特定数据库对象（如表或列）中所包含值的限制。例如，列可以具有唯一性约束，该约束要求该列中的所有值互不相同。表可以具有外键约束，该约束指定该表中的信息与某个其它表中数据的关系。

另请参见：

- “检查约束”一节第 189 页
- “外键约束”一节第 201 页
- “主键约束”一节第 208 页
- “唯一约束”一节第 202 页

运营公司

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关供服务器启动的同步使用的公共运营公司的信息。

另请参见： “服务器启动的同步”一节第 185 页

增量备份

仅包含事务日志的备份，通常在两次完全备份之间使用。

另请参见： “事务日志”一节第 195 页

争用

为获取资源而竞争的行为。例如，就数据库而言，如果有两个或更多个用户试图编辑数据库的同一行，就会为获得编辑该行的权利而发生争用。

正则表达式

正则表达式是字符、通配符和运算符的序列，用于定义某种模式以在字符串内进行搜索。

直方图

直方图是列统计信息最重要的组成部分，是一种表示数据分布的方式。SQL Anywhere 维护直方图以为优化程序提供有关列值分布情况的统计信息。

直接行处理

MobiLink 中一种用于将表数据同步到 MobiLink 支持的统一数据库以外的数据源的方法。使用直接行处理时，上载和下载都可以实现。

另请参见：

- “统一数据库”一节第 199 页
- “基于 SQL 的同步”一节第 189 页

主表

包含外键关系中的主键的表。

主键

其值唯一标识表中各行的一个列或多个列。

另请参见： “外键”一节第 200 页

主键约束

一种对主键列的唯一性约束。一个表只能有一个主键约束。

另请参见：

- “约束”一节第 207 页
- “检查约束”一节第 189 页
- “外键约束”一节第 201 页
- “唯一约束”一节第 202 页
- “完整性”一节第 201 页

子查询

嵌套在 SELECT、INSERT、UPDATE 或 DELETE 语句或者其它子查询中的 SELECT 语句。

有两种类型的子查询：相关子查询和嵌套子查询。

字符串

字符串是以单引号围起的字符序列。

字符集

字符集是一组符号，包括字母、数字、空格和其它符号。字符集的一个例子是 ISO-8859-1，又称作 Latin1。

另请参见：

- “代码页”一节第 183 页
- “编码”一节第 181 页
- “归类”一节第 187 页

索引

A

AuthStatusCode 类 [UL M-Business Anywhere API]
  toString 方法, 55
  属性, 55
  说明, 55
autoCommit 模式
  UltraLite for M-Business Anywhere, 24
AvantGo (*见* M-Business Anywhere)
AvantGo M-Business Server (*见* M-Business Anywhere)
AvGo
  UltraLite for M-Business Anywhere 创建者 ID, 9
安全性
  UltraLite for M-Business Anywhere, 11

B

BLOB
  UltraLite for M-Business Anywhere, 24
  UltraLite for M-Business Anywhere 中的 GetBytes 方法, 24
  UltraLite for M-Business Anywhere 中的 GetBytesSection 方法, 24
帮助
  技术支持, x
包
  术语定义, 181
被引用对象
  术语定义, 181
编码
  术语定义, 181
标识符
  术语定义, 181
表
  在 UltraLite for M-Business Anywhere 中访问模式信息, 26
并发
  术语定义, 181
部署
  UltraLite for M-Business Anywhere, 32
  UltraLite for M-Business Anywhere 到 Windows Mobile, 32
  UltraLite for M-Business Anywhere 到 Windows 桌面, 32
  UltraLite for M-Business Anywhere 应用程序到 Palm OS, 32

C

CHECK 约束
  术语定义, 189
重定向器
  UltraLite for M-Business Anywhere 同步, 30
  术语定义, 182
commit 方法
  UltraLite for M-Business Anywhere, 24
ConnectionParms 类 [UL M-Business Anywhere API]
  toString 方法, 71
  属性, 70
  说明, 70
Connection 类 [UL M-Business Anywhere API]
  cancelGetNotification 方法, 57
  changeEncryptionKey 方法, 57
  close 方法, 58
  commit 方法, 58
  countUploadRow 方法, 58
  createNotificationQueue 方法, 59
  declareEvent 方法, 59
  destroyNotificationQueue 方法, 60
  executeNextSQLPassthroughScript 方法, 60
  executeSQLPassthroughScripts 方法, 60
  getDatabaseID 方法, 61
  getGlobalAutoIncrementUsage 方法, 61
  getLastDownloadTime 方法, 61
  getLastIdentity 方法, 62
  getNewUUID 方法, 62
  getNotification 方法, 62
  getNotificationParameter 方法, 63
  getSQLPassthroughScriptCount 方法, 63
  getTable 方法, 64
  grantConnectTo 方法, 64
  isOpen 方法, 64
  prepareStatement 方法, 64
  registerForEvent 方法, 65
  resetLastDownloadTime 方法, 65
  revokeConnectFrom 方法, 66
  rollback 方法, 66
  rollbackPartialDownload 方法, 66
  saveSyncParms 方法, 66
  sendNotification 方法, 67
  setDatabaseID 方法, 67
  startSynchronizationDelete 方法, 67

stopSynchronizationDelete 方法, 68
synchronize 方法, 68
synchronizeWithParm 方法, 68
triggerEvent 方法, 68
validateDatabase 方法, 69
属性, 56
说明, 56
CreationParms 类 [UL M-Business Anywhere API]
属性, 72
说明, 72
参考数据库
术语定义, 182
参照完整性
术语定义, 182
策略
术语定义, 182
插件模块
术语定义, 182
插入
UltraLite for M-Business Anywhere 中的行, 22
插入模式
UltraLite for M-Business Anywhere, 22
查询
术语定义, 182
查寻模式
UltraLite for M-Business Anywhere, 22
查找模式
UltraLite for M-Business Anywhere, 22
查找详细信息并请求技术协助
技术支持, xi
冲突解决
术语定义, 182
抽取
术语定义, 182
触发器
术语定义, 183
传输规则
术语定义, 183
窗口 (OLAP)
术语定义, 183
创建者 ID
UltraLite for M-Business Anywhere, 9
术语定义, 183
存储过程
术语定义, 183
错误
在 UltraLite for M-Business Anywhere 中进行处理, 27
提供反馈, x
错误处理
UltraLite for M-Business Anywhere, 27

D

DatabaseManager 类 [UL M-Business Anywhere API]
createDatabase 方法, 75
dropDatabase 方法, 75
getDatabaseOptions 方法, 76
openConnection 方法, 76
reOpenConnection 方法, 76
validateDatabase 方法, 77
属性, 74
说明, 74
DatabaseSchema 类
UltraLite for M-Business Anywhere 开发, 26
DatabaseSchema 类 [UL M-Business Anywhere API]
getCollationName 方法, 78
getDatabaseProperty 方法, 78
getDateFormat 方法, 79
getDateOrder 方法, 79
getNearestCentury 方法, 79
getPrecision 方法, 79
getPublicationCount 方法, 79
getPublicationName 方法, 80
getPublicationSchema 方法, 80
getSignature 方法, 80
getTableAGDBSet 方法, 81
getTableCount 方法, 81
getTableName 方法, 81
getTimeFormat 方法, 82
getTimestampFormat 方法, 82
isCaseSensitive 方法, 82
isOpen 方法, 82
常量, 78
说明, 78
DBA 权限
术语定义, 184
DBMS
术语定义, 197
dbspaces
术语定义, 184
DCX
关于, vi
DDL

术语定义, 196
DML
术语定义, 196
DML 操作
UltraLite for M-Business Anywhere, 16
DocCommentXchange (DCX)
关于, vi
代理 ID
术语定义, 183
代理表
术语定义, 183
代码页
术语定义, 183
动态 SQL
术语定义, 184
对象层次
UltraLite for M-Business Anywhere, 3
对象树
术语定义, 184

E

EBF
术语定义, 184

F

FILE
术语定义, 185
FILE 消息类型
术语定义, 185
find 方法
UltraLite for M-Business Anywhere, 21
发布
在 UltraLite for M-Business Anywhere 中访问模式信息, 26
术语定义, 184
发布更新
术语定义, 184
发布者
术语定义, 185
反馈
报告错误, x
提供, x
文档, x
请求更新, x
防火墙
M-Business Anywhere 同步, 30
访问模式信息
UltraLite for M-Business Anywhere, 26
分析树
术语定义, 185
服务
术语定义, 185
服务器管理请求
术语定义, 185
服务器启动的同步
术语定义, 185
服务器消息存储库
术语定义, 185
复制
术语定义, 185
复制代理
术语定义, 185
复制服务器
术语定义, 186
复制频率
术语定义, 186
复制消息
术语定义, 186

G

GetBytesSection 方法
UltraLite for M-Business Anywhere, 24
GetBytes 方法
UltraLite for M-Business Anywhere, 24
grantConnectTo 方法
UltraLite for M-Business Anywhere, 28
隔离级别
术语定义, 186
个人服务器
术语定义, 186
更新
UltraLite for M-Business Anywhere 中的行, 22
更新模式
UltraLite for M-Business Anywhere, 22
工作表
术语定义, 186
功能
用于 M-Business Anywhere, 2
故障排除
UltraLite M-Business Anywhere SQL 错误代码, 117
UltraLite M-Business Anywhere 使用 setKeepPartialDownload, 132
UltraLite M-Business Anywhere 处理错误, 27

故障切换
术语定义, 186
关键连接
术语定义, 196
规范化
术语定义, 187
归类
术语定义, 187
滚动
UltraLite for M-Business Anywhere, 20

H

HotSync
UltraLite for M-Business Anywhere, 9
HotSync 同步
UltraLite for M-Business Anywhere 同步参数, 66
环境变量
命令 shell, ix
命令提示符, ix
回退
UltraLite for M-Business Anywhere, 24
回退日志
术语定义, 187
获取帮助
技术支持, x

I

iAnywhere JDBC 驱动程序
术语定义, 187
iAnywhere 开发人员社区
新闻组, xi
IndexSchema 类 [UL M-Business Anywhere API]
getColumnCount 方法, 83
getColumnName 方法, 83
getName 方法, 83
getReferencedIndexName 方法, 83
getReferencedTableName 方法, 84
isColumnDescending 方法, 84
isForeignKey 方法, 84
isForeignKeyCheckOnCommit 方法, 84
isForeignKeyNullable 方法, 84
isPrimaryKey 方法, 85
isUniqueIndex 方法, 85
isUniqueKey 方法, 85
说明, 83
InfoMaker
术语定义, 187
install-dir
文档用法, viii
Interactive SQL
术语定义, 188

J

JAR 文件
术语定义, 188
JavaScript
维护应用程序状态, 11
Java 开发
UltraLite for M-Business Anywhere API, 53
Java 类
术语定义, 188
jConnect
术语定义, 188
JDBC
术语定义, 188
校验
术语定义, 190
校验和
术语定义, 190
基表
术语定义, 188
基于 SQL 的同步
术语定义, 189
基于会话的同步
术语定义, 188
基于脚本的上载
术语定义, 189
基于文件的下载
术语定义, 189
集成登录
术语定义, 189
技术支持
新闻组, xi
加密
UltraLite for M-Business Anywhere 开发, 15
监听器
术语定义, 189
检查点
术语定义, 189
脚本
术语定义, 189
脚本版本
术语定义, 190
角色

术语定义, 190
角色名
术语定义, 190
教程
UltraLite for M-Business Anywhere, 35
镜像日志
术语定义, 190
局部临时表
术语定义, 190

K

开发平台
UltraLite for M-Business Anywhere, 2
开发人员社区
新闻组, xi
客户端/服务器
术语定义, 190
客户端消息存储库
术语定义, 190
客户端消息存储库 ID
术语定义, 191
口令
UltraLite for M-Business Anywhere 中的验证, 28
快照隔离
术语定义, 191

L

lookup 方法
UltraLite for M-Business Anywhere, 21
LTM
术语定义, 192
联机手册
PDF, vi
连接
术语定义, 191
连接 ID
术语定义, 191
连接类型
术语定义, 191
连接配置
术语定义, 191
连接启动的同步
术语定义, 191
连接条件
术语定义, 191
列
在 UltraLite for M-Business Anywhere 中访问模式信息, 26
列集合
UltraLite for M-Business Anywhere, 20
临时表
术语定义, 191
轮询
术语定义, 192
逻辑索引
术语定义, 192

M

M-Business Anywhere
(*参见* UltraLite for M-Business Anywhere)
MobiLink
术语定义, 192
MobiLink 服务器
术语定义, 192
MobiLink 监控器
术语定义, 192
MobiLink 客户端
术语定义, 193
MobiLink 系统表
术语定义, 193
MobiLink 用户
术语定义, 193
moveFirst 方法
UltraLite for M-Business Anywhere, 20
UltraLite for M-Business Anywhere 开发, 17
moveNext 方法
UltraLite for M-Business Anywhere, 20
UltraLite for M-Business Anywhere 开发, 17
名称
UltraLite for M-Business Anywhere 持久性, 12
命令 shell
大括号, ix
引号, ix
括号, ix
环境变量, ix
约定, ix
命令提示符
大括号, ix
引号, ix
括号, ix
环境变量, ix
约定, ix
命令文件

术语定义, 192
模糊处理
UltraLite for M-Business Anywhere, 15
模式
UltraLite for M-Business Anywhere, 22, 26
术语定义, 193

N

内连接
术语定义, 193

O

ODBC
术语定义, 193
ODBC 管理器
术语定义, 193
ODBC 数据源
术语定义, 193
open 方法
UltraLite for M-Business Anywhere 中的 ULTable 对象, 20

P

PDB
术语定义, 193
PDF
文档, vi
persistName
UltraLite for M-Business Anywhere 参数, 12
PowerDesigner
术语定义, 193
PowerJ
术语定义, 194
PreparedStatement 类
UltraLite for M-Business Anywhere 用法, 16
PreparedStatement 类 [UL M-Business Anywhere API]
appendBytesParameter 方法, 86
appendStringChunkParameter 方法, 86
close 方法, 87
executeQuery 方法, 87
executeStatement 方法, 88
getPlan 方法, 88
getResultSetSchema 方法, 88
hasResultSet 方法, 88
isOpen 方法, 89
setBooleanParameter 方法, 89
setBytesParameter 方法, 89
setDateParameter 方法, 90
setDoubleParameter 方法, 90
setFloatParameter 方法, 90
setIntParameter 方法, 91
setLongParameter 方法, 91
setNullParameter 方法, 92
setShortParameter 方法, 92
setStringParameter 方法, 92
setTimeParameter 方法, 93
setTimestampParameter 方法, 93
setULongParameter 方法, 94
setUUIDParameter 方法, 94
说明, 86
PublicationSchema 类
UltraLite for M-Business Anywhere 开发, 26
PublicationSchema 类 [UL M-Business Anywhere API]
getName 方法, 95
说明, 95
平台
在 UltraLite for M-Business Anywhere 中受支持, 2

Q

QAnywhere
术语定义, 194
QAnywhere 代理
术语定义, 194
嵌入式 SQL
术语定义, 194
全局临时表
术语定义, 194

R

RDBMS
术语定义, 186
REMOTE DBA 权限
术语定义, 206
ResultSetSchema 类 [UL M-Business Anywhere API]
getColumnCount 方法, 113
getColumnID 方法, 113
getColumnName 方法, 113
getColumnPrecision 方法, 114
getColumnPrecisionByColID 方法, 114
getColumnScale 方法, 114
getColumnScaleByColID 方法, 115
getColumnSize 方法, 115

getColumnSizeByColID 方法, 115
getColumnSQLType 方法, 116
getColumnSQLTypeByColID 方法, 116
isOpen 方法, 116
说明, 113
ResultSet 类 [UL M-Business Anywhere API]
appendBytes 方法, 96
appendStringChunk 方法, 97
close 方法, 97
deleteRow 方法, 97
getAGDBSet 方法, 98
getBoolean 方法, 98
getBytes 方法, 98
getBytesSection 方法, 98
getDate 方法, 99
getDouble 方法, 100
getFloat 方法, 100
getInt 方法, 100
getLong 方法, 100
getRowCount 方法, 101
getRowCountWithThreshold 方法, 101
getShort 方法, 101
getString 方法, 101
getStringChunk 方法, 102
getTime 方法, 102
getTimestamp 方法, 102
getULong 方法, 103
getUUID 方法, 103
isBOF 方法, 103
isEOF 方法, 103
isNull 方法, 104
isOpen 方法, 104
moveAfterLast 方法, 104
moveBeforeFirst 方法, 104
moveFirst 方法, 104
moveLast 方法, 105
moveNext 方法, 105
movePrevious 方法, 105
moveRelative 方法, 105
setBoolean 方法, 106
setBytes 方法, 106
setDate 方法, 107
setDateTime 方法, 107
setDouble 方法, 107
setFloat 方法, 108
setInt 方法, 108
setLong 方法, 108
setNull 方法, 109
setShort 方法, 109
setString 方法, 109
setTime 方法, 110
setTimestamp 方法, 110
setULong 方法, 110
setUUID 方法, 111
update 方法, 111
updateBegin 方法, 111
属性, 96
说明, 96
revokeConnectFrom 方法
UltraLite for M-Business Anywhere, 28
rollback 方法
UltraLite for M-Business Anywhere, 24
日志文件
术语定义, 194

S

samples-dir
文档用法, viii
SELECT 语句
UltraLite for M-Business Anywhere 开发, 17
SQL
术语定义, 198
SQL Anywhere
文档, vi
术语定义, 198
SQLError 类 [UL M-Business Anywhere API]
说明, 117
SQL Remote
术语定义, 198
SQLType 类 [UL M-Business Anywhere API]
toString 方法, 127
说明, 126
SQL 语句
术语定义, 198
Sybase Central
术语定义, 199
SynchResult 类 [UL M-Business Anywhere API]
说明, 138
SyncParms 类 [UL M-Business Anywhere API]
getAdditionalParms 方法, 128
getAuthenticationParms 方法, 128
getDownloadOnly 方法, 128
getKeepPartialDownload 方法, 129
getNewPassword 方法, 129

getPartialDownloadRetained 方法, 129
getPassword 方法, 129
getPingOnly 方法, 129
getResumePartialDownload 方法, 130
getSendColumnNames 方法, 130
getSendDownloadAck 方法, 130
getStream 方法, 130
getStreamParms 方法, 130
getUploadOnly 方法, 130
getUserName 方法, 131
getVersion 方法, 131
setAdditionalParms 方法, 131
setAuthenticationParms 方法, 131
setDownloadOnly 方法, 132
setKeepPartialDownload 方法, 132
setMBAServer 方法, 133
setMBAServerWithMoreParms 方法, 133
setNewPassword 方法, 134
setPassword 方法, 134
setPingOnly 方法, 135
setSendColumnNames 方法, 135
setSendDownloadAck 方法, 135
setStream 方法, 136
setStreamParms 方法, 136
setUploadOnly 方法, 136
setUserName 方法, 137
setVersion 方法, 137
常量, 128
说明, 128
SyncResult 类 [UL M-Business Anywhere API]
getAuthStatus 方法, 138
getIgnoredRows 方法, 138
getPartialDownloadRetained 方法, 138
getStreamErrorCode 方法, 138
getStreamErrorParameters 方法, 139
getStreamErrorSystem 方法, 139
getTimestamp 方法, 139
getUploadOK 方法, 139
SYS
术语定义, 199
散列
术语定义, 194
删除
UltraLite for M-Business Anywhere 中的行, 22
上载
术语定义, 195
设备跟踪
术语定义, 195
生成的连接条件
术语定义, 196
实例化视图
术语定义, 195
世代号
术语定义, 195
事件模型
术语定义, 195
事务
UltraLite for M-Business Anywhere, 24
术语定义, 195
事务处理
UltraLite for M-Business Anywhere, 24
事务日志
术语定义, 195
事务日志镜像
术语定义, 196
事务完整性
术语定义, 196
视图
术语定义, 195
授权选项
术语定义, 196
受保护的功能
术语定义, 196
术语表
SQL Anywhere 术语列表, 181
数据操作
UltraLite for M-Business Anywhere, 16
UltraLite for M-Business Anywhere 中的 SQL, 16
UltraLite for M-Business Anywhere 中的表 API, 20
数据操作语言
术语定义, 196
数据库
术语定义, 197
数据库对象
术语定义, 197
数据库服务器
术语定义, 197
数据库管理员
术语定义, 197
数据库连接
术语定义, 197
数据库名称
术语定义, 197

数据库模式
在 UltraLite for M-Business Anywhere 中访问, 26
数据库所有者
术语定义, 198
数据库文件
术语定义, 198
数据类型
JavaScript, 54
UltraLite for M-Business Anywhere API, 54
在 UltraLite for M-Business Anywhere 中访问, 21
在 UltraLite for M-Business Anywhere 中转换, 21
术语定义, 196
数据立方体
术语定义, 197
死锁
术语定义, 198
索引
术语定义, 199
锁定
术语定义, 198

T

TableSchema 类
UltraLite for M-Business Anywhere 开发, 26
TableSchema 类 [UL M-Business Anywhere API]
getColumnCount 方法, 141
getColumnDefaultValue 方法, 141
getColumnDefaultValueByColID 方法, 141
getColumnID 方法, 141
getColumnName 方法, 142
getColumnPartitionSize 方法, 142
getColumnPartitionSizeByColID 方法, 142
getColumnPrecision 方法, 143
getColumnPrecisionByColID 方法, 143
getColumnScale 方法, 143
getColumnScaleByColID 方法, 144
getColumnSize 方法, 144
getColumnSizeByColID 方法, 144
getColumnSQLType 方法, 145
getColumnSQLTypeByColID 方法, 145
getIndex 方法, 145
getIndexCount 方法, 145
getIndexName 方法, 146
getName 方法, 146
getOptimalIndex 方法, 146
getPrimaryKey 方法, 146
getUploadUnchangedRows 方法, 147
isColumnAutoIncrement 方法, 147
isColumnAutoIncrementByColID 方法, 147
isColumnCurrentDate 方法, 147
isColumnCurrentDateByColID 方法, 148
isColumnCurrentTime 方法, 148
isColumnCurrentTimeByColID 方法, 148
isColumnCurrentTimestamp 方法, 149
isColumnCurrentTimestampByColID 方法, 149
isColumnGlobalAutoIncrement 方法, 149
isColumnGlobalAutoincrementByColID 方法, 149
isColumnNewUUID 方法, 150
isColumnNewUUIDByColID 方法, 150
isColumnNullable 方法, 150
isColumnNullableByColID 方法, 150
isInPublication 方法, 151
isNeverSynchronized 方法, 151
说明, 141
提交
UltraLite for M-Business Anywhere, 24
体系结构
UltraLite for M-Business Anywhere, 3
通告程序
术语定义, 199
通信流
术语定义, 199
同步
UltraLite for M-Business Anywhere 一键, 29
UltraLite for M-Business Anywhere 代码摘要, 30
UltraLite for M-Business Anywhere 体系结构说明, 30
UltraLite for M-Business Anywhere 概述, 29
术语定义, 199
同步 UltraLite 应用程序
UltraLite for M-Business Anywhere 开发, 29
统一数据库
术语定义, 199
图标
此帮助文档中使用的, ix
推式请求
术语定义, 200
推式通知
术语定义, 200

U

ULTable 类
UltraLite for M-Business Anywhere 开发, 17
ULTable 类 [UL M-Business Anywhere API]

appendBytes 方法, 152
appendStringChunk 方法, 153
deleteAllRows 方法, 153
deleteRow 方法, 153
findBegin 方法, 154
findFirst 方法, 154
findFirstForColumns 方法, 154
findLast 方法, 155
findLastForColumns 方法, 156
findNext 方法, 156
findNextForColumns 方法, 157
findPrevious 方法, 157
findPreviousForColumns 方法, 157
getBoolean 方法, 158
getBytes 方法, 158
getBytesSection 方法, 159
getDate 方法, 159
getDouble 方法, 160
getFloat 方法, 160
getInt 方法, 160
getLong 方法, 160
getRowCount 方法, 161
getRowCountWithThreshold 方法, 161
getShort 方法, 161
getString 方法, 161
getStringChunk 方法, 162
getTime 方法, 162
getTimestamp 方法, 162
getULong 方法, 163
getUUID 方法, 163
insert 方法, 163
insertBegin 方法, 163
isBOF 方法, 166
isEOF 方法, 166
isNull 方法, 166
isOpen 方法, 166
lookupBackward 方法, 164
lookupBackwardForColumns 方法, 164
lookupBegin 方法, 165
lookupForward 方法, 165
lookupForwardForColumns 方法, 165
moveAfterLast 方法, 166
moveBeforeFirst 方法, 167
moveFirst 方法, 167
moveLast 方法, 167
moveNext 方法, 168
movePrevious 方法, 168
moveRelative 方法, 168
open 方法, 169
openWithIndex 方法, 169
setBoolean 方法, 169
setBytes 方法, 169
setDate 方法, 170
setDouble 方法, 170
setFloat 方法, 171
setInt 方法, 171
setLong 方法, 172
setNull 方法, 172
setShort 方法, 173
setString 方法, 173
setTime 方法, 173
setTimestamp 方法, 174
setToDefault 方法, 174
setULong 方法, 175
setUUID 方法, 175
truncate 方法, 176
update 方法, 176
updateBegin 方法, 176
属性, 152
说明, 152
UltraLite
术语定义, 200
UltraLite for M-Business Anywhere
API, 53
体系结构, 3
使用 SQL 进行的数据操作, 16
使用 Table API 进行的数据操作, 20
关于, 1
功能, 2
加密, 15
同步 UltraLite 应用程序, 29
对象层次, 3
将应用程序部署到 Palm OS, 32
将应用程序部署到 Windows Mobile, 32
将应用程序部署到 Windows 桌面, 32
快速入门, 6
支持的平台, 2
教程, 35
构建 CustDB 和 Simple 应用程序, 6
用户验证, 28
系统要求, 2
维护状态, 11
访问模式信息, 26
连接到 UltraLite 数据库, 10

部署应用程序, 32
错误处理, 27
项目体系结构, 37
UltraLite for M-Business Anywhere API
AuthStatusCode 类, 55
Connection 类, 56
ConnectionParms 类, 70
CreationParms 类, 72
DatabaseManager 类, 74
DatabaseSchema 类, 78
IndexSchema 类, 83
PreparedStatement 类, 86
PublicationSchema 类, 95
ResultSet 类, 96
ResultSetSchema 类, 113
SQLError 类, 117
SQLType 类, 126
SyncParms 类, 128
SyncResult 类, 138
TableSchema 类, 141
ULTable 类, 152
UUID 类, 177
说明, 53
UltraLite for M-Business Anywhere API 属性
DatabaseManager 类, 74
UltraLite M-Business Anywhere (*见* UltraLite for M-Business Anywhere)
UltraLite 数据库
在 UltraLite for M-Business Anywhere 中连接, 10
访问 M-Business Anywhere 的模式信息, 26
UltraLite 引擎
M-Business Anywhere 属性用于, 74
UltraLite 运行时
M-Business Anywhere 属性用于, 74
术语定义, 200
UUID 类 [UL M-Business Anywhere API]
equals 方法, 177
toString 方法, 177
说明, 177

V

Visual Basic
UltraLite for M-Business Anywhere 中支持的版本, 2

W

Windows
术语定义, 202
Windows Mobile
UltraLite for M-Business Anywhere 中的目标平台, 2
使用 M-Business Anywhere 构建 CustDB 和 Simple 应用程序, 6
术语定义, 202
外表
术语定义, 200
外部登录
术语定义, 200
外键
术语定义, 200
外键约束
术语定义, 201
外连接
术语定义, 201
完全备份
术语定义, 201
完整性
术语定义, 201
网关
术语定义, 201
网络服务器
术语定义, 201
网络协议
术语定义, 202
网络协议选项
UltraLite for M-Business Anywhere API, 136
唯一约束
术语定义, 202
维护版本
术语定义, 202
位数组
术语定义, 202
谓语
术语定义, 202
文档
SQL Anywhere, vi
约定, vii
文件定义数据库
术语定义, 202
物理索引
术语定义, 203

X

系统表

术语定义, 203
系统对象
术语定义, 203
系统视图
术语定义, 203
下载
术语定义, 203
相关名
术语定义, 203
项目
术语定义, 203
消息存储库
术语定义, 203
消息类型
术语定义, 203
消息日志
术语定义, 204
消息系统
术语定义, 204
卸载
术语定义, 204
新闻组
技术支持, xi
行
在 UltraLite for M-Business Anywhere 中访问值, 21
行级触发器
术语定义, 187
性能
UltraLite 关闭对象, 13
UltraLite 将公用代码置于 JavaScript 文件中, 13
性能统计
术语定义, 204

Y

业务规则
术语定义, 204
疑难解答
新闻组, xi
引用对象
术语定义, 204
永久名称
UltraLite for M-Business Anywhere, 12
用户定义数据类型
术语定义, 205
用户验证
UltraLite for M-Business Anywhere, 28
游标
术语定义, 205
游标结果集
术语定义, 205
游标位置
术语定义, 205
语句级触发器
术语定义, 205
域
术语定义, 205
预订
术语定义, 206
预准备语句
UltraLite for M-Business Anywhere, 16
元数据
术语定义, 206
原子事务
术语定义, 206
远程 ID
术语定义, 206
远程数据库
术语定义, 206
约定
命令 shell, ix
命令提示符, ix
文档, vii
文档中的文件名, viii
约束
术语定义, 207
运营公司
术语定义, 207

Z

增量备份
术语定义, 207
争用
术语定义, 207
正则表达式
术语定义, 207
支持
新闻组, xi
支持的平台
UltraLite for M-Business Anywhere, 2
直方图
术语定义, 207
直接行处理
术语定义, 207

值
在 UltraLite for M-Business Anywhere 中访问, 21
主表
术语定义, 208
主键
术语定义, 208
主键约束
术语定义, 208
主题
图标, ix
转换
UltraLite for M-Business Anywhere 中的数据类型, 21
子查询
术语定义, 208
自然连接
术语定义, 196
字符串
术语定义, 208
字符集
术语定义, 208
作用域
UltraLite for M-Business Anywhere 中的变量, 11