应用程序开发指南

版本 8 发行版 1

S152-0065-01

应用程序开发指南

版本 8 发行版 1

S152-0065-01

注意:

在使用本资料及其支持的产品之前，请阅读第 329 页的『声明』中的概要信息。

第一版

本版本适用于 DB2 Everyplace 的版本 8.1（产品号 5724-D04）以及所有后续发行版和修订版，直到在新版本中另有声明为止。

本版本替换 S152-0065-00

目录

第 1 部分 介绍

第 1 章 DB2 Everyplace 产品概述

本节提供 DB2 Everyplace 的介绍、构成 DB2 Everyplace 解决方案的组件的描述以及 DB2 Everyplace 典型方案的示例。本节包含下列主题:

什么是 DB2 Everyplace?

DB2 Everyplace 是 IBM 普及计算解决方案的一部分。有了 DB2 Everyplace, 移动性大的专业人员（如销售人员、检查人员、审计员、现场服务技术人员、医生、房地产经纪和保险索赔评定员）就能够在办公室以外的地方获取他们需要的重要数据。

组织机构现在能够把这些人员的 DB2 企业数据传递给移动式和嵌入式设备。借助 DB2 Everyplace, 您可以存取移动式设备上的数据库并对其执行更新。有了 DB2 Everyplace Sync Server, 您能够将移动式设备中的数据与企业中的其它数据源同步。“文件适配器”能力使您能够将文件和应用程序分布至移动用户。

DB2 Everyplace 数据库是驻留在移动式设备上的关系数据库。要存取移动式设备上的数据, 可使用快速应用程序开发工具、一组受支持的“DB2 调用层接口”（CLI）函数、“Java 数据库连接”（JDBC）方法或 ADO.NET 方法来编写应用程序。

DB2 Everyplace 解决方案的组件

DB2 Everyplace 解决方案具有下列主要功能部件和组件:

- DB2 Everyplace 移动式数据库。
- DB2 Everyplace Sync Server。
- DB2 Everyplace Sync Client。
- DB2 Everyplace Mobile Application Builder。
- DB2 Everyplace 样本应用程序。

DB2 Everyplace 移动式数据库

此数据库驻留在移动式设备上。移动式数据库“DB2 Everyplace 数据库版”、“DB2 Everyplace 企业版”和“DB2 Everyplace Software Development Kit”包含在一起。与移动式数据库相关联的另一组件为:

- 样本应用程序（引擎端）

DB2 Everyplace 移动式数据库可用于:

- Palm OS
- Symbian OS
- Windows CE／Pocket PC
- Win32（Windows® 95、Windows® 98、Windows® NT®、Windows® 2000® 和 Windows® XP®）
- QNX Neutrino、Linux 和嵌入式 Linux 设备。

DB2 Everyplace 也支持使用 MIDP 数据库的 MIDP 移动式设备。

DB2 Everyplace Sync Server

DB2 Everyplace Sync Server 与 “DB2 Everyplace 企业版” 包括在一起。与 Sync Server 相关联的其它重要的组件包括:

- DB2 Everyplace 移动式设备管理中心
- 样本应用程序（服务器端）

可以使用 DB2 Everyplace Sync Server 和 DB2 Everyplace Sync Client 来在 DB2 Everyplace 移动式设备与企业数据源之间使数据和应用程序同步。

数据同步可以是双向的，也可以是单向的。可以在 DB2 Everyplace 移动式设备或企业数据库上更新数据库。例如，用户可以从 DB2 z/OS 版数据库下载数据子集到移动式设备上的 DB2 Everyplace 数据库，查看数据、更改数据，然后将更改过的数据重新与 z/OS 服务器同步。DB2 Everyplace Sync Server 也提供了一种机制来解决冲突问题。

DB2 Everyplace Sync Server 提供了一个管理工具，该工具可以帮助您管理同步服务以及将它们传递至具有类似数据同步需求的用户组。有关 “移动式设备管理中心” 的更多信息，请参阅《Sync Server 管理指南》。

DB2 Everyplace Sync Server 支持使关系数据与具有 JDBC 接口的数据源（如 DB2 通用数据库）同步。

DB2 Everyplace Sync Server 支持使关系数据与下列数据源同步:

- DB2 通用数据库 z/OS 版
- DB2 通用数据库 iSeries 版
- DB2 通用数据库 Linux 版、UNIX 版和 Windows 版
- 带有 JDBC 接口的任何数据源

DB2 Everyplace Sync Client

DB2 Everyplace Sync Client 与 “DB2 Everyplace 企业版” 包括在一起。

DB2 Everyplace Sync Client 在移动式设备上运行，它由使用 DB2 Everyplace Sync Server 的应用程序组成。它处理企业关系数据与移动式设备上的 DB2 Everyplace 移动式数据库的双向同步。移动式设备还管理与文件预订相关的操作，以便可以容易地将移动式应用程序分布至设备以及能够执行存储在 DB2 数据库中的存储过程。

Sync Client 在下列操作系统上可用:

- Palm OS

- Symbian OS
- Windows CE／Pocket PC
- Win32（Windows® 95、Windows® 98、Windows® NT®、Windows® 2000® 和 Windows® XP®）
- QNX Neutrino、Linux 和嵌入式 Linux 设备

有关随 Sync Client 提供的“应用程序编程接口”（API）的信息，请参阅《*DB2 Everyplace 应用程序开发指南*》。

DB2 Everyplace Mobile Application Builder

DB2 Everyplace Mobile Application Builder 与 Software Development Kit 包括在一起，也可从 IBM Web 站点下载它。

可以使用 DB2 Everyplace Mobile Application Builder 来为 Palm OS、WinCE、Symbian OS 和支持用户界面和“Java 虚拟机”的其它平台开发 DB2 Everyplace 应用程序。有了 Mobile Application Builder，不必写一行代码就可构建应用程序。有关如何获取 Mobile Application Builder 的信息，访问 DB2 Everyplace Web 站点。

其它开发工具包括 WebSphere Studio Device Developer、Visual Age Micro Edition、Metrowerks CodeWarrior 和 GNU Software Developer’s Kit。

DB2 Everyplace 样本应用程序

样本应用程序提供使用 DB2 Everyplace 的应用程序的示例。可以使用 Visiting Nurse 样本应用程序来快速测试移动式数据库和 Sync Server 之间的双向同步。样本应用程序具有两部分，一部分在 Sync Server 上运行，另一部分在移动式数据库上运行。此移动式数据库样本应用程序演示独立环境中的数据库引擎功能。当 Sync Server 样本应用程序和数据库引擎样本应用程序一起使用时，它们作为调用 DB2 Everyplace 的所有组件的完整应用程序运行。

IBM Sync 也是一个样本应用程序，它演示如何使用 DB2 Everyplace Sync Client API 来使在 MDAC 中定义的预订的表同步。

“命令行处理器”是一种应用程序开发工具，它是在具有命令行界面的平台上作为使用 DB2 Everyplace 的示例应用程序提供的。“命令行处理器”用于移动式设备上的 DB2 Everyplace 数据库。Sync Server 不使用它。

DB2 Everyplace 支持的 SQL 语句使您能够创建和删除表和索引以及删除、插入和更新表行。

有关受支持的 SQL 语句的更多信息，请参阅《*DB2 Everyplace 应用程序开发指南*》。

示例 DB2 Everyplace 方案

保险索赔评定员负责检查申请赔偿的客户的财产损失情况。在许多公司中，评定员了解索赔对象的财产、填写表单以确认或拒绝索赔，并评定要付给索赔对象的损失金额。之后，当评定员回到办公室时，将表单以手工方式输入到公司的计算机系统中，此过程单调乏味并且是高成本的。

让评定员配备运行 DB2 Everyplace 应用程序的移动式设备可以大大提高此过程的效率。因为在任何地方都可以使用移动式设备，评定员可以访问他们的检查安排、路径和索赔对象策略信息。评定员还可以在移动式设备上完成评定表单。当评定员回到办公室时，他们可以通过将新的评定表单数据上载至公司的企业数据库来使他们的移动式设备上的数据与公司的计算机系统同步。如果评定员在现场需要信息，他们可以通过调制解调器使移动式设备上的数据立即与公司的计算机系统同步。索赔评定过程现在可以完全做到无纸化，这可以为保险公司节省大笔费用。因为评定员可以即时访问他们公司的企业数据库，所以索赔可以更快得到解决。

第 2 部分 开发 DB2 Everyplace 应用程序

第 2 章 开发 DB2 Everyplace C/C++ 应用程序

本章包含有关开发用于 DB2 Everyplace 的 C/C++ 应用程序的信息。本章包含下列各节:

开发 DB2 Everyplace C/C++ 应用程序

要使用 C/C++ 开发 DB2 Everyplace 应用程序，应使用 DB2 Everyplace CLI/ODBC 接口。本主题提供要使用 DB2 Everyplace 开发 C/C++ 应用程序必须完成的任务的概述。

要开发使用 C/C++ 的 DB2 Everyplace 应用程序: :

1. 在开发工作站上安装 DB2 Everyplace。有关详细的指示信息，请参阅《*DB2 Everyplace 安装与用户指南*》。
2. 定义应用程序及其数据需求。

   确定最终用户需要查看或更改的数据以及如何在 DB2 Everyplace 数据库中检索、存储和更新数据。
3. 了解 DB2 CLI 接口并确定在应用程序中要使用哪些 DB2 CLI 函数。
4. 使用 DB2 Everyplace 支持的 DB2 CLI 函数编写 C/C++ 应用程序。
5. 准备和编译应用程序代码，并将其与 DB2 Everyplace 头文件和操作系统库相链接。
6. 测试应用程序:
   a. 将 DB2 Everyplace 库复制到操作系统的仿真器或设备。
   b. 在设备或仿真器上测试应用程序（如果适用的话）。

相关概念:

- 第 85 页的第 13 章，『样本 C/C++ 应用程序』

相关参考:

- 『支持的 C/C++ 开发工具』
- 第 11 页的『支持 C/C++ 的操作系统』
- 第 175 页的『DB2 CLI 函数摘要』

支持的 C/C++ 开发工具

可使用 DB2 Everyplace 支持的 DB2 CLI 函数编写 C/C++ 应用程序。

用于受支持的操作系统的受支持的标准 C/C++ 开发工具包括:

Palm OS

您可以使用:

- DB2 Everyplace Mobile Application Builder. 有关 Mobile Application Builder 的信息，请访问 DB2 Everyplace Web 站点，网址为: http://www.ibm.com/software/data/db2/everyplace/
- GNU Software Developer's Kit。
- Metrowerks CodeWarrior for Palm Computing Platform。此商业开发环境允许您使用 Windows 工作站来为 Palm OS 操作系统创建 C/C++ 程序。

建议: 向 Palm 公司注册应用程序创建者标识，以避免与其它 Palm OS 应用程序发生冲突。DB2 Everyplace 表和应用程序的创建者标识类似于 IBDB 或 DB2*x*，其中，*x* 是 a 到 z 的字母。有关创建者标识的更多信息，请访问以下 Web 站点:

http://www.palmos.com/dev/

Symbian OS V6.0

可使用 Microsoft Visual C++ V6 和 Symbian V6.0 C++ Software Developer's Kit（SDK）来开发应用程序。

建议: 从 Symbian OS 中获取 UID 以加入到项目文件中。可从 SDK 或以下 Web 站点获取这些标识:

http://www.symbian.com/developer/techlib/papers/tn_uid/uidinfo.html

Symbian OS V7.0

可使用 Metrowerks CodeWarrior for Symbian 和 Symbian OS V7.0 SDK 来开发应用程序。

Windows CE

使用 Microsoft eMbedded Visual Tools 3.0 来开发应用程序。

Windows NT 和 Windows 2000 操作系统

可以使用 Microsoft Visual C++ 来开发应用程序。

QNX Neutrino

可使用 Metrowerks CodeWarrior for QNX Neutrino 或 QNX Neutrino Software Developer's Kit（SDK）来开发应用程序。

Linux 可使用嵌入式 Linux 分布式系统的跨平台开发工具来开发应用程序。嵌入式 Linux 内核需要启用对 ELF 二进制的支持。

相关概念:

• 第 85 页的第 13 章，『样本 C/C++ 应用程序』

相关任务:

• 第 9 页的『开发 DB2 Everyplace C/C++ 应用程序』

支持 C/C++ 的操作系统

下列操作系统完全支持 C/C++ 接口:

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Win32（Windows 95、Windows 98、Windows NT、Windows 2000 和 Windows XP）
- QNX Neutrino
- Linux 和嵌入式 Linux

相关概念:

- 第 85 页的第 13 章, 『样本 C/C++ 应用程序』

相关任务:

- 第 9 页的『开发 DB2 Everyplace C/C++ 应用程序』

相关参考:

- 第 9 页的『支持的 C/C++ 开发工具』
- 第 175 页的『DB2 CLI 函数摘要』

准备、编译和链接 C/C++ 项目

此任务是“使用 C/C++ 开发 DB2 Everyplace 应用程序”这一大型任务的一部分。当完成“准备、编译和链接 C/C++ 项目”的步骤时, 请返回至第 9 页的『开发 DB2 Everyplace C/C++ 应用程序』。

过程:

DB2 Everyplace 包含用于应用程序开发的头文件和操作系统库文件。

要使用正确的编译器来准备项目文件并编译和链接 DB2 Everyplace 应用程序:

1. 创建一个项目文件。此过程根据开发工具和开发操作系统而有所变化。
2. 将下列 DB2 Everyplace 头文件加入该项目。头文件包含随 DB2 Everyplace 一起提供的常量、数据类型和 C/C++ 函数原型。头文件包括:

   ```
   \db2everyplace\Clients\include\sqlcli.h
   \db2everyplace\Clients\include\sqlcli1.h
   \db2everyplace\Clients\include\sqlext.h
   \db2everyplace\Clients\include\sqlsystm.h
   ```

3. 包括特定于应用程序的任何头文件。
4. 将适当的 DB2 Everyplace 库加入该项目。

下表概述了 DB2 Everyplace 库并列出每个操作系统的附加信息。

表 1. DB2 Everyplace 库

操作系统	所需的库文件和附加信息
Palm OS	\db2everyplace\clients\palmos\database\DB2e.lib 可选: 将堆栈大小增大为 8 KB。缺省值为 4 KB。 Palm OS 应用程序具有有限的缺省应用程序堆栈大小。视应用程序的不同，您可能会在运行时遇到堆栈溢出问题。为避免此问题，在 DB2 Everyplace 附带的 palm-pref.r 文件中指定较大的堆栈大小。遵循 palm-pref.r 文件中的指示信息并将其包括在项目文件中。 如果在使用 PRC-Tools 开发应用程序，则将 stack=0x8000 添加到应用程序的 .def 文件中。例如: application {"MyApplicationName" APID stack=0x8000 }
Symbian OS V6	仿真器应用程序: \db2everyplace\clients\symbian6\database\wins\DB2e.lib 设备应用程序: \db2everyplace\clients\symbian6\database\armi\DB2e.lib
Symbian OS V7	仿真器应用程序: \db2everyplace\clients\Symbian7\database\wins\DB2e.lib 设备应用程序: \db2everyplace\clients\Symbian7\database\armi\DB2e.lib
Windows CE	ARM 处理器: • V3.00 \db2everyplace\clients\wince\database\wce300\armrel\DB2e.lib • V4.00 \db2everyplace\clients\wince\database\wce400\ARM4VRel\DB2e.lib MIPS 处理器: • V3.00 \db2everyplace\clients\wince\database\wce300\mipsrel\DB2e.lib • V4.00 \db2everyplace\clients\wince\database\wce400\MIPSIVRel\DB2e.lib SH3 处理器: • V3.00 \db2everyplace\clients\wince\database\wce300\sh3rel\DB2e.lib • V4.00 \db2everyplace\clients\wince\database\wce400\SH3Rel\DB2e.lib Windows CE 仿真器: • V3.00 \db2everyplace\clients\wince\database\wce300\x86emrel\DB2e.lib（用于 Pocket PC 仿真器） \db2everyplace\clients\wince\database\wce300\x86rel\DB2e.lib（用于 Pocket PC 2002 仿真器） • V4.00 \db2everyplace\clients\wince\database\wce400\emulatorRel\DB2e.lib（用于 WinCE.NET 仿真器） 验证 UNICODE 是否支持该项目。将 UNICODE 和 _UNICODE 添加至“项目设置”的**预处理器定义**。 XScale 处理器: • V3.00 \db2everyplace\clients\wince\database\wce300\xscale\DB2e.lib
Win32	\db2everyplace\clients\Win32\database\x86\DB2e.lib
Neutrino	libdb2e.so Neutrino 库文件在 DB2 Everyplace Neutrino gzipped tar 文件 DB2EveryplaceNTO.tar.gz 中。此文件位于 \db2everyplace\clients\neutrino\database\ 目录中。libdb2e.so 位于 /db2e/database/x86/（对于 x86 处理器类型）目录中。
Linux	libdb2e.so Linux 库文件在 DB2 Everyplace Linux gzipped tar 文件 DB2EveryplaceLN.tar.gz 中。此文件位于 \db2everyplace\clients\embeddedlinux\database\ 目录中。libdb2e.so 位于 /db2e/database/x86/（对于 x86 处理器类型）和 /db2e/database/strongarm/（对于 strongarm 处理器类型）目录中。

5. 可选: 在项目文件中定义宏 UNICODE 和 _UNICODE 以获取 UNICODE 支持。有关 UNICODE 的更多信息，请参阅第 322 页的『DB2 Everyplace UNICODE 支持』
6. 编译项目并将对象代码与适当的 DB2 Everyplace 库相链接。

许多应用程序开发工具都提供了在集成开发环境中自动编译和链接的功能。有关编译和链接项目的附加信息，请参阅应用程序开发软件附带的文档。

相关概念:

- 第 85 页的第 13 章，『样本 C/C++ 应用程序』

相关任务:

- 『测试 C/C++ 应用程序』

相关参考:

- 第 9 页的『支持的 C/C++ 开发工具』
- 第 11 页的『支持 C/C++ 的操作系统』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 322 页的『DB2 Everyplace UNICODE 支持』

测试 C/C++ 应用程序

此任务是“使用 C/C++ 开发 DB2 Everyplace 应用程序”这一大型任务的一部分。当完成“测试 C/C++ 应用程序”的步骤时，请返回至第 9 页的『开发 DB2 Everyplace C/C++ 应用程序』。

过程:

要测试应用程序:

1. 将 DB2 Everyplace 库复制到操作系统的仿真器或设备。如果没有这些文件，DB2 Everyplace 应用程序将无法装入。下表概述每个操作系统所需的 DB2 Everyplace 文件。

表 2. 测试所需的 DB2 Everyplace 文件

操作系统	设备或仿真器上所需的文件
Palm OS	\db2everyplace\clients\palmos\database\DB2eCat.prc \db2everyplace\clients\palmos\database\DB2eCLI.prc \db2everyplace\clients\palmos\database\DB2eComp.prc \db2everyplace\clients\palmos\database\DB2eRunTime.prc \db2everyplace\clients\palmos\database\DB2eDMS.prc
Symbian OS V6.0	要进行仿真器测试，将文件 \db2everyplace\clients\symbian6\database\wins\DB2e.dll 复制到以下每个仿真器目录: \EPOCROOT%EPOC32\Release\wins\udeb\（用于调试仿真器） \EPOCROOT%EPOC32\Release\wins\urel\（用于释放仿真器） 要进行设备测试，使用 PsiWin 连接软件安装下列文件: \db2everyplace\clients\symbian6\database\armi\DB2e.sis

表 2. 测试所需的 DB2 Everyplace 文件 (续)

操作系统	设备或仿真器上所需的文件
Windows CE	为操作系统安装适当的库。 ARM 处理器: • V3.00 db2everyplace\clients\wince\database\wce300\armrel\DB2e.dll MIPS 处理器: • V3.00 \db2everyplace\clients\wince\database\wce300\mipsrel\DB2e.dll SH3 处理器: • V3.00 \db2everyplace\clients\wince\database\wce300\sh3rel\DB2e.dll Windows CE 仿真器: • V3.00 对于 Pocket PC 仿真器: db2everyplace\clients\wince\database\wce300\x86emrel\DB2e.dll 对于 Pocket PC 2002 仿真器: \db2everyplace\clients\wince\database\wce300\x86rel\DB2e.dll
Win32	将 \db2everyplace\clients\win32\database\x86\DB2e.dll 复制到应用程序的当前目录或系统的 PATH 环境变量所表示的目录中。
Neutrino	/db2e/database/x86/libdb2e.so（对于 x86 处理器类型）和 /db2e/database/strongarm/libdb2e.so（对于 strongarm 处理器类型）
Linux	/db2e/database/x86/libdb2e.so（对于 x86 处理器类型）和 /db2e/database/strongarm/libdb2e.so（对于 strongarm 处理器类型）

2. **对于 Linux 和 Neutrino**: 使用下列其中一个方法将 libdb2e.so 添加至库搜索路径。
   - 将 libdb2e.so 复制至库搜索路径中的目录。这可能需要 root 用户许可权。
   - 将 libdb2e.so 复制至另一目录并将该目录添加至库搜索路径。将目录永久添加至库搜索路径需要在 /etc/ld.config 中有一个条目。将目录临时添加至库搜索路径可通过适当地设置 LD_LIBRARY_PATH 环境变量实现。

     例如，输入以下命令（bash，libdb2e.so 在当前目录中）：export LD_LIBRARY_PATH=
3. 装入正在测试的应用程序的文件。例如，要在 Palm OS 上测试 Visiting Nurse 样本应用程序，装入 NurseInit.prc 和 Nurse.prc 文件。
4. 测试应用程序。

相关概念:

- 第 85 页的第 13 章，『样本 C/C++ 应用程序』

相关任务:

- 第 11 页的『准备、编译和链接 C/C++ 项目』

相关参考:

- 第 9 页的『支持的 C/C++ 开发工具』

- 第 11 页的『支持 C/C++ 的操作系统』
- 第 175 页的『DB2 CLI 函数摘要』

使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序

本主题概述如何使用 C/C++ 基于版本 8.1 的 IBM Sync Client C-API 来开发 DB2 Everyplace Sync Client 应用程序。 第 279 页的『IBM Sync Client C-API 函数摘要』提供所有 C API 函数的函数规范。

先决条件:

在开发工作站上安装 DB2 Everyplace。有关详细信息，请参阅《*DB2 Everyplace 安装与用户指南*》。

过程:

要使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序:

1. 定义包含下列各项的同步应用程序:
   - 将同步的数据;
   - 允许的操作;
   - 用户和用户组;
   - 数据安全性（例如，基于线路的数据加密和本地数据加密）

   有关要同步的数据定义和用户管理的更多详细信息，请参阅《*DB2 Everyplace Sync Server 管理指南*》。
2. 将 DB2 Everyplace Sync Client 头文件（`isyncore.h`）包含在 C 应用程序程序中，并遵循下列函数规范使用 DB2 Everyplace Sync Client C API 函数。
3. 准备、编译应用程序代码，并将其与 DB2 Everyplace Sync Client 操作系统库、`isyncconf` 和 `isyncore` 链接。
4. 测试应用程序:
   - 在操作系统的仿真器或设备上安装 DB2 Everyplace 库。
   - 在仿真器上测试应用程序（如果适用的话）。
   - 在设备上测试应用程序。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关参考:

- 第 9 页的『支持的 C/C++ 开发工具』

第 3 章 开发 DB2 Everyplace Java 应用程序

本章描述如何开发 DB2 Everyplace Java 应用程序。涉及的主题包括:

- 『支持 JDBC 接口的操作系统』
- 『开发 DB2 Everyplace Java 应用程序』

支持 JDBC 接口的操作系统

下列操作系统上支持 JDBC 接口:

- Palm OS
- Symbian OS
- Windows CE® for Pocket PC
- Win32（Windows 95、Windows 98、Windows NT、Windows 2000 和 Windows XP）
- QNX Neutrino
- Linux 和嵌入式 Linux

开发 DB2 Everyplace Java 应用程序

要使用 Java 开发 DB2 Everyplace 应用程序，可将 Java Software Developer's Kit 与用于 Java 的 DB2 Everyplace Java 数据库连接（JDBC）接口配合使用。

本主题提供使用 DB2 Everyplace 开发 Java 应用程序所需的任务的高级概述。

限制:

DB2 Everyplace 在 Symbian 上不支持多任务。为了存取来自第二个线程的数据库，必须关闭来自第一个线程的“连接”对象，才可在第二个线程中建立连接。线程之间不能共享同一“连接”对象。

先决条件:

访问 DB2 Everyplace 的 Java 应用程序使用 DB2 Everyplace JDBC 驱动程序。如果尚未在工作站上安装 Java 和 JDBC，则安装它们。

过程:

要使用 Java 开发 DB2 Everyplace 应用程序:

1. 导入 `java.sql` 包和其它必需的 Java 类。
2. 装入 DB2 Everyplace JDBC 驱动程序。类名为 `com.ibm.db2e.jdbc.DB2eDriver`。
3. 使用格式为 `jdbc:`*`subprotocol:subname`* 的 URL 连接至数据库。DB2 Everyplace *subprotocol* 为 `db2e`。如果数据库在 `c:\dir1\dir2` 中，则使用 URL `jdbc:db2e:c:/dir1/dir2/`。还可对 *subname* 使用相对路径。
4. 创建“语句”对象。
5. 存取数据库（应用程序逻辑执行到此处）:

- 使用“语句”对象执行 SQL 语句。
- 从返回的 ResultSet 对象检索数据（如果执行的 SQL 语句是一个查询）。

6. 通过关闭 ResultSet、“语句”和“连接”对象，释放数据库和 JDBC 资源。

相关概念:

- 第 87 页的『样本 Java 应用程序的概述』

相关任务:

- 第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』
- 第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』

第 4 章 开发 Java Sync Client 应用程序

本章描述如何开发 DB2 Everyplace Sync Client Java 应用程序。涉及的主题包括:

- 『支持 Java Sync API 的操作系统』
- 『IBM Java Sync API』

支持 Java Sync API 的操作系统

Java Sync API 在下列操作系统上可用:

- Win32
- Symbian OS
- Windows CE（带有 MIPS 和 ARM 处理器）
- Palm OS
- Linux
- QNX Neutrino

IBM Java Sync API

可以使用 “Java 数据库连接”（JDBC）和 Java 接口创建 Java 应用程序，以集成 “DB2 Everyplace 数据库” 和 Sync Server 功能。

有关随 DB2 Everyplace 支持的 IBM Java Sync API 提供的接口、类和异常的详细信息，参阅 `Clients\javadoc` 目录中的 Javadoc 文档。

相关概念:

- 第 87 页的『样本 Java 应用程序的概述』

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

DB2 Everyplace 同步提供程序的概述

本主题描述 DB2 Everyplace 支持的 Sync Client Java-API。API 是一组库，这些库允许开发者构建使数据在 DB2 Everyplace 和企业关系数据库之间双向同步的应用程序。它与 DB2 Everyplace Sync Server 一起工作来简化关系数据和文件的同步。Sync Server 提供冲突解决方法并管理数据在移动式 PDA、嵌入式设备或支持 MIDP 1.0 的设备之间的移动。

Sync Client Java API 由两种类型的同步提供程序组成:

- 第 20 页的『DB2 Everyplace 本机同步』
- 第 26 页的『DB2 Everyplace Java 同步』

样本文件中提供了有关如何基于这些提供程序在客户机设备上创建 Java 应用程序的信息。

相关任务:

- 『安装基于 JNI 的本机同步提供程序』
- 第 23 页的『安装和验证基于陷阱的本机同步提供程序』

相关概念:

- 第 111 页的『样本 Java 本机同步应用程序』
- 第 115 页的『样本 Java MIDP 同步应用程序』

DB2 Everyplace 本机同步

:

本机同步提供程序提供调用本机同步客户机库的 Java 接口。

注: 本机同步提供程序不支持此发行版中的线程安全，协调线程同步是应用程序的职责。

有两种类型的 DB2 Everyplace 本机同步提供程序:

- 基于 Java 本机接口（JNI）的本机同步提供程序
- 基于 Palm OS 陷阱的本地同步提供程序

相关任务:

- 『安装基于 JNI 的本机同步提供程序』
- 第 23 页的『安装和验证基于陷阱的本机同步提供程序』

相关概念:

- 第 19 页的『DB2 Everyplace 同步提供程序的概述』

安装 DB2 Everyplace 本机同步提供程序

本章描述如何安装 DB2 Everyplace Java 本机同步提供程序。涉及的主题包括:

- 『安装基于 JNI 的本机同步提供程序』
- 第 23 页的『安装和验证基于陷阱的本机同步提供程序』

安装基于 JNI 的本机同步提供程序

基于 JNI 的同步提供程序与支持 “Java 本机接口” 的 Java VM 一起工作。 .

在下列操作系统上支持此提供程序:

- Win32
- Symbian 发行版 6（对于 Nokia 9210/9290 Communicator 设备）
- Symbian 发行版 7（对于 Sony Ericsson P800 设备）
- Windows CE（对于 Pocket PC 设备）
- Linux
- QNX Neutrino

先决条件:

基于 JNI 的同步提供程序需要下列文件:

- isync4j.jar 文件
- 下列本机 Sync Client 二进制文件:
  - isyncore.dll
  - isyncconf.dll
  - imsadb2e.dll
  - imsafile.dll
  - imsaconfig.dll
  - wbxmllib.dll
  - isync4j.dll
  - isyncxpt.dll

如果应用程序在使用基于 JNI 的本机同步应用程序，必须导入下列 isync4j Java 包:

- com.ibm.mobileservices.isync
- com.ibm.mobileservices.isync.event
- com.ibm.mobileservices.isync.sql

验证在系统上安装了以下软件:

- DB2 Everyplace Sync Server V8
- DB2 Everyplace Sync Client 库版本 8

  有关更多信息，请参阅*《DB2 Everyplace 安装与用户指南》*。
- 支持“Java 本机接口”的 Java VM

阅读下列主题以获取有关如何在每个受支持的操作系统上安装基于 JNI 的同步提供程序的更多信息:

- 『在 Win32 上安装基于 JNI 的同步提供程序』
- 第 22 页的『在使用 Symbian V6 的 Nokia 9210/9290 Communicator 设备上安装基于 JNI 的同步提供程序』
- 第 22 页的『在 Windows CE 上安装基于 JNI 的同步提供程序』

在 Win32 上安装基于 JNI 的同步提供程序

要在 Win32 操作系统上安装基于 JNI 的同步提供程序，必须编译并运行 ISyncSample 程序。已在 Sun MicroSystems Java™ VM 和 IBM Java™ 2 Standard Edition Developer Kit 上测试了 Win32 设备的基于 JNI 的实现。

过程:

1. 编译 ISyncSample 程序。
   a. 将 PATH 系统变量更改为包含下列目录:

      ```
      <DB2e_InstDir>\Clients\Win32\database\x86
      <DB2e_InstDir>\Clients\Win32\sync
      ```

   b. 更改 CLASSPATH 变量以包含 isync4j.jar 文件:

      ```
      <DB2e_InstDir>\Clients\Win32\Sync\isync4j.jar
      ```

   c. 编译包括在 <DB2e_InstDir>\Clients\clientapisample\Java_API 目录中的样本文件。例如:

      ```
      javac ISyncSample.java
      ```

2. 编辑 isyncdb2e.properties 文件以指定服务器 URL、用户和密码。
3. 运行 ISyncSample 程序。
   a. 输入以下命令:

      ```
      java.exe ISyncSample <property file>
      ```

      其中 *<property file>* 是客户机数据库的属性文件。例如:

      ```
      java.exe -classpath .; isync4j.jar ISyncSample isyncdb2e.properties
      ```

相关任务:

- 第 20 页的『安装基于 JNI 的本机同步提供程序』

在使用 Symbian V6 的 Nokia 9210/9290 Communicator 设备上安装基于 JNI 的同步提供程序

要在使用 Symbian V6 的 Nokia 9210/9290 Communicator 设备上安装基于 JNI 的同步提供程序，必须编译并运行 ISyncSample 程序。已在 Symbian OS 6.0 PersonalJava JVM 上测试了使用 Symbian V6 的 Nokia 9210/9290 设备的基于 JNI 的实现。

过程:

1. 在工作站上编辑和编译 ISyncSample 程序。
   a. 编辑 ISyncSample.java 以将 isyncdb2e.properties 用作参数。
   b. 通过输入以下命令在类路径中使用 isync4j.jar 编译 ISyncSample.java:

      ```
      javac -classpath isync4j.jar ISyncSample.java
      ```

   c. 编辑 isyncdb2e.properties 以指定服务器 URL、用户和密码。
2. 运行 ISyncSample 程序。
   a. 确保在设备上安装了 DB2 Everyplace 数据库和 Sync Client 库。
   b. 将 ISyncSample.class 和 isyncdb2e.properties 文件复制至设备上的 C:\System\Apps\ISync 目录。
   c. 使用“Windows 文件管理器”，定位并选择 isync4j.jar 文件。单击**输入**。使用安装在 Nokia 设备上的“重定向”程序来传送 Java 程序的输出，然后在控制台上显示此输出或将此输出写入某个文件。

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

在 Windows CE 上安装基于 JNI 的同步提供程序

要在 Windows CE 操作系统上安装基于 JNI 的同步提供程序，必须编译并运行 ISyncSample 程序。已在 OTI J9 JVM 上测试了用于 Windows CE 设备的基于 JNI 的同步提供程序。

过程:

1. 在工作站上编译 ISyncSample 程序。
   a. 输入以下命令以在类路径中使用 isync4j.jar 编译 ISyncSample.java:

      ```
      javac -classpath isync4j.jar ISyncSample.java
      ```

   b. 编辑 isyncdb2e.properties 以指定服务器 URL、用户和密码。

2. 运行 ISyncSample 程序。
   a. 验证在设备上安装了 J9 Java 虚拟机（JVM）运行时环境（例如，\wsdd）。此外，还必须安装 DB2 Everyplace 和 Sync Client 库。
   b. 将 ISyncSample.class 和 isyncdb2e.properties 文件复制至设备（例如，\）。
   c. 使用下列两种方法的其中一种来调用类路径中具有 isync4j.jar 的 ISyncSample 程序。

      Java 控制台

      输入以下命令:

      ```
      j9.exe -bp:\wsdd\classes.zip -cp:\wsdd;\Windows\isync4j.jar ISyncSample
        <property file>
      ```

      例如:

      ```
      j9.exe -bp:\wsdd\classes.zip -cp:\wsdd;\Windows\isync4j.jar
       ISyncSample isyncdb2e.properties
      ```

      Windows 快捷方式

      在工作站上创建和编辑称为 ISyncSample.lnk 的 Windows 快捷方式。例如:

      ```
      255#"\wsdd\j9.exe" "-bp:\wsdd;\Windows\isync4j.jar;\wsdd\classes.zip"
      "ISyncSample" "isyncdb2e.properties"
      ```

      在单个行上输入快捷方式，并用双引号括起每个字段。您输入的第一个字段*必须*是可执行文件的名称。您指定的文件和目录必须是全限定的。
   d. 运行样本程序并验证同步数据驻留在属性文件中指定的目标目录中。

相关任务:

• 第 20 页的『安装基于 JNI 的本机同步提供程序』

安装和验证基于陷阱的本机同步提供程序

基于陷阱的本机同步提供程序仅与 Palm OS 平台上的 WebSphere Studio Device Developer（WSDD）J9 JVM 配合使用。

本主题描述 DB2 Everyplace isync4j for Palm OS 可与 J9 的 jclMidp（J2ME MIDP）配置配合使用的方式。此同步提供程序引用 com.ibm.oti.palmos 包，所以它仅在 WSDD J9 JVM for PalmOS V1.5 或以上版本上运行。

下表描述用于在 Palm 设备上安装 API 的程序的所在位置，其中 %DSYINSTDIR% 表示 DB2 Everyplace 的安装目录。

目录	描述
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib	包含 isync4j for Palm OS Java 类的文件夹。这些类是在实现期间导入的。
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/sample	包含样本 isync4j 应用程序的源代码的文件夹。
%DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/bin/ISyncSample.prc	与 J9 Palm OS CLDC 库配合使用的样本 isync4j 应用程序。

先决条件:

基于陷阱的本机同步提供程序需要下列 Sync Client 本机共享库和 DB2 Everyplace 库:

- isyncore.prc
- isyncconf.prc
- imsaconfig.prc

- imsafile.prc
- imsadb2e.prc
- wbxmllib.prc
- isyncxpt.prc

此外，还需要在设备上安装 J9 Palm OS JVM 二进制文件。

如果应用程序在使用基于陷阱的本机同步提供程序，必须导入下列 isync4j Java 包:

- com.ibm.mobileservices.isync
- com.ibm.mobileservices.isync.db2e.sti
- com.ibm.mobileservices.isync.event
- com.ibm.mobileservices.isync.sql

验证在系统上安装了以下软件:

- Palm OS V3.5 或更高版本（至少具有 8 MB 内存）
- WebSphere Studio Device Developer（WSDD）V4.0
- DB2 Everyplace 数据库 Palm OS 版的版本 7.1 或更高版本
- DB2 Everyplace Sync Client 库版本 8.1 或更高版本

安装了 WSDD 之后，必须设置 Palm OS 目标。要设置 Palm OS 目标，请参阅 WSDD Development Environment & Tools Product Documentation （位于称为 “Getting Started with Palm OS Targets” 的一章中）。WSDD 文档位于产品 CD-ROM 上的 IBM\wsdd\wsdd4.0\doc\wsddCustomer.pdf 中。最后，通过构建并运行 WSDD 样本应用程序验证 WSDD 安装正确。

过程:

要验证 WSDD 正确安装:

1. 为 isync4j 样本应用程序创建新的项目:
   a. 在 WSDD 中打开 “Java 透视图”。
   b. 选择**文件 -> 新建 -> 其它**。
   c. 选择 “J2ME for J9 的向导” 和 “创建 MIDlet 套件”。
   d. 在 “MIDlet 套件创建” 对话框中命名定制项目、MIDlet 名称和 MIDlet 类名。
   e. 单击**下一步**。
   f. 再次单击**下一步**以转至 “Java 设置”。
   g. 在 “Java 设置” 上，单击**库**选项卡并单击**创建文件夹...**。在 “新建类文件夹” 对话框中输入 **lib**。
   h. 单击**完成**。
2. 导入 DB2 Everyplace ISYNC4J Java 类并设置构建路径。
   a. 在**包**视图中单击项目，然后单击菜单项**文件 -> 导入...**
   b. 导入 %DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib 文件夹并选择 %DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/lib 作为源目录。
   c. 展开 lib 目录并对 /lib 下的 com 目录选择复选框。在**文件夹**: 字段的**选择导入资源的目的地**: 下输入后跟 /lib 的项目名称。例如，如果项目名是 ISyncSample，则该字段应显示为 ISyncSample/lib。

d. 单击**完成**。

e. 展开 lib 文件夹，应看到以下 ISYNC4J Java 包:

```
com.ibm.mobileservices.isync
com.ibm.mobileservices.isync.db2e.sti
com.ibm.mobileservices.isync.event
com.ibm.mobileservices.isync.sql
```

3. 通过构建并运行样本应用程序来验证 isync4j 库设置。

a. 导入样本应用程序。

- 在**包**视图中单击该项目的 src 文件夹，然后从主菜单中单击**文件 > 导入**。
- 导入 ISyncSample.java。选择 %DSYINSTDIR%/Clients/PalmOS/Sync/isync4j-palm/samples/ISyncSample/ 作为源目录，然后选择 ISyncSample.java 的复选框。验证导入资源的目的地是否是 <project>/src。）.

b. 为样本应用程序创建构建文件。

- 在编辑器中，单击**包含 / 排除**选项卡，然后单击**新建**。
- 对主类输入 **ISyncSample**，对平台选择 **J9 for Palm 68k**。单击**下一步**。
- 对 “App 名称” 输入 “创建程序” 标识和 ISyncSample。单击**下一步**两次。
- 选择 **PalmOS 仿真器上的 Prc 应用程序**。单击**完成**。

c. 修改 ISyncSample.jxeLinkOptions 文件。

- 在**包**视图中展开项目的 **palm68k** 文件夹。
- 双击 ISyncSample.jxeLinkOptions。
- 在编辑器中，单击**包含 / 排除**选项卡，然后单击**新建**。
- 对 “规则” 模式输入 **com.ibm.mobileservices.isync.db2e.sti.DB2eISyncProvider**，然后单击**确定**。
- 在编辑器内，单击**源**选项卡
- 输入 **-vmOption -ms:15** 来设置堆栈大小。
- 保存更改。

d. 运行样本应用程序。

- 单击菜单中的**运行**图标，然后从构建文件中选择**运行 –> 构建 –> 启动**。
- 选择样本应用程序的目标，并单击**完成**。
- 如果没有出错，Palm OS 仿真器应启动并运行该应用程序。

现在可以创建自己的应用程序。当创建新的应用程序时，将 DB2 Everyplace isync4j 的新项目名包括在项目的构建路径中。在创建应用程序的构建文件之后，修改其中的 jxeLinkOptions 文件以满足应用程序的需求。

相关概念:

- 第 19 页的『DB2 Everyplace 同步提供程序的概述』

DB2 Everyplace Java 同步提供程序

本章描述 DB2 Everyplace Java 同步提供程序。涉及的主题包括:

- 『DB2 Everyplace Java 同步』
- 『DB2 Everyplace J2ME MIDP 同步』
- 第 27 页的『DB2 Everyplace Java Sync Client for Cloudscape』

DB2 Everyplace Java 同步

:

Java 同步提供程序提供 Java 接口以调用 Java 同步客户机库。

有两种类型的 DB2 Everyplace Java 同步提供程序:

- 『DB2 Everyplace J2ME MIDP 同步』
- 第 27 页的『DB2 Everyplace Java Sync Client for Cloudscape』

相关任务:

- 第 20 页的『安装基于 JNI 的本机同步提供程序』
- 第 23 页的『安装和验证基于陷阱的本机同步提供程序』

相关概念:

- 第 19 页的『DB2 Everyplace 同步提供程序的概述』

DB2 Everyplace J2ME MIDP 同步

J2ME MIDP ISync Client 允许您构建使预订与“MIDP 记录存储管理系统”（RMS）同步的应用程序。J2ME MIDP ISync Client 是一组库，它与 DB2 Everyplace Sync Server 一起工作来简化企业数据库和支持 MIDP 1.0 的设备之间关系数据的同步。Sync Server 管理数据与 MIDP 设备之间的移动。

本主题包括关于 J2ME MIDP ISync Client 的下列信息:

- 安装 J2ME MIDP ISync Client 所需的 Web 服务器软件
- 在 Motorola iDEN 电话上运行 J2ME MIDP ISync Client 所需的软件和硬件
- J2ME MIDP ISync Client 安装目录布局

安装 J2ME MIDP ISync Client 所需的 Web 服务器软件:

要安装 J2ME MIDP Sync Client，需要下列两种软件产品中的其中一个:

- WebSphere Application Server 高级单服务器版的版本 4.x 或更高版本。可以从 IBM Web 站点 http://www-3.ibm.com/software/webservers/appserv/advanced.html 下载此软件的免费试用版本。
- Apache Tomcat 版本 4.0.x 或更高版本。可以从 http://jakarta.apache.org/tomcat/ 下载此软件的免费拷贝。

在 Motorola iDEN 电话上运行 J2ME MIDP ISync Client 所需的软件和硬件:

要在 Motorola iDEN 电话上安装和运行 MIDP 同步提供程序，需要下列硬件和软件:

- Sun Microsystems JavaTM 2 Platform Micro Edition, Wireless Toolkit
- “iDEN Update”和数据电缆（用于在电话上装入应用程序）

- Apache ANT
- RetroGuard Ofuscator
- isync4j Java 包（与 DB2 Everyplace 包括在一起）
  - com.ibm.mobileservices.isync
  - com.ibm.mobileservices.isync.midp
  - com.ibm.mobileservices.isync.event

J2ME MIDP ISync Client 安装目录布局:

J2ME MIDP ISync Client 安装过程创建四个初始目录:

- `bin – %DSYINSTDIR%\Clients\Midp\bin`，包含从命令行运行 WTK 仿真器的脚本。
- `lib – %DSYINSTDIR%\Clients\Midp\lib` 包含 MIDP ISync API jar、Servlet for MIDP、FilterServlet.jar 文件和具有相关 JAD 文件的样本 MIDLet。
- `docs – %DSYINSTDIR%\Clients\Midp\doc` 包含 J2ME MIDP ISync Client Javadoc。
- `samples – %DSYINSTDIR%\Clients\Midp\samples`，包含样本 isync4j 应用程序的源代码，其中 `%DSYINSTDIR%` 是 DB2 Everyplace 的安装目录。

如果决定要重新编译样本，您将查找几个 build*Classes 目录，它们用于预验证和混淆（obfuscation）。

相关任务:

- 第 20 页的『安装基于 JNI 的本机同步提供程序』
- 第 23 页的『安装和验证基于陷阱的本机同步提供程序』

相关概念:

- 第 19 页的『DB2 Everyplace 同步提供程序的概述』

DB2 Everyplace Java Sync Client for Cloudscape

DB2 Everyplace Java Sync Client for Cloudscape 允许您构建将预订与 Cloudscape 数据库同步的应用程序。Java Sync Client for Cloudscape 是一组库，它们与 DB2 Everyplace Java Sync Server 一起工作来简化企业数据库与 Cloudscape 客户机之间的关系数据的同步过程。Sync Server 管理数据在设备上的移动。

本主题包括有关 Java Sync Client for Cloudscape 的下列信息:

- 运行 Java Sync Client for Cloudscape 所需的软件
- Java Sync Client for Cloudscape 目录布局
- 设置 CLASSPATH 环境变量

运行 Java Sync Client for Cloudscape 所需的软件:

为了运行 Java Sync Client for Cloudscape，需要下列软件产品:

- DB2 Everyplace V8.1.4 或更高版本
- Cloudscape 数据库

Java Sync Client for Cloudscape 安装目录布局:

Java Sync Client for Cloudscape 文件在下列目录中:

- `%DSYINSTDIR%\Clients\javaclient` 包含 Cloudscape ISync API JAR 文件。
- `%DSYINSTDIR%\doc\javadoc\javaclient` 包含 Java Sync Client for Cloudscape Javadoc。

设置 CLASSPATH 环境变量:

要使用 Java Sync Client for Cloudscape, 将 CLASSPATH 环境变量设置为包括下列文件:

- Cloudscape 安装中的 Cloudscape JAR 文件（`db2j.jar` 和 `db2jtools.jar`）。
- Cloudscape ISync API JAR 文件（`db2jisync.jar`）。
- 可选: 样本应用程序（ISyncSample 和 GoISyncConsole）。

相关任务:

- 第 122 页的『编译和运行 GoISyncConsole 样本 Java 同步应用程序』

第 5 章 开发 Visual Basic 应用程序

本章提供有关使用 Visual Basic 开发 DB2 Everyplace 应用程序的信息。涉及的主题包括:

- 『开发 DB2 Everyplace Visual Basic 应用程序』
- 第 30 页的『支持 Visual Basic 接口的操作系统』

开发 DB2 Everyplace Visual Basic 应用程序

要使用 Visual Basic 开发 DB2 Everyplace 应用程序，应使用 DB2 Everyplace CLI/ODBC 接口。本主题提供要使用 DB2 Everyplace 开发 Visual Basic 应用程序则必须完成的任务的高级概述。

限制:

在使用 Visual Basic 为 DB2 Everyplace 开发应用程序时，应考虑下列限制和需求:

- 不要直接在应用程序代码中使用 SQLAllocHandleVer 函数。SQLAllocHandleVer 由 DB2 Everyplace 以内部方式使用。如果在应用程序代码中使用它，可能会导致程序故障。
- 调试可能会因 Visual Basic 装入和处理对 DLL 内部的函数的调用的方式而不能工作。
- 在 `db2e.dll` 中调用 DB2 Everyplace 函数的 Visual Basic 函数必须带有 “On Error Resume Next” 语句，否则程序将不能正确工作。

过程:

开发 DB2 Everyplace Visual Basic 应用程序的基本步骤是:

1. 创建新的 Visual Basic 项目。
2. 将文件 `db2ecli.bas`（从 DB2 Everyplace Visual Basic 项目目录）复制到项目文件夹中，并将该文件插入到 Visual Basic 项目中。
3. 将 `DB2e.dll` 复制到项目文件夹中。如果不想将 `DB2e.dll` 放在项目文件夹中，则在 `db2ecli.bas` 文件中的函数声明中修改 `DB2e.dll` 的路径。
4. 编写您自己的应用程序代码。可使用 DB2 Everyplace 样本 Visual Basic 程序来为您提供帮助。
5. 通过选择菜单项**文件 -> 生成***项目*来为应用程序创建可执行程序。

相关概念:

- 第 101 页的『样本 Visual Basic 应用程序的概述』

相关参考:

- 第 175 页的『DB2 CLI 函数摘要』
- 第 30 页的『支持 Visual Basic 接口的操作系统』

支持 Visual Basic 接口的操作系统

下列操作系统完全支持 Visual Basic 接口:

- Windows CE® for Pocket PC
- Win32（Windows 95、Windows 98、Windows NT、Windows 2000 和 Windows XP）

第 6 章 开发 JSP 应用程序

本章提供有关使用 JavaServer Pages 开发 DB2 Everyplace 应用程序的信息。涉及的主题包括:

- 『支持 JSP 的操作系统』
- 『开发 DB2 Everyplace JSP 应用程序』
- 第 32 页的『DB2 Everyplace JSP 支持概述』

支持 JSP 的操作系统

JSP 支持对于下列操作系统可用:

- Win32（Windows® NT® 和 Windows® 2000®）
- Windows CE® for Pocket PC
- Symbian OS

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关概念:

- 第 32 页的『DB2 Everyplace JSP 支持概述』
- 第 107 页的第 16 章, 『样本 JSP 应用程序』

开发 DB2 Everyplace JSP 应用程序

DB2 Everyplace 支持 JavaServer Pages（JSP）, 使您可以容易地创建 DB2 Everyplace 基于 Web 的应用程序。您创建的应用程序独立于操作系统, 且可以在移动式设备上以断开连接方式运行或在与“局域网”断开连接时运行。

JSP 技术提供简化且快速的方法来开发和维护动态 Web 页面。JSP 技术将用户界面与与内容生成分开, 以便您可以创建和更新页面布局而不必更改底层的动态内容。

JSP 使用 JDBC 技术。您使用 JSP 创建的基于 Web 的应用程序可以通过 DB2 Everyplace JDBC 驱动程序存取 DB2 Everyplace 数据库。

有关如何生成存取 DB2 Everyplace（使用 WebSphere Studio）的 JSP 页的信息, 参阅 `<DB2Everyplace>\SDK\JSP\doc` 中的附带文档。

通过遵循第 39 页的『在 Windows 工作站上运行 JSP 应用程序』中的步骤来测试 JSP 应用程序。都要在工作站上运行 JSP 应用程序之后才将其传送至设备。在工作站上运行 JSP 应用程序将对某些应用程序文件进行必要的预处理。

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关概念:

- 『DB2 Everyplace JSP 支持概述』
- 第 107 页的第 16 章, 『样本 JSP 应用程序』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

DB2 Everyplace JSP 支持概述

DB2 Everyplace JSP 支持包含两个组件:

- 微型 HTTP Web 服务器
- JSP 处理器

微型 HTTP Web 服务器接收来自 Web 浏览器的请求并将响应发送回 Web 浏览器(使用 HTTP 1.1 作为请求和响应的协议)。JSP 处理器解析 JSP 文件, 生成对应的 Java 源代码, 并编译源代码。Java 源代码可以包含请求 JSP 页时生成动态内容的 JavaBean。当请求 JSP 页时, 微型 HTTP Web 服务器执行对应的 Java 代码, 并将输出发送回 Web 浏览器作为对请求的响应。

当您在 Web 浏览器中输入诸如 `http://localhost/request.jsp` 之类的 URL(其中 **request.jsp** 是您正在请求的 JSP 页)时, Web 浏览器会将该请求发送至微型 HTTP Web 服务器。如果下列其中一项条件成立, 将请求转发到 JSP 处理器:

- 没有与 JSP 页对应的 .class 文件(例如, 如果 JSP 页是新创建的页)。
- 存在与 JSP 页对应的 .class 文件, 但 JSP 文件上的时间戳记比 .class 文件上的时间戳记更新(例如, 如果已修改了 JSP 页)。

  注: 对于 **request.jsp**, 对应的 .class 文件是 **_request_jsp_.class**。

如果将请求转发到 JSP 处理器且 JSP 文件语法有效, 则微型 HTTP Web 服务器将输出发送到 Web 浏览器, 指示 JSP 页是否有效。单击输出中的 **request.jsp** 链接以查看 JSP 页。

如果将请求转发到 JSP 处理器且 JSP 文件语法无效, 则微型 HTTP Web 服务器将诊断信息发送到 Web 浏览器。

如果不需要将请求转发到 JSP 处理器, 则微型 HTTP Web 服务器执行 JSP 页的对应 .class 文件并将输出发送到 Web 浏览器。

应在 Windows 工作站上进行 JSP 应用程序开发。在整个开发过程测试 JSP 页是重要的。JSP 处理器将捕捉 JSP 页中的任何语法错误。在修正错误之后, 通过单击 Web 浏览器中的“刷新”按钮再次测试 JSP 页。可能需要删除 Web 浏览器的 Temporary Internet Files 文件夹中的文件或高速缓存才能反映 JSP 页的更改。完成应用程序后, 可将该应用程序传送至设备并在该设备上运行它。

注: JSP 处理器在工作站上运行, 在设备上不需要它。

相关任务:

- 第 37 页的『将 JSP 应用程序传送至 Windows CE 设备』
- 第 33 页的『在 Windows 工作站上验证 JSP 支持』

- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关概念:

- 第 31 页的『开发 DB2 Everyplace JSP 应用程序』
- 第 107 页的第 16 章, 『样本 JSP 应用程序』

为 JSP 开发设置

本节提供有关如何设置以使用 Java Server Pages 开发 DB2 Everyplace 应用程序的信息。涉及的主题包括:

- 『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』
  - 第 34 页的『在 Windows CE 设备上安装 J9 JVM 运行时环境』
  - 第 35 页的『在 Windows CE 设备上安装和验证 JSP 支持』
- 第 36 页的『在 Symbian OS V6 设备上安装和验证 JSP 支持』

在 Windows 工作站上验证 JSP 支持

先决条件:

验证以下软件已安装在 Windows 工作站上:

- DB2 Everyplace Software Development Kit
- Java 2 Standard Development Kit, Standard Edition
  - 版本 1.1.8, 如果目标是 Symbian OS V6 设备的话
  - 版本 1.2.2 或更新版本（对于其它目标）
- Web 浏览器
  - Internet Explorer V5.50 或更新版本或
  - Netscape Navigator V6.2.1 或较早版本

过程:

要验证 Windows 工作站设置为使用 DB2 Everyplace JSP 支持:

- 运行 Visiting Nurse 样本 JSP 应用程序:
  1. 启动微型 HTTP Web 服务器:
     a. 打开 MS-DOS 窗口。
     b. 使用 cd 命令切换至 `<DB2Everyplace>\SDK\JSP\Win32` 目录。
     c. 输入 `runJspServer`。
  2. 打开 Web 浏览器以访问以下 URL: http://localhost/VisitingNurse/schedule.jsp
     或者, 可输入 URL http://localhost/ 并浏览至 VisitingNurse/schedule.JSP 页。

如果成功地将工作站设置为使用 DB2 Everyplace JSP 支持, Visiting Nurse Schedule 表将出现在 Web 浏览器中。

相关任务:

- 第 39 页的『在 Windows 工作站上运行 JSP 应用程序』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

在 Windows CE 设备上为 JSP 开发进行设置

先决条件:

验证在工作站上安装了以下软件:

- DB2 Everyplace Software Development Kit

验证设备上的 \Windows 目录是否包含下列文件:

```
<DB2Everyplace>\Clients\WinCE\database\ver\proc\CryptoPlugin.dll
<DB2Everyplace>\Clients\WinCE\database\ver\proc\DB2e.dll
<DB2Everyplace>\Clients\WinCE\database\ver\proc\DB2eJDBC.dll
<DB2Everyplace>\Clients\WinCE\database\jdbc\db2ejdbc.jar
```

其中 *ver* 是设备上的 Windows CE 操作系统的版本号，而 *proc* 是处理器类型。

验证在设备上安装了以下软件:

- Web 浏览器

过程: :

1. 在设备上安装 J9 运行时环境。
2. 在设备上安装和验证 DB2 Everyplace JSP 支持。

相关任务:

- 第 40 页的『在 Windows CE 设备上运行 JSP 应用程序』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

在 Windows CE 设备上安装 J9 JVM 运行时环境

目前，在 Pocket PC 上仅支持 StrongARM 设备。如果设备具有不同的处理器类型，可尝试使用支持 JNI 的另一 JVM（例如，Sun PersonalJava、Insignia Jeode 和 NSIcom CrEme）。如果使用 J9 JVM 之外的 JVM，必须相应地修改 \SDK\JSP\WinCE\MiniHttpServer.lnk。

安装 J9 允许您运行样本 JSP 应用程序。可能需要安装附加 J9 文件以用于您自己的应用程序。

此任务是 “在 Windows CE 设备上为 JSP 开发进行设置” 这一主任务的一部分。完成这些步骤后，请返回至『在 Windows CE 设备上为 JSP 开发进行设置』。

过程:

要安装 J9 JVM 运行时环境:

1. 在工作站上安装 WebSphere Studio Device Developer V5.5（WSDD）。WSDD 的试用版本可从以下地址下载: http://www.ibm.com/software/pervasive/products/wsdd/。在上面的步骤中，<WSDD> 指的是安装 WSDD 的目录。
2. 在 WSDD 中，使用 “更新管理器” 来安装 WCE Tooling for WSDD。

a. 单击**帮助 —> 软件更新 —> 更新管理器**以打开“安装／更新透视图”。

b. 在“功能部件更新”视图中，展开下列节点: **要访问的站点 -> WebSphere 定制环境 -> WebSphere 定制环境**。

c. 单击 WCE Tooling for WSDD 5.5.0。

d. 在“预览”视图中，单击**安装**按钮并遵循用于安装的步骤。

e. 通过遵循类似步骤安装下列附加功能部件:

- WCE jclMax 类库
- WCE 数据库使能器库
- WCE 个人配置类库

3. 在设备上创建下列目录结构:

```
\wsdd\bin
\wsdd\lib
\wsdd\lib\jclMax
```

4. 将下列文件复制至 `\wsdd\bin`:

```
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9.exe
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9dyn20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9int20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9max20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9prt20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9thr20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9vm20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\j9zlib20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\iverel20.dll
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\arm\ive\bin\swt-win32-2104.dll
```

5. 将以下文件复制至 `\wsdd\lib`:

```
<WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\charconv.zip
```

6. 将下列文件复制至 `\wsdd\lib\jclMax`:

```
<WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\jclMax\classes.zip
<WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\jclMax\database_enabler.jar
<WSDD>\wsdd5.0\ive\runtimes\common\ive\lib\jclMax\locale.zip
<WSDD>\wsdd5.0\ive\runtimes\pocketpc\common\ive\lib\jclMax\prsnlwin.jar
```

返回至第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』。

在 Windows CE 设备上安装和验证 JSP 支持

此任务是“在 Windows CE 设备上为 JSP 开发进行设置”这一主任务的一部分。完成这些步骤后，请返回至第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』。

过程:

要安装和验证 JSP 支持: :

1. 将以下文件安装到设备上:

```
<DB2 Everyplace>\SDK\JSP\WinCE\DB2eJSP.CAB
```

2. 通过运行 Visiting Nurse 样本 JSP 应用程序验证设备设置为使用 DB2 Everyplace JSP 支持:

a. 可选（根据 JVM）: 在设备上修改 `MiniHttpServer.lnk`。如果在使用 J9 JVM 之外的 JVM，则需要修改快捷方式。快捷方式开始的数字为“#”字符后的字符数。“#”字符后的最大字符数为 256。

b. 启动微型 HTTP Web 服务器:

1) 打开“文件资源管理器”。

2) 导航至根目录。

3) 单击 MiniHttpServer 快捷方式。

c. 打开 Web 浏览器以访问以下 URL: http://localhost/VisitingNurse/schedule.jsp

或者，可输入 URL http://localhost/ 并浏览至 VisitingNurse/schedule.JSP 页。

如果成功地将设备设置为使用 DB2 Everyplace JSP 支持，Visiting Nurse Schedule 表将出现在 Web 浏览器中。

返回至第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』。

在 Symbian OS V6 设备上安装和验证 JSP 支持

先决条件:

验证在工作站上安装了以下软件:

- DB2 Everyplace Software Development Kit

验证在设备上安装了以下软件:

- Web 浏览器
- Java 运行时环境（Java.sis）

  Java.sis 可从因特网下载。较旧设备可能会在该设备的销售包 CD-ROM 上提供 Java.sis。

验证设备上的 \system\libs 目录包含下列文件:

- <DB2Everyplace>\Clients\Symbian6\database\armi\CryptoPlugin.dll
- <DB2Everyplace>\Clients\Symbian6\database\armi\DB2e.dll
- <DB2Everyplace>\Clients\Symbian6\database\armi\db2ejdbc.dll
- <DB2Everyplace>\Clients\Symbian6\database\armi\ECSPKCS11.DLL

验证设备上的 \system\java\ext 目录包含以下文件:

- <DB2Everyplace>\Clients\Symbian6\database\armi\db2ejdbc.jar

过程:

要在 Symbian OS V6 设备上安装和验证 JSP 支持:

1. 将以下文件安装到设备上: <DB2Everyplace>\SDK\JSP\Symbian6\DB2eJSP.sis。
2. 通过运行 Visiting Nurse 样本 JSP 应用程序验证设备设置为使用 DB2 Everyplace JSP 支持:
   a. 启动微型 HTTP Web 服务器:
      1) 转至设备上的“附加”屏幕。
      2) 启动 DB2eJSP 应用程序。
   b. 打开 Web 浏览器以访问以下 URL: http://localhost/VisitingNurse/schedule.jsp

      或者，可输入以下 URL: http://localhost/ 并浏览至 VisitingNurse/schedule.JSP 页。

如果成功地将设备设置为使用 DB2 Everyplace JSP 支持，Visiting Nurse Schedule 表将出现在 Web 浏览器中。

相关任务:

- 第 40 页的『在 Symbian OS V6 设备上运行 JSP 应用程序』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

将 JSP 应用程序传送至 Windows CE 设备

先决条件:

在将 JSP 应用程序传送至设备之前，完成下列任务:

- 在设备上安装 JSP 支持。
- 在工作站上开发和测试应用程序。

过程:

要将 JSP 应用程序传送至 Windows CE 设备:

1. 将应用程序使用的数据库复制至设备，如果它在设备上还不存在的话。确保将数据库置于应用程序期望它位于的目录中。即，由应用程序连接至的 URL 指定的目录。
2. 将应用程序文件复制至对设备的 `MiniHttpConfig.properties` 中的 JspPath 属性指定的目录。如果应用程序在工作站上 JspPath 下的子目录中，则在设备上的 JspPath 下创建相同的子目录结构。在将应用程序文件复制至设备时使用下列规则:

表 3.

工作站上的文件	要复制至设备的文件
<page>.jsp	<page>_jsp_.class
<webapp>\web.xml	<webapp>\<webapp>_config_.class

注意:

- 当在工作站上运行应用程序时，JSP 处理器将生成要复制至设备的 .class 文件。它们与相应的 `.jsp` 和 `web.xml` 文件位于同一目录中。
- 应用程序可能不包括 `web.xml` 文件。WebSphere Studio Application Developer 生成的应用程序使用 `web.xml` 文件。
- 快捷方式的最大长度为 256 个字符（在文件开始处的 “#” 字符之后）。要保持在最大长度以下，将下列 WebSphere Studio Application Developer 样本文件复制至根目录而不是样本应用程序的目录:
  - `dbbeans.jar`
  - 任何 ViewBean 类文件

  要运行使用这些文件的 WebSphere Studio 生成的应用程序，`MiniHttpServer.lnk` 中的类路径必须包括 `dbbeans.jar` 以及包含 ViewBean 类文件的任何目录。
- 还应将 `.jsp` 或 `web.xml` 文件以外的应用程序文件复制至该设备。

例如，要将 Visiting Nurse 样本 JSP 应用程序复制至该设备:

a. 将 Visiting Nurse 样本数据库复制至设备:
   1) 在设备上创建以下目录结构: `\sample\data`。这是由应用程序连接至的 URL 指定的目录。

2) 将工作站上的 <DB2Everyplace>\SDK\JSP\sample\data 目录的内容复制至设备上的 \sample\data。

b. 将应用程序文件复制至设备:

1) 在设备上创建以下目录结构: \sample\jsp。这是对缺省 MiniHttpConfig.properties 文件中的 JspPath 指定的目录。

2) 在 \sample\jsp 下创建子目录 \VisitingNurse。

3) 将下列文件复制至 \sample\jsp\VisitingNurse:

```
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_schedule_jsp_.class
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_contact_jsp_.class
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_medrecord_jsp_.class
<DB2Everyplace>\SDK\JSP\sample\jsp\VisitingNurse\_person_jsp_.class
```

相关任务:

- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

运行 JSP 应用程序

本节描述如何运行 JSP 应用程序。涉及下列主题:

- 可选(根据工作站或应用程序配置): 『配置微型 HTTP Web 服务器』
- 第 39 页的『在 Windows 工作站上运行 JSP 应用程序』
- 第 40 页的『在 Windows CE 设备上运行 JSP 应用程序』
- 第 40 页的『在 Symbian OS V6 设备上运行 JSP 应用程序』

配置微型 HTTP Web 服务器

过程:

下列文件是 Windows 工作站的缺省 MiniHttpConfig.properties 文件。可以使用此缺省文件或修改它来满足您的应用程序和系统的需求。还可对设备修改缺省 MiniHttpConfig.properties 文件。

```
# Mini HTTP Web server properties - Win32

# JspPath
# Specifies: the path that contains the JSP pages (.jsp and .class files).
# Default: JspPath=<directory where the mini HTTP Web server is started>
#
# Note: use \\ to denote the directory separator
#
JspPath=sample\\jsp

# Port
#
# Specifies: port that the server listens on
# Default: Port=80
#
# Note: If you are not using the default Port, start URL requests with:
#  http://localhost:Port/
# (instead of http://localhost/) where Port is the specified Port.

# Mime
#
# Specifies: Mime types
# Default: Mime=text/html wml htm html,text/plain txt,image/gif gif,image/jpeg jpg
```

```
#
# Note: Additional Mime types can be added using the following format:
# Mime=mime_type_A ext1 ext2 ext3 ...,mime_type_B ext4 ext5 ...,...
#
Mime=application/octet-stream exe class,image/jpeg jpeg jpg

# LogFile
#
# Specifies: log file for the server
# Values:
# "" - log entries are written to the console
# "no" - no log entries are kept
# "<log_file_name>" - log entries are written to <log_file_name>
# Default: LogFile=
#
LogFile=JspServer.log

# Index
#
# Specifies: the index page (displayed if http://localhost is requested)
# Values:
# "" - the JspPath directory will be loaded
# "no" - no page will be loaded
# "<index_file>" - <index_file> will be loaded
# Default: Index=
```

相关任务:

- 第 36 页的『在 Symbian OS V6 设备上安装和验证 JSP 支持』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』
- 第 31 页的『支持 JSP 的操作系统』

在 Windows 工作站上运行 JSP 应用程序

过程:

要在 Windows 工作站上运行 JSP 应用程序:

1. 如果需要，在 `<DB2Everyplace>\SDK\JSP\Win32` 目录中配置 `MiniHttpConfig.properties` 文件。
2. 启动微型 HTTP Web 服务器:
   a. 打开 MS-DOS 窗口。
   b. 使用 “cd” 命令将目录切换至 `<DB2Everyplace>\SDK\JSP\Win32`。
   c. 输入 `runJspServer`。
3. 打开 Web 浏览器并输入 URL 来启动 JSP 应用程序。例如，如果应用程序的开始页面是 `start.jsp`，则输入 `http://localhost/start.jsp`。如果在 `MiniHttpConfig.properties` 文件中配置了 `Port` 属性，则输入 `http://localhost:Port/start.jsp` 代替，其中 “Port” 是指定的端口号。
4. 要在运行完 JSP 应用程序时停止微型 HTTP Web 服务器:
   a. 转至在其中启动服务器的 MS-DOS 窗口。
   b. 输入 Ctrl+C，然后输入 “y” 以终止批处理作业。

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

在 Windows CE 设备上运行 JSP 应用程序

过程:

要在 Windows CE 设备上运行 JSP 应用程序:

1. 将应用程序传送至设备。
2. 如果需要，在设备上修改 `MiniHttpServer.lnk`。如果在使用 J9 JVM 之外的 JVM，则需要修改快捷方式。快捷方式开始的数字为“#”字符后的字符数。
3. 如果需要，为 Windows CE 设备配置 `MiniHttpConfig.properties` 文件。
4. 启动微型 HTTP Web 服务器:
   a. 打开“文件资源管理器”。
   b. 导航至根目录。
   c. 单击 MiniHttpServer 快捷方式。
5. 打开 Web 浏览器并输入 URL 来启动 JSP 应用程序。例如，如果应用程序的开始页面是 `start.jsp`，则输入 `http://localhost/start.jsp`。如果在 `MiniHttpConfig.properties` 文件中配置了 `Port` 属性，则输入 `http://localhost:Port/start.jsp` 代替，其中“Port”是指定的端口号。
6. 要在运行完 JSP 应用程序时停止微型 HTTP Web 服务器:
   a. 转至 J9 控制台窗口。
   b. 单击**文件 —> 关闭**。

相关任务:

- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

在 Symbian OS V6 设备上运行 JSP 应用程序

过程:

1. 将应用程序传送至设备。
2. 如果需要的话，为 Symbian 设备配置 `MiniHttpConfig.properties` 文件。
3. 启动微型 HTTP Web 服务器:
   a. 转至设备上的“附加”屏幕。
   b. 启动 DB2eJSP 应用程序。
4. 打开 Web 浏览器并输入 URL 来启动 JSP 应用程序。例如，如果应用程序的开始页面是 `start.jsp`，则输入 `http://localhost/start.jsp`。如果在 `MiniHttpConfig.properties` 文件中配置了 `Port` 属性，则输入 http://localhost:Port/start.jsp 代替，其中“Port”是指定的端口号。
5. 要在运行完 JSP 应用程序时停止微型 HTTP Web 服务器，执行软复位。

相关任务:

- 第 36 页的『在 Symbian OS V6 设备上安装和验证 JSP 支持』

相关参考:

- 『支持的 JSP 版本 1.1 子集』
- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』
- 第 31 页的『支持 JSP 的操作系统』

支持的 JSP 版本 1.1 子集

本节列示并描述 DB2 Everyplace JSP 支持中包括的伪指令、隐式对象、脚本编制元素和标准操作。

伪指令:

Page 伪指令:

描述 page 伪指令定义页面从属属性。

语法

```
<%@ page page_directive_attr_list %>
page_directive_attr_list ::=
{language="scriptingLanguage"}
{extends="className" }
{import="importList" }
{contentType="ctinfo" }
```

属性 此伪指令的四个有效属性是:

- **language** – 必须为 “java” (如果指定的话)。
- **extends** – 全限定 Java 编程语言类名, 用于命名将此 JSP 页变换至的类的超类。
- **import** – 缺省导入列表为 com.ibm.db2e.jsp.server.*、java.io.*、java.sql.* 和 java.util.*。
- **contentType** – 可以为任何值 (例如, text/html、text/xml 和 application/x-octet)。

示例

```
<%@ page contentType="text/html" %>
```

Include 伪指令:

描述 使用 include 伪指令来将数据包含在 JSP 页。包含的文件可能具有也将处理的元素。

语法

```
<%@ include file="relativeURLspec" %>
```

属性 此伪指令的属性是 **file**, 它必须是页面相对路径。该路径不能以 “/” 开始, 且将解释为当前 JSP 页的相对路径。

示例

```
<%@ include file="copyright.html" %>
```

隐式对象:

当创建 JSP 页时，您具有对某些隐式对象的访问权。可以在 scriptlet 和表达式中使用这些对象，而不必首先声明它们。每个隐式对象具有在核心 Java 技术或 com.ibm.db2e.jsp.server 包中定义的类，如表 4 中所示。

表 4. 隐式对象

隐式变量	类型	表示	方法摘要
request	com.ibm.db2e.jsp.server.MiniHttpRequest	对 JSP 页的请求。	java.lang.String **getParameter**(java.lang.String name) java.lang.String **getQueryString**()
response	com.ibm.db2e.jsp.server.MiniHttpResponse	对请求的响应	java.lang.String **encodeURL**(java.lang.String url) void **setDateHeader**(java.lang.String name, long date) void **setHeader**(java.lang.String name, java.lang.String value)
in	java.io.BufferedReader	此对象当前不可用。	
out	java.io.PrintStream	写入到 Web 浏览器的对象。	
config	com.ibm.db2e.jsp.server.DB2eJspConfig	用于此 JSP 页的 DB2eJspConfig。	java.lang.String **getInitParameter**(java.lang.String name)
exception	java.lang.Throwable	执行 JSP 页期间抛出的异常。	

注: 由于 DB2 Everyplace JSP 支持的实现不同，以上隐式对象的某些类型与 JSP 1.1 中的不同。

脚本编制元素:

声明:

描述 使用“声明”来声明在 JSP 页中使用的 Java 变量和方法。声明是 JSP 页的 Java 类的成员变量（字段和方法）。

语法

```
<%!declaration(s) %>
```

示例

```
<%!
String name = "Joe Smith";

public String getName() {
return name;
}
%>
```

标准操作:

<jsp:useBean>:

描述 jsp:useBean 操作通过新声明的同一标识的脚本编制变量使在可用“page”作用域中定义的 Java 编程语言对象的实例与给定标识相关联。

语法

```
<jsp:useBean id="name" scope="page|request|session|application" typeSpec/>
typeSpec ::=
class="className"
```

属性 此标记的三个属性是:

- **id** – 指定此 bean 的标识。不要在 JSP 页中重用此名称。此属性是必需的
- **scope** – 此属性将被忽略（如果指定的话）。将使用缺省作用域“page”。
- **class** – 指定此 bean 表示的类。此属性是必需的。

示例　<jsp:useBean id=″masterViewDBBean″ class=″Query1HTMLResultsMasterViewBean″ />

<jsp:setProperty>:

描述　jsp:setProperty 操作设置 Bean 中的属性的值。

语法

```
<jsp:setProperty name="beanName" prop_expr />
prop_expr ::=
property="propertyName" value="propertyValue"
propertyValue ::= string
值 propertyValue 还可为请求时属性值。
propertyValue ::= expr_scriptlet
```

属性　此标记的三个属性是:

- **name** – Bean 实例的名称由 <jsp:useBean> 元素在此操作出现之前定义。Bean 实例必须包含您想要设置的属性。此属性是必需的。
- **property** – 您想要为其设置值的 Bean 属性的名称。此属性是必需的。
- **value** – 要指定给指定属性的值。此属性可接受请求时属性表达式作为值。此属性是必需的。

示例

```
<jsp:setProperty name="masterViewDBBean" property="username"
value='<%=config.getInitParameter("username")%>' />
```

Scriptlet:

描述　使用 scriptlet 来挂起任何有效的 Java 代码片段。这些代码片段放置在 JSP 页的源代码中且与 JSP 页的其它元素相关。

语法

```
<% scriptlet %>
```

示例

```
<%
          try {
String name = Query1DBBean.getString(1);
out.println("Name = " + name);
}
catch (SQLException e) {

}
%>
```

表达式:

描述　表达式是数据类型的字符串表示。可以在查询和 HTML 注释中使用表达式。应用程序在运行时对表达式求值并将表达式转换为字符串。

语法

```
<%= expression %>
```

示例

```
<%= new java.util.Date() %>
```

注释: 以 *__db2ejsp__* 开始的变量名是关键字，在内部使用。不要在 JSP 页中使用这些变量。

相关概念:

- 第 32 页的『DB2 Everyplace JSP 支持概述』

- 第 31 页的『开发 DB2 Everyplace JSP 应用程序』
- 第 107 页的第 16 章, 『样本 JSP 应用程序』

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关参考:

- 『用于 JSP 应用程序数据库存取的 IBM 定制标记』

用于 JSP 应用程序数据库存取的 IBM 定制标记

以下标记是 IBM 定制标记, 可以在 JSP 应用程序中使用它们来存取 DB2 Everyplace 数据库。

<tsx:dbconnect>:

描述 此标记使用 DB2 Everyplace JDBC 驱动程序建立与特定 DB2 Everyplace 数据库的连接。

语法

```
<tsx:dbconnect
id="connection_id"
driver="com.ibm.db2e.jdbc.DB2eDriver"
url="jdbc:db2e:database"
userid="db_user"
passwd="user_password">
</tsx:dbconnect>
```

属性 此标记的五个属性包括:

- **id** – 指定此连接的标识。不要在 JSP 页中重用此名称。此属性是必需的。
- **driver** – 指定 DB2 Everyplace JDBC 驱动程序。此属性是必需的。
- **url** – 指定 DB2 Everyplace 数据库。术语*数据库*指的是 DB2 Everyplace 数据库的路径。此属性是必需的。
- **userid** – 指定对要存取的数据库有效的用户标识。此属性是可选的。
- **passwd** – 指定 userid 属性的用户密码。如果指定了 userid, 则此属性是必需的。

示例

```
<tsx:dbconnect
id="conn"
driver="com.ibm.db2e.jdbc.DB2eDriver"
url="jdbc:db2e:sample/data/" >
</tsx:dbconnect>
```

<tsx:dbquery>:

描述 此标记使用通过 <tsx:dbconnect> 标志指定的连接将查询提交给数据库, 并生成一个 java.sql.ResultSet 对象, 在该对象中游标指向结果集的第一行。可以使用此查询的标识和 java.sql.ResultSet 的 DB2 Everyplace JDBC 接口引用此结果集。

语法

```
<tsx:dbquery id="query_id" connection="connection_id" limit="value">
select_SQL_statement
</tsx:dbquery>
```

属性 此标记的属性是:

- **id** – 指定此查询的标识。不要在 JSP 页中重新使用此查询标识。此属性是必需的。
- **connection** – 指定此 JSP 文件中的 <tsx:dbconnect> 标记的标识。此属性是必需的。
- **limit** – 指定查询可以返回的最大行数。此属性是可选的。

参数 此标记的有效参数是:

- **select_SQL_statement** – 指定要提交给数据库的 SQL 查询。此 SQL 查询语句可以包含动态数据。

示例

```
<tsx:dbquery id="Query1DBBean" connection="conn">
select <%= request.getParameter("column") %> from vnperson
</tsx:dbquery>
```

<tsx:dbmodify>:

描述 此标记使用通过 <tsx:dbconnect> 标记指定的连接来提交命令，以修改数据库内的数据。没有此标记的结果。

语法

```
<tsx:dbmodify connection="connection_id">
modify_command
</tsx:dbmodify>
```

属性 此标记的属性是:

- **connection** – 指定此 JSP 文件中的 <tsx:dbconnect> 标记的标识。此属性是必需的。

参数 此标记的有效参数是:

- **modify_command** – 指定要提交到数据库以修改数据的 SQL 命令。此修改命令可以包含动态数据。

示例

```
<tsx:dbmodify connection="conn">
update vnperson set Name = '<%=Name%>' where ID = '<%=id%>'
</tsx:dbmodify>
```

<tsx:repeat>:

描述 使用此标记来循环查询结果中的每一行。启动和停止属性控制循环过程。如果未指定启动和停止属性，当结果集的游标（由 <tsx:getProperty> 标记引用）到达结果集的末尾时循环终止。可以嵌套此标记。

语法

```
<tsx:repeat index="name" start="starting_index" stop="ending_index">
repeat_block
</tsx:repeat>
```

属性 此标记的属性是:

- **index** – 指定此标记的索引的标识。此属性是可选的。
- **start** – 指定处理重复块之前要跳过的行数。缺省值为 0，此属性是可选的。
- **stop** – 指定此重复块的结束索引值。缺省值是 2,147,483,647。此属性是可选的。

参数 此标记的有效参数是:

- **repeat_block** – 指定包含 <tsx:getProperty> 标记语法的 HTML 标记块和用于格式化内容的 HTML 标记。如果将 <tsx:getProperty> 标记放置在重复块中，游标在每次处理重复块时都会进到下一行。

示例

```
<TABLE border="1">
<TR>
<TH>Name</TH>
</TR>

<tsx:repeat>
<TR>
<TD>
<tsx:getProperty name="Query1DBBean" property="Name" />
</TD>
</TR>
</tsx:repeat>
</TABLE>
```

<tsx:getProperty>:

描述 此标记检索要在 JSP 页（即 HTML 结果页面）中显示的 ResultSet bean 的值。如果将此标记放置在 <tsx:repeat> 块标记内，ResultSet bean 的游标在每次处理重复块时会进到下一行。

语法

```
<tsx:getProperty name="bean_name" property="property_name" />
```

属性 此标记的属性是:

- **name** – 指定 JSP 文件中先前使用 <tsx:dbquery> 标记声明的 ResultSet bean 的名称。
- **property** – 指定要访问的 ResultSet bean 的列。

示例

```
<tsx:getProperty name="Query1DBBean" property="FIRSTNAME" />
```

相关概念:

- 第 32 页的『DB2 Everyplace JSP 支持概述』
- 第 31 页的『开发 DB2 Everyplace JSP 应用程序』
- 第 107 页的第 16 章, 『样本 JSP 应用程序』

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关参考:

- 第 41 页的『支持的 JSP 版本 1.1 子集』

JSP 应用程序故障诊断

故障诊断:

微型 HTTP Web 服务器控制台输出:

1. 与 WebSphere Studio Application Developer 有关
   - java.lang.NoClassDefFoundError: javax/sql/DataSource

WSAD V4.0.3+ 包含此错误的修正。可在 <DB2Everyplace>\SDK\JSP\sample\jsp\VNSchedule_wsad40 中使用 dbbeans.jar，它是 WSAD V4.0.3 中的 JAR。

- “不能确定列类型名称 <name> 的搜索值。假定 searchable = true。”

  此消息是 dbbeans.jar 生成的，可忽略。

Web 浏览器:

- “找不到您正在查找的页面。”

  如果 Pocket Internet Explorer 在您尝试连接至 URL “http://localhost/” 时显示此消息，表明 Windows CE 设备可能比 Pocket PC V3.0.11171 要旧且您需要获取 Web 浏览器的修订。（在因特网上，搜索有关如何通过 Pocket Internet Explorer 连接至本地主机的信息。）确保微型 HTTP Web 服务器正在运行。

应用程序开发:

- JSP 页的更改未反映在 Web 浏览器中

  如果发生这种情况，可能需要删除 Web 浏览器的 Temporary Internet Files 文件夹或高速缓存中的文件。

相关概念:

- 第 32 页的『DB2 Everyplace JSP 支持概述』
- 第 31 页的『开发 DB2 Everyplace JSP 应用程序』
- 第 107 页的第 16 章，『样本 JSP 应用程序』

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关参考:

- 第 44 页的『用于 JSP 应用程序数据库存取的 IBM 定制标记』

第 7 章 开发 .NET 应用程序

本章提供有关使用 .NET 开发 DB2 Everyplace 应用程序的信息。涉及的主题包括:

- 『同步支持』
- 第 52 页的『对构建 .NET 应用程序的支持』

同步支持

本节提供有关 DB2 Everyplace .NET 同步支持的信息。涉及的主题包括:

- 『ISync.Net API 文件位置』
- 第 50 页的『使用 ISync.NET API』
- 第 51 页的『使用 ISyncComponent』
- 第 51 页的『使用 ISync.NET API 的简单示例应用程序』

ISync.Net API 文件位置

DB2 Everyplace Sync Client 提供两个 API, 它们使开发者能够构建用于 DB2 Everyplace Sync Server 的受管应用程序。这些 API 包括:

- ISyncComponent
- ISync.Net

ISyncComponent 小于 ISync.Net, 但它为想要使用此功能的开发者提供可视设计支持。

有关数据库引擎的 .Net 提供程序的信息, 请参阅第 52 页的『用于在客户机数据库上构建应用程序的 .NET 支持的概述』。

表 5. ISync.NET 受管提供程序位置和名称空间

可用提供程序	名称空间	受支持平台	位置, **\DB2Everyplace\Clients**
.NET Framework 的非 Unicode	IBM.Data.Sync IBM.Data.Sync.DB2e	Win 32	Win32\Sync\NMP\IBM.Data.Sync.DB2e.dll
.NET Framework 的 Unicode	I B M . D a t a . S y n c IBM.Data.Sync.DB2e	Win32 Unicode	Win32\Sync\NMP\unicode\IBM.Data.Sync.DB2e.dll
.NET Compact Framework	I B M . D a t a . S y n c IBM.Data.Sync.DB2e.CF	WinCE	WinCE\Sync\NMP\IBM.Data.Sync.DB2e.CF.dll

样本应用程序 ISync.NET

\DB2Everyplace\Clients\clientapisample\NMP*lang* 中有两个样本应用程序:

- GoISync
- sample1

其中 *lang* 是 C#（cs）或 Visual Basic（vb）

ISync.NET API 规范

可在 \DB2Everyplace\Clients\clientapisample\NMP\ISync.NET.chm 中找到 ISync.NET 的 API 规范

相关概念:

- 第 51 页的『使用 ISync.NET API 的简单示例应用程序』
- 第 52 页的『用于在客户机数据库上构建应用程序的 .NET 支持的概述』
- 第 57 页的『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

相关任务:

- 『使用 ISync.NET API』
- 第 51 页的『使用 ISyncComponent』
- 第 53 页的『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』

使用 ISync.NET API

可在 \DB2Everyplace\Clients\clientapisample\NMP\ISync.NET.chm 中找到 ISync.NET 的 API 规范

先决条件:

软件需求

- DB2 Everyplace V8.1.4 beta 1
- Microsoft. NET Standard Framework 1.0（随 Visual Studio 2002 包括）- 这是在 Win32 上开发应用程序所需的
- Microsoft .NET Compact Framework（随 Visual Studio 2003 包括）- 这是在 WinCE 上开发应用程序所需的

尽管 ISync.NET 提供程序与平台和语言无关，但它仍然依赖于底层的本机 Sync Client 库。在应用程序运行时，提供程序和 Sync Client 库必须包括在用户路径中。在安装 DB2 Everyplace 期间，应更新用户路径。

有关客户机库的信息，请参阅*《DB2 Everyplace 安装与用户指南》*中的第三章『在移动式设备上安装 DB2 Everyplace』。

过程:

要在应用程序中使用 Isync.NET，必须对所有 Visual Studio Framework 和应用程序类型执行下列步骤。

1. 在 Microsoft Visual Studio .NET 中，以您选择的语言创建新项目。
2. 在应用程序中，导入 DB2 Everyplace 名称空间。下面是 Standard Framework 的示例:

   ```
   [Visual Basic]
        Imports IBM.Data.Sync
        Imports IBM.Data.Sync.DB2e
        [C#]
        using IBM.Data.Sync;
        using IBM.Data.Sync.DB2e;
   ```

   有关更多信息，可查看 \DB2Everyplace\Clients\clientapisample\NMP 中的样本同步应用程序
3. 添加引用:

   a. 在 Visual Studio 中，右键单击项目名并选择**添加引用**。

b. 在“项目”选项卡下，进行浏览以查找 **IBM.Data.Sync.DB2e.dll** 的位置。

c. 在命令行上，输入：**csc /t:exe /r:IBM.Data.Sync.DB2e.dll ISyncSample.cs**。

相关概念:

- 第 49 页的『ISync.Net API 文件位置』
- 『使用 ISync.NET API 的简单示例应用程序』
- 第 52 页的『用于在客户机数据库上构建应用程序的 .NET 支持的概述』
- 第 57 页的『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

相关任务:

- 『使用 ISyncComponent』
- 第 53 页的『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』

使用 ISyncComponent

过程:

ISyncComponent 还在 Standard Framework 中提供了最小设计支持。这一基本支持使您能够拖放至表单以及修改 ConnectionString（服务器、端口和用户名）和 TargetPath（数据的目标目录）属性。在开发 “Visual Studio Windows 应用程序” 时，一定要将 DB2 Everyplace 组件 **IBM.Data.Sync.DB2e.dll** 添加至**工具箱**。

注: 要使此过程成功完成，本机 Sync Client 库必须已在用户路径中。
对于 Standard Framework，还有一个选择，可通过使用 IBM.Data.Sync.DB2e.ISyncComponent 来使用更简单的 API。

```
ISyncComponent comp1 = new ISyncComponent();
comp1.ConnectionString = SERVER=localhost;PORT=80;UID=username;PWD=password;
comp1.TargetPath = data;
comp1.Sync();
comp1.Close();
```

相关概念:

- 第 49 页的『ISync.Net API 文件位置』
- 『使用 ISync.NET API 的简单示例应用程序』
- 第 52 页的『用于在客户机数据库上构建应用程序的 .NET 支持的概述』
- 第 57 页的『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

相关任务:

- 第 50 页的『使用 ISync.NET API』
- 第 53 页的『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』

使用 ISync.NET API 的简单示例应用程序

本主题包括提供如何使用 ISync.NET API 的快速参考的示例。

```
// Synchronization properties
private Hashtable userProps = new Hashtable();

// Get an instance DB2eISyncProvider
```

```
ISyncProvider provider = DB2eISyncProvider.GetInstance();

// Set up properties
userProps.Add("isync.user", username);
userProps.Add("isync.password", password);

// Get an instance of synchronization service from the provider
ISyncService service = provider.CreateSyncService(http://localhost:80, userProps);

// Get an instance of the configuration store
ISyncConfigStore config = service.GetConfigStore(data);

// Get an instance of the sync driver to perform synchronization
ISyncDriver syncer = config.GetSyncDriver();

// Perform synchronization
syncer.Sync();

// Close objects
syncer.Close();
config.Close();
service.Close();
```

相关概念:

- 第 49 页的『ISync.Net API 文件位置』
- 『用于在客户机数据库上构建应用程序的 .NET 支持的概述』
- 第 57 页的『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

相关任务:

- 第 50 页的『使用 ISync.NET API』
- 第 51 页的『使用 ISyncComponent』
- 第 53 页的『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』

对构建 .NET 应用程序的支持

本节提供有关对构建 .NET 应用程序的支持的信息。涉及的主题包括:

- 『用于在客户机数据库上构建应用程序的 .NET 支持的概述』
- 第 53 页的『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』
- 第 57 页的『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

用于在客户机数据库上构建应用程序的 .NET 支持的概述

DB2 Everyplace 提供了一些工具，它们使开发者能够构建这样的应用程序: 这些应用程序使用 ADO.NET API 来处理由 DB2 Everyplace 数据库管理的数据。DB2 Everyplace 包含两个 .NET 数据提供程序。一个提供程序在 .NET Framework 1.0 上运行，而另一个提供程序在 .NET Compact Framework 上运行。您可在以下位置找到这些提供程序或 API:

- **对于 Win32**:

  ```
  \DB2Everyplace\Clients\Win32\database\nmp\IBM.Data.DB2.DB2e.dll
  ```

- **对于 WinCE**:

  ```
  \DB2Everyplace\Clients\WinCE\database\nmp\IBM.Data.DB2.DB2e.CF.dll
  ```

API 规范位于:

\DB2Everyplace\Clients\Win32\database\nmp\doc\readme.html

有关 Sync Client 的 .NET 提供程序的信息，请参阅《*DB2 Everyplace Sync Server 管理指南*》中『在 Windows 源系统上配置源数据库』一节。

为简化过去使用 Microsoft ODBC .NET Data Provider 的程序员的转换过程，新的 DB2 Everyplace .NET Data Provider 接口几乎与 Microsoft ODBC .NET Data Provider 的那些接口完全相同。例如，Microsoft ODBC .NET Data Provider 具有 OdbcConnection 类，而 IBM DB2 Everyplace .NET Data Provider 具有等效的功能类: DB2eConnection。同样，可在其它类名中将 “Odbc” 替换为 “DB2e” 以获取相应的 DB2 Everyplace .NET Data Provider 类。

相关概念:

- 第 49 页的『ISync.Net API 文件位置』
- 第 51 页的『使用 ISync.NET API 的简单示例应用程序』
- 第 57 页的『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

相关任务:

- 第 51 页的『使用 ISyncComponent』
- 『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』

使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述

DB2 Everyplace .NET Data Provider 的名称空间如下所示:

- **在 .NET Compact Framework 上运行**: IBM.Data.DB2.DB2e.CF
- **在 .NET Framework 上运行**: IBM.Data.DB2.DB2e

DB2 Everyplace .NET Data Provider 提供用于连接至 DB2 Everyplace 数据源、执行命令和检索结果的功能。这些结果可直接处理，或放在 ADO.NET DataSet 中以便在处于断开连接状态时作进一步地处理。当在 DataSet 中时，数据可被用户见到（连同多个源中的其它数据）或在层与层之间远程传递。当在 DataSet 中时，对数据执行的任何处理都将与数据源协调。

DB2 Everyplace .NET Data Provider 被设计为轻量级的。它由 DB2 Everyplace 与扩展功能但未牺牲性能的代码之间的最小层组成。

DB2 Everyplace .NET Data Provider 类从其它 .NET Framework 类或接口继承或实现成员。此提供程序文档包括每个类中的受支持成员的摘要。有关特定继承成员的更详细信息，请参阅 Microsoft® .NET Framework SDK 中的相应主题。

先决条件:

表 6. 使用 DB2 Everyplace .NET Data Provider 的先决条件

组件	最小需求
Microsoft.NET Framework	Microsoft.NET Framework 1.0 必须安装它才能安装用于应用程序开发的 DB2 Everyplace .NET Data Provider
Microsoft Visual Studio.NET 2003	用于开发移动式应用程序的 Microsoft Visual Studio.NET 2003
Microsoft.NET Compact Framework	用于移动式开发的 Microsoft .NET Compact Framework 1.0 必须将它安装在设备上才能安装用于移动式应用程序开发的 DB2 Everyplace .NET Data Provider。
DB2 Everyplace 产品	• 版本 8.1.4 或以上版本的 `DB2e.dll` • 版本 8.1.4 或以上版本的 `AgentProxy.dll` 是远程存储过程调用所必需的 • 版本 8.1.4 或以上版本的 `wbxmllib.dll`，是远程存储过程调用所必需的。 • DB2 Everyplace Sync Server V8.1.4 或以上版本，是远程存储过程调用所必需的 `DB2e.dll`、`AgentProxy.dll` 和 `wbxmllib.dll` 都是本机库（从而依赖于处理器）；因此，操作系统需要定位这些本机库（例如，设置环境变量 PATH）以使 DB2 Everyplace .NET Data Provider 正常工作。

限制:

提供程序限制:

- 对主键列的更新在 DB2 Everyplace 中目前是不允许的。
- 使用远程存储过程调用的结果集检索对结果集的大小有限制。
- 不支持本地存储过程调用。
- 对于不受支持的方法或属性，将抛出 `System.NotSupportedException`

线程安全:

此提供程序的任何公用非实例成员对于多线程操作都是安全的。不能保证所有实例成员都是线程安全的。

过程:

DB2 Everyplace .NET Data Provider 由四个核心对象组成。下表描述这些对象及其功能。

表 7. DB2 Everyplace .NET Data Provider，核心对象

对象	描述
DB2eConnection	建立与 DB2 Everyplace 数据源的连接且可以开始一个事务。
DB2eCommand	在 DB2 Everyplace 服务器上执行命令并显示参数。

表 7. DB2 Everyplace .NET Data Provider，核心对象 (续)

对象	描述
DB2eDataAdapter	填充 *DataSet* 并使用 DB2 Everyplace 数据源解析更新。
DB2eDataReader	显示和读取来自 DB2 Everyplace 数据源的仅正向数据流。

除了先前表中列示的核心类，DB2 .NET Data Provider 还包含下表中列示的类。

表 8. DB2 Everyplace .NET Data Provider，附加类

对象	描述
DB2eCommandBuilder	一个帮助程序对象，将自动生成 *DB2eDataAdapter* 的命令属性或从存储过程派生参数信息并填充 *DB2eCommand* 对象的 *DB2eParameters* 集合。注意: 不建议使用 *DB2eCommandBuilder*，原因是它可能生成效率非常低甚至无效（在某些情况下）的 SQL 语句。
DB2eError	显示 DB2 Everyplace 数据源返回的警告或错误中的信息。
DB2eException	在 DB2 Everyplace 数据源上遇到错误时返回。对于在客户机上遇到的错误，.NET 数据提供程序将抛出 .NET Framework 异常。
DB2eParameter	定义命令和存储过程的输入、输出和返回值参数。
DB2eTransaction	使您能够让命令加入 DB2 Everyplace 数据源中的事务。

要使用 DB2 Everyplace .NET Data Provider，必须将 IBM.Data.DB2.DB2e 或名称空间的 imports 或 using 语句添加至应用程序 .DLL，如以下代码所示:

[Visual Basic]

```
Imports IBM.Data.DB2.DB2e
```

[C#]

```
using IBM.Data.DB2.DB2e;
```

在编译代码时，还必须包括对 .DLL 的引用。例如，如果在编译 Microsoft® Visual C#™ 程序，命令行应包括:

```
csc /r:IBM.Data.DB2.DB2e.dll
```

对于 .NET Compact Framework，名称空间为 IBM.Data.DB2.DB2e.CF，而应用程序需要引用 IBM.Data.DB2.DB2e.CF.dll 汇编。

有关如何使此名称空间得到最好利用的信息，请参阅有关下列 DB2 Everyplace.NET Data Provider 类的文档:

- DB2eDataAdapter
- DB2eCommand
- DB2eConnection
- DB2eDataReader

有关 DB2 Everyplace .NET Data Provider 在 .NET Framework 内如何工作的更多信息，请参阅 IBM.Data.DB2.DB2e Hierarchy。

表 9. 类

对象	描述
DB2eCommand	表示要对数据源执行的 SQL 语句或存储过程。不能继承此类。
DB2eCommandBuilder	自动生成用于将对 *DataSet* 的更改与相关联的数据源协调的单个表命令。不能继承此类。
DB2eConnection	表示与数据源的打开连接。
DB2eDataAdapter	表示一组数据命令和与数据源的连接，它们用来填充 *DataSet* 并更新数据源。不能继承此类。
DB2eDataReader	提供一种方式读取来自数据源的仅正向数据行流。不能继承此类。
DB2eError	收集与数据源返回的警告或错误有关的信息。不能继承此类。
DB2eException	当 DB2 Everyplace 数据源返回警告或错误时生成的异常。不能继承此类。
DB2eParameter	表示 *DB2eCommand* 的参数及其对 *DataColumn* 的映射（可选）。不能继承此类。
DB2eTransaction	表示要在数据源上进行的 SQL 事务。不能继承此类。

表 10. 代表

代表	描述
DB2eInfoMessageEventHandler	表示将用来处理 DB2eConnection 的 InfoMessage 事件的方法。
DB2eRowUpdatedEventHandler	表示将用来处理 DB2eDataAdapter 的 RowUpdated 事件的方法。
DB2eRowUpdatingEventHandler	表示将用来处理 DB2eDataAdapter 的 RowUpdating 事件的方法。

表 11. 枚举

枚举	描述
DB2eType	指定字段、属性或 DB2eParameter 的数据类型。

表 12. DB2 Everyplace .NET Provider 连接字符串关键字

关键字	描述
Database	数据库位置。例如: `C:\data1\`
UID	用户标识
PWD	密码

C# 示例:

```
string connString = @"Database=C:\data1\; UID=user; PWD=userpwd";
```

相关概念:

- 第 49 页的『ISync.Net API 文件位置』
- 第 51 页的『使用 ISync.NET API 的简单示例应用程序』

- 第 52 页的『用于在客户机数据库上构建应用程序的 .NET 支持的概述』
- 『WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码』

相关任务:

- 第 50 页的『使用 ISync.NET API』
- 第 51 页的『使用 ISyncComponent』

WinCE 和 Win32 的样本 DB2 Everyplace .NET Data Provider 应用程序代码

有两个样本应用程序演示如何使用 DB2 Everyplace .NET Data Provider 为 WinCE 和 Win32 开发应用程序:

- DB2eSample1.cs
- DB2eSample2.sc

两个文件都在以下位置:

- **对于 Win32 和 WinCE**

  `\DB2Everyplace\Clients\Win32\database\nmp\samples`

下面是样本应用程序之一的示例。

```
using System;
using System.Text;
using System.Data;

using IBM.Data.DB2.DB2e;

/*
 * Sample1
 *
 * The following example creates a table, insert some rows to it, fetches
 * all the rows from the table, and finally drops the table.
 *
 */
namespace IBM.Data.DB2.DB2e.Samples
{
        class DB2eSample1
        {
                /// <summary>



     /// The main entry point for the application.
     /// </summary>
     [STAThread]
     static void Main(string[] args)
       {
         DB2eConnection conn   = null;
         DB2eCommand cmd       = null;
         DB2eDataReader reader = null;
          String connString     = @"database=.\; uid=user1; pwd=user1";
          int rowsAffected      = 0;

           try
                {
```

```
         conn = new DB2eConnection(connString);
         conn.Open();

           Console.WriteLine("creating table t1...");
           cmd = new DB2eCommand("create table t1 (c1 int primary key not null,
c2 smallint, c3 char(10), c4 varchar(10), c5 decimal(8,2), c6 date,
c7 time, c8 timestamp )", conn);
              rowsAffected = cmd.ExecuteNonQuery();
              Console.WriteLine("inserting a row into table t1...");
              cmd.CommandText = "insert into t1 values (1, 10, 'John',
 'Yip', null, current date, current time, current timestamp)";
          rowsAffected = cmd.ExecuteNonQuery();
          Console.WriteLine("inserting a row into table t1...");
          cmd.CommandText = "insert into t1 values (2, 20, 'Mary', 'Jann',
 2.2, current date, current time, current timestamp)";
          rowsAffected = cmd.ExecuteNonQuery();
          cmd.CommandText = "select * from t1";
          Console.WriteLine("fetching resultset from table t1...");
          reader = cmd.ExecuteReader();
              while (reader.Read())
                {
                  if (!reader.IsDBNull(0))
                     Console.Write(reader.GetInt32(0) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(1))
                      Console.Write(reader.GetInt16(1) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(2))
                      Console.Write(reader.GetString(2) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(3))
                      Console.Write(reader.GetString(3) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(4))
                      Console.Write(reader.GetDecimal(4) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(5))
                      Console.Write(reader.GetDate(5) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(6))
                      Console.Write(reader.GetTime(6) + "\t");
        else
                      Console.Write("NULL " + "\t");
                  if (!reader.IsDBNull(7))
                      Console.Write(reader.GetDateTime(7) + "\t");
        else
                      Console.Write("NULL " + "\t");
                      Console.WriteLine();
                  }
              reader.Close();
              reader = null;
              Console.WriteLine("dropping table t1...");
              cmd.CommandText = "drop table t1";
              cmd.ExecuteNonQuery();
```

```
                    }
                      catch (DB2eException e1)
                     {
                          int cnt = e1.Errors.Count;
                          for (int i=0; i < cnt; i++)
                            {
                              Console.WriteLine("Error #" + i + "\n" +
                                 "Message: " + e1.Errors[i].Message + "\n" +
                                 "Native: " + e1.Errors[i].NativeError.ToString() + "\n" +
                                 "SQL: " + e1.Errors[i].SQLState + "\n");
                            }
                         }
        catch (Exception ex)
                         {
                              Console.WriteLine(ex.Message);
                         }
     finally
                          {
                              if (reader != null)
                                 {
                                    reader.Close();
                                    reader = null;
                                 }
                              if (conn != null)
                                 {
                                    conn.Close();
                                    conn = null;
                                 }
                          }
                }  // end of Main

        }  // end of class

} // end of namespace
```

相关概念:

- 第 49 页的『ISync.Net API 文件位置』
- 第 51 页的『使用 ISync.NET API 的简单示例应用程序』
- 第 52 页的『用于在客户机数据库上构建应用程序的 .NET 支持的概述』

相关任务:

- 第 50 页的『使用 ISync.NET API』
- 第 51 页的『使用 ISyncComponent』
- 第 53 页的『使用 DB2 Everyplace .NET Data Provider 开发 ADO.NET 应用程序的概述』

第 8 章 连接至 DB2 Everyplace 数据库

本章说明如何连接至 DB2 Everyplace 数据库。涉及下列主题:

- 『DB2 Everyplace 数据库表的概述』
- 『处理命名冲突』
- 第 62 页的『连接至 DB2 Everyplace 数据库』
- 第 63 页的『连接串行化』
- 第 63 页的『只读介质上的 DB2 Everyplace 数据库』

DB2 Everyplace 数据库表的概述

DB2 Everyplace 数据库由几个系统目录表和一些用户定义的表组成。每个表存储在两个文件中，一个文件用于数据本身，另一个文件用于索引。将在同一索引文件中保存所有索引。与 “DB2 通用数据库” 不同，DB2 Everyplace 数据库没有名称且不能编目或取消编目。即，忽略数据库名。在 DB2 Everyplace 中，数据库只是可以复制或移动到另一个位置的一些文件。系统目录表包含有关用户定义的表的元数据。例如，如果除去用户定义的表的文件而没有删除目录表中的相应条目，则此操作将导致不一致性。DB2 Everyplace 数据库必须包含下列系统目录表:

- DB2eSYSTABLES
- DB2eSYSCOLUMNS
- DB2eSYSRELS
- DB2eSYSUSERS（如果使用本地数据加密，则创建此表）

要在查询中访问目录表，必须使用定界标识。例如，以下查询返回是否存在表 T:

```
SELECT 1 FROM "DB2eSYSTABLES" WHERE TNAME = 'T'
```

相关参考:

- 第 313 页的第 19 章，『DB2 Everyplace 系统目录基本表』

处理命名冲突

过程:

本主题显示某些方法示例，您可以使用这些方法处理用户定义的表的文件命名冲突。假设应用程序执行以下 CREATE TABLE 语句:

```
CREATE TABLE T (PK INT NOT NULL PRIMARY KEY, A INT)
```

一旦执行此语句，DB2 Everyplace 将为表 *T* 创建以下两个文件:

- DSY_T（数据）
- DSY_iT（索引）

如果创建另一个表并使用名称 *iT*，DB2 Everyplace 将创建两个附加文件: DSY_iT（数据）和 DSY_iiT（索引）。因为表 *T* 的索引文件和表 *iT* 的数据文件具有相同的名称，

所以它们之间有冲突。两个文件都命名为 DSY_iT。为了避免此问题，DB2 Everyplace 支持文件名映射。即，文件名将完全由 DB2 Everyplace 创建和管理。要使用此功能部件，应用程序必须设置连接属性且必须在创建第一个表之前执行它。例如，在 CLI 中:

```
SQLSetConnectAttr(hdbc, SQL_ATTR_FILENAME_FORMAT,
                  (SQLPOINTER)SQL_FILENAME_FORMAT_83, 0)
```

或在“命令行处理器”中:

```
DISABLE LONG FILENAME
```

一旦执行此命令并创建了第一个表，产生的文件将是针对表 *T* 的:

- 0001.DBd
- 0001.DBi

相关概念:

- 第 61 页的『DB2 Everyplace 数据库表的概述』
- 第 63 页的『连接串行化』

相关任务:

- 『连接至 DB2 Everyplace 数据库』

连接至 DB2 Everyplace 数据库

应用程序经常需要在特定位置（如 Win32 平台上的 C:\TEMP 文件夹）中创建和访问表。可以使用以下 CLI 调用，通过指定数据库路径来连接到数据库:

```
rc = SQLConnect(hdbc, "C:\\TEMP\\db", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

其中 *db* 是数据库名称（DB2 Everyplace 忽略该名称）。

另外，可以使用此 CLI 调用:

```
rc = SQLConnect(hdbc, "C:\\TEMP\\", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

连接到设备或 WinCE 对象存储库上的扩展内存需要特殊路径规范。

- 对于 PalmOS 上的 Sony Memory Stick

  ```
  rc = SQLConnect(hdbc, "#0:\\", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
  ```

- 对于 WinCE 对象存储库

  ```
  rc = SQLConnect(hdbc, "@:\\", SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
  ```

使用“命令行处理器”，用户可以使用“CONNECT TO”命令连接到特定的位置。例如，下列命令将连接到保存在 Win32 平台上的 C:\TEMP\ 文件夹中的数据库。

```
CONNECT TO C:\TEMP\
```

相关概念:

- 第 61 页的『DB2 Everyplace 数据库表的概述』

相关任务:

- 第 61 页的『处理命名冲突』

连接串行化

一次仅能有一个与 DB2 Everyplace 数据库的连接。在除 Palm 之外的所有平台上，DB2 Everyplace 支持连接串行化。使用连接串行化，可以将数据源的连接请求串行起来。每次只可以建立一个与给定数据源的活动连接。将其它连接请求放入到队列中。可以使用 SQLSetConnectAttr() 函数 SQL_ATTR_LOGIN_TIMEOUT 属性设置超时时间。典型的串行化方案包括:

- 串行化尝试打开与单个数据源的连接的多个进程；每次仅可对一个进程打开一个给定的数据源。
- 串行化尝试打开与单个数据源的连接的单个进程内的多个线程。
- 多个不同进程每个都可以同时对多个不同数据源打开一个连接。

例如，以下 CLI /JDBC 调用将把连接超时时间设置为 10。即，如果存在与同一数据库的另一个连接，则应用程序将接收到错误。

对于 **CLI**:

```
 int i = 10; // 10 seconds timeout
rc = SQLSetConnectAttr(hdbc, SQL_ATTR_LOGIN_TIMEOUT, (SQLPOINTER) i, 0);
```

对于 **JDBC**:

```
int waitTime = 10;
String url = "jdbc:db2e:mysample";

Properties prop = new Properties();
prop.setProperty("LOGIN_TIMEOUT", Integer.toString(waitTime));

Connection con = driver.connect(url,prop);
```

注:

1. 缺省超时时间是 0 秒。
2. 多线程应用程序可以使用一个线程连接到数据库，而使用另一线程与数据库断开连接。
3. 当应用程序成功地连接到数据库时，将创建名为 “DSYLOCK” 的文件。如果应用程序进程异常终止，将自动除去文件 DSYLOCK。
4. 限制: 连接串行化可能对驻留在网络驱动器上的数据库不起作用。
5. 在 JDBC 程序中，如果以属性的形式将超时值传送到 DriverManager.getConnection() 方法，将忽略超时值并将它设置为零。

相关任务:

- 第 62 页的『连接至 DB2 Everyplace 数据库』

只读介质上的 DB2 Everyplace 数据库

可以从只读介质（如 CD-ROM 或嵌入式设备中的 ROM 芯片）直接运行 DB2 Everyplace 数据库和应用程序。例如，可以将每个季度的产品目录的样本应用程序通过 CD-ROM 分发给销售代表。每个季度销售代表会接收到一张 CD-ROM，它包含完整的公司产品目录和一个 DB2 Everyplace 应用程序，以便浏览、显示和查询满足客户特定需要的产品的产品信息。该 DB2 Everyplace 应用程序直接从 CD-ROM 运行，而不必首先在工作站上安装它。

当 DB2 Everyplace 检测到应用程序在只读介质上运行时（或检测到文件是写保护的），则它设置为只读方式。在此方式下，将禁用更新、插入、删除、创建和撤销语句并返

回错误。注意，对于某些选择查询，DB2 Everyplace 创建临时表和文件。在缺省临时目录中创建它们。在 Win32 平台上，通过环境变量 *TEMP* 指定此目录。如果 *TEMP* 环境变量不存在，也可以通过 *TMP* 指定它。在 Linux 中，使用 `/tmp/` 目录。

仅在 Win32 和 Linux 平台上支持此功能。

第 9 章 通过 CLI 逐块检索数据

用户有两种方法通过 CLI 从 DB2 Everyplace 表检索数据:

- 应用程序可选择分配列值可能占用的最大内存（根据 *SQLDescribeCol()* 确认的结果集中的列或先前的确认），然后通过 *SQLBindCol()* 绑定它。
- 应用程序可能调用 *SQLGetData()* 以将列数据装入到应用程序分配的缓冲区中。

如果是二进制数据（BLOB）或字符数据（CHAR 或 VARCHAR），列可能会很长。应用程序开发者可能不想分配的缓冲区太大以装入整列，或者可能无法分配这么大的缓冲区。此外，在某些情况下，应用程序只需要列的某些部分。在这些方案中，需要逐块检索数据。

过程:

SQLGetData() 功能允许应用程序使用重复的调用来依次获取单个列的值（以多个可管理数据块的形式）。实际上，调用 *SQLGetData()* 将返回 SQL_SUCCESS_WITH_INFO（带有 SQLSTATE 01004）以指示此列存在更多数据。重复调用 *SQLGetData()* 以获取余下数据块，直到它返回 SQL_SUCCESS，指示已检索此列的全部数据。

语法:

```
SQLRETURN   SQLGetData       (
                SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLSMALLINT       TargetType,        /* fCType */
                SQLPOINTER        TargetValuePtr,    /* rgbValue */
                SQLINTEGER        BufferLength,      /* cbValueMax */
                SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

这将一次检索 *BufferLength* 个字节，而 *StrLen_or_IndPtr* 指示余下的字节数。如果有字节余下，函数的返回值为 SQL_SUCCESS_WITH_INFO（带有 SQLSTATE 01004）。或者，如果返回值为 SQL_SUCCESS，则 *StrLen_or_IndPtr* 指示 DB2 Everyplace CLI 可在 *TargetValuePtr* 缓冲区中返回的字节数。如果 C 数据类型（*TargetType*）为 SQL_C_CHAR 或 SQL_C_BINARY，或者如果 *TargetType* 为 SQL_C_DEFAULT 且列类型表示二进制或字符串，则可以此方式使用 *SQLGetData()* 来检索长列。

要使用 *SQLGetData()* 的这一功能，必须先将语句属性 SQL_ATTR_GETDATA_MODE 设置为 SQL_PIECEMEAL_DATA。此属性的缺省值为 SQL_CHUNK_DATA。这两种方式之间的区别在于: 在 SQL_CHUNK_DATA 方式（这是缺省方式）中，当发生截断时，SQLGetData() 的返回值 *StrLen_or_IndPtr* 指示此列的总字节数，而第二个调用仍然从该列的最开始检索数据。

示例代码片段:

```
sqlrc = SQLSetStmtAttr(hstmt, SQL_ATTR_GETDATA_MODE, (SQLPOINTER) SQL_PIECEMEAL_DATA, 0);
    SQLCHAR * stmt = (SQLCHAR *) "SELECT blobColumn FROM t1 where c1 = ?";
    sqlrc = SQLPrepare( hstmt, stmt, SQL_NTS ) ;
    sqlrc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, ...);
    sqlrc = SQLExecute( hstmt ) ;
    sqlrc = SQLFetch( hstmt );
    /* get BUFSIZ bytes at a time, bufInd indicates number of Bytes LEFT */
    sqlrc = SQLGetData (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer, BUFSIZ, &bufInd);
```

```
while( sqlrc == SQL_SUCCESS_WITH_INFO ) {
    // handle BUFSIZ bytes of blob data in buffer
        :
    sqlrc = SQLGetData (hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer, BUFSIZ, &bufInd);
}
if (sqlrc == SQL_SUCCESS) {  /* partial buffer on last GetData */
    // handle bufInd bytes of blob data in buffer
        :
}
```

第 10 章 参数标记

本章提供有关在 DB2 Everyplace 查询中使用参数标记的信息。涉及的主题包括:

- 『参数标记的概述』
- 『参数标记使用的示例』
- 第 72 页的『DB2 Everyplace 支持的参数标记』

参数标记的概述

对于需要执行多次的 SQL 语句，通常准备 SQL 语句一次并通过在运行时期间使用参数标记来替换输入值以重复使用查询方案比较好。

在 DB2 Everyplace 中，参数标记由 “?” 字符表示并指示在 SQL 语句中的何处替换应用程序变量。参数标记将按编号引用，且它们是从左至右连续编号的（从 1 开始）。在执行 SQL 语句之前，应用程序必须将变量存储区绑定至在 SQL 语句中指定的每个参数标记。此外，绑定变量必须是有效的存储区且在对数据库执行准备语句时必须包含输入数据值。

以下示例演示包含两个参数标记的 SQL 语句。

```
SELECT * FROM customers WHERE custid = ? AND lastname = ?
```

相关概念:

- 『参数标记使用的示例』

参数标记使用的示例

DB2 Everyplace 提供一组丰富的标准接口（包括 CLI/ODBC、JDBC 和 ADO.NET）以有效地存取数据。以下示例代码片段显示将带有参数标记的准备语句用于每个数据存取 API 的情况。

考虑表 t1 的以下表模式，其中列 c1 是表 t1 的主键。

表 13. 示例表模式

列名	DB2 Everyplace 数据类型	可空
c1	INTEGER	false
c2	SMALLINT	true
c3	CHAR(20)	true
c4	VARCHAR(20)	true
c5	DECIMAL(8,2)	true
c6	DATE	true
c7	TIME	true
c8	TIMESTAMP	true
c9	BLOB(30)	true

以下示例演示如何使用准备语句将一行插入表 t1 中。

CLI 示例

```
void parameterExample1(void)
{
   SQLHENV henv;
   SQLHDBC hdbc;
   SQLHSTMT hstmt;
   SQLRETURN rc;
   TCHAR server[] = _T("C:\\mysample\\");
   TCHAR uid[] = _T("db2e");
   TCHAR pwd[] = _T("db2e");
   long p1 = 10;
   short p2 = 100;
   TCHAR p3[100];
   TCHAR p4[100];
   TCHAR p5[100];
   TCHAR p6[100];
   TCHAR p7[100];
   TCHAR p8[100];
   char  p9[100];
   long len = 0;

   _tcscpy(p3, _T("data1"));
   _tcscpy(p4, _T("data2"));
   _tcscpy(p5, _T("10.12"));
   _tcscpy(p6, _T("2003-06-30"));
   _tcscpy(p7, _T("12:12:12"));
   _tcscpy(p8, _T("2003-06-30-17.54.27.710000"));

   memset(p9, 0, sizeof(p9));
   p9[0] = 'X';
   p9[1] = 'Y';
   p9[2] = 'Z';

   rc = SQLAllocEnv(&henv);
   // check return code ...

   rc = SQLAllocConnect(henv, &hdbc);
   // check return code ...

   rc = SQLConnect(hdbc, (SQLTCHAR*)server, SQL_NTS,
    (SQLTCHAR*)uid, SQL_NTS, (SQLTCHAR*)pwd, SQL_NTS);
   // check return code ...

   rc = SQLAllocStmt(hdbc, &hstmt);
   // check return code ...

   // prepare the statement
   rc = SQLPrepare(hstmt, _T("INSERT INTO t1 VALUES (?,?,?,?,?,?,?,?,?)"), SQL_NTS);
   // check return code ...

   // bind input parameters
   rc = SQLBindParameter(hstmt, (unsigned short)1, SQL_PARAM_INPUT,
       SQL_C_LONG, SQL_INTEGER, 4, 0, &p1, sizeof(p1), &len);
   // check return code ...

   rc = SQLBindParameter(hstmt, (unsigned short)2, SQL_PARAM_INPUT, SQL_C_LONG,
       SQL_SMALLINT, 2, 0, &p2, sizeof(p2), &len);
   // check return code ...
```

```
   len = SQL_NTS;
   rc = SQLBindParameter(hstmt, (unsigned short)3, SQL_PARAM_INPUT, SQL_C_TCHAR,
       SQL_CHAR, 0, 0, &p3[0], 100, &len);
   // check return code ...

   rc = SQLBindParameter(hstmt, (unsigned short)4, SQL_PARAM_INPUT, SQL_C_TCHAR,
       SQL_VARCHAR, 0, 0, &p4[0], 100, &len);
   // check return code ...

   rc = SQLBindParameter(hstmt, (unsigned short)5, SQL_PARAM_INPUT, SQL_C_TCHAR,
       SQL_DECIMAL, 8, 2, &p5[0], 100, &len);
   // check return code ...

   rc = SQLBindParameter(hstmt, (unsigned short)6, SQL_PARAM_INPUT, SQL_C_TCHAR,
       SQL_TYPE_DATE, 0, 0, &p6[0], 100, &len);
   // check return code ...

   rc = SQLBindParameter(hstmt, (unsigned short)7, SQL_PARAM_INPUT, SQL_C_TCHAR,
       SQL_TYPE_TIME, 0, 0, &p7[0], 100, &len);
   // check return code ...

   rc = SQLBindParameter(hstmt, (unsigned short)8, SQL_PARAM_INPUT, SQL_C_TCHAR,
       SQL_TYPE_TIMESTAMP, 0, 0, &p8[0], 100, &len);
   // check return code ...

   len = 3;
   rc = SQLBindParameter(hstmt, (unsigned short)9, SQL_PARAM_INPUT, SQL_C_BINARY,
       SQL_BINARY, 0, 0, &p9[0], 100, &len);
   // check return code ...

   // execute the prepared statement
   rc = SQLExecute(hstmt);
   // check return code ...

   rc = SQLFreeStmt(hstmt, SQL_DROP);
   // check return code ...

   rc = SQLDisconnect(hdbc);
   // check return code ...

   rc = SQLFreeConnect(hdbc);
   // check return code ...

   rc = SQLFreeEnv(henv);
   // check return code ...

}
```

JDBC 示例

```
public static void parameterExample1() {

   String driver = "com.ibm.db2e.jdbc.DB2eDriver";
   String url    = "jdbc:db2e:mysample";
   Connection conn = null;
   PreparedStatement pstmt = null;

 try
   {
      Class.forName(driver);
```

```
      conn = DriverManager.getConnection(url);

      // prepare the statement
      pstmt = conn.prepareStatement("INSERT INTO t1 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");

      // bind the input parameters
      pstmt.setInt(1, 1);
      pstmt.setShort(2, (short)2);
      pstmt.setString(3, "data1");
      pstmt.setString(4, "data2");
      pstmt.setBigDecimal(5, new java.math.BigDecimal("12.34"));
      pstmt.setDate(6, new java.sql.Date(System.currentTimeMillis() ) );
      pstmt.setTime(7,  new java.sql.Time(System.currentTimeMillis() ) );
      pstmt.setTimestamp (8,  new java.sql.Timestamp(System.currentTimeMillis() ) );
      pstmt.setBytes(9, new byte[] { (byte)'X', (byte)'Y', (byte)'Z' } );

      // execute the statement
      pstmt.execute();

      pstmt.close();

      conn.close();
   }
        catch (SQLException sqlEx)
   {
            while(sqlEx !=null)
      {
                System.out.println("SQLERROR:\n"+sqlEx.getErrorCode()+
                    ",SQLState:"+sqlEx.getSQLState()+
                    ",Message:"+sqlEx.getMessage()+
                    ",Vendor:"+sqlEx.getErrorCode());
                sqlEx =sqlEx.getNextException();
      }
   }
        catch (Exception ex)
   {
            ex.printStackTrace();
   }
}
```

ADO.NET 示例

[C#]

```
public static void ParameterExample1()
{
   DB2eConnection conn = null;
   DB2eCommand cmd  = null;
   String connString   = @"database=.\; uid=db2e; pwd=db2e";
   int i = 1;

 try
   {
      conn = new DB2eConnection(connString);

      conn.Open();

      cmd = new DB2eCommand("INSERT INTO t1 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)", conn);

      // prepare the command
```

```
   cmd.Prepare();

   // bind the input parameters
   DB2eParameter p1 = new DB2eParameter("@p1", DB2eType.Integer);
   p1.Value = ++i;
   cmd.Parameters.Add(p1);

   DB2eParameter p2 = new DB2eParameter("@p2", DB2eType.SmallInt);
   p2.Value = 100;
   cmd.Parameters.Add(p2);

   DB2eParameter p3 = new DB2eParameter("@p3", DB2eType.Char);
   p3.Value = "data1";
   cmd.Parameters.Add(p3);

   DB2eParameter p4 = new DB2eParameter("@p4", DB2eType.VarChar);
   p4.Value = "data2";
   cmd.Parameters.Add(p4);

   DB2eParameter p5 = new DB2eParameter("@p5", DB2eType.Decimal);
   p5.Value = 20.25;
   cmd.Parameters.Add(p5);

   DB2eParameter p6 = new DB2eParameter("@p6", DB2eType.Date);
   p6.Value = DateTime.Now;
   cmd.Parameters.Add(p6);

   DB2eParameter p7 = new DB2eParameter("@p7", DB2eType.Time);
   p7.Value = new TimeSpan(23, 23, 23);
   cmd.Parameters.Add(p7);

   DB2eParameter p8 = new DB2eParameter("@p8", DB2eType.Timestamp);
   p8.Value = DateTime.Now;
   cmd.Parameters.Add(p8);

   byte []barr = new byte[3];
   barr[0] = (byte)'X';
   barr[1] = (byte)'Y';
   barr[2] = (byte)'Z';

   DB2eParameter p9 = new DB2eParameter("@p9", DB2eType.Blob);
   p9.Value = barr;
   cmd.Parameters.Add(p9);

   // execute the prepared command
   cmd.ExecuteNonQuery();
}
catch (DB2eException e1)
{
   for (int i=0; i < e1.Errors.Count; i++)
   {
      Console.WriteLine("Error #" + i + "\n" +
         "Message: " + e1.Errors[i].Message + "\n" +
         "Native: " + e1.Errors[i].NativeError.ToString() + "\n" +
         "SQL: " + e1.Errors[i].SQLState + "\n");
   }
}
catch (Exception e2)
{
    Console.WriteLine(e2.Message);
```

```
    }
      finally
    {
       if (conn != null && conn.State != ConnectionState.Closed)
       {
             conn.Close();
             conn = null;
       }
    }
}
```

相关概念:

- 第 67 页的『参数标记的概述』

DB2 Everyplace 支持的参数标记

参数标记（由问号（?）指示）是 SQL 语句中的占位符，它的值在语句执行期间获取。应用程序使用 SQLBindParameter() 来将绑定参数标记与应用程序变量相关联。在执行 SQLExecute() 和 SQLExecDirect() DB2 CLI 函数期间，这些变量的值分别替换每个参数标记。在此过程中，可能会发生数据转换。有关支持的数据类型转换的更多信息，请参阅第 252 页的表 95。

DB2 Everyplace 只支持隐式类型参数标记，这些标记可以在 SQL 语句的所选位置中使用。表 14 列出了参数标记用法的限制。

表 14. 参数标记使用的限制

隐式类型参数标记位置	数据类型
表达式: 在选择列表中单独出现	错误
表达式: 算术运算符的两个操作数	错误
谓词: IN 谓词的左边操作数	错误
谓词: 关系运算符的两个操作数	错误
函数: 聚集函数的操作数	错误

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

第 11 章 连接上下文中的游标行为

在另一语句句柄的写冲突下的常规读游标:

应用程序可有多个语句句柄同时对同一个表执行读写操作。当一个句柄对该表执行写操作（例如，UPDATE、DELETE 或 INSERT）而另一个句柄也正在执行读或写操作时，将发生冲突。DB2 Everyplace 采用的行为是首先执行读操作并且总是读取最新的数据。它忽略写冲突（不管该写冲突是否在使用索引）。例如，假定应用程序有两个语句句柄。句柄 #1 用来从表 T 取装行，而句柄 #2 用来从同一个表删除某些行。很可能已由不同的线程创建了每个句柄（例如，在 Java 线程环境中）。

以下是可能的方案:

```
// Fetch 2 rows from table T
   Statement handle 1: execute "SELECT A FROM T WHERE primary_key < 10"
   Statement handle 1: fetch one row; fetch another row
    // Delete some rows in table T
   Statement handle 2: prepare "DELETE FROM T WHERE primary_key = ?"
   Statement handle 2: execute
    // Continue to fetch one more row from T
   Statement handle 1: fetch one row
```

此时在执行中，语句句柄 #1 将能够继续取装下一行（如果有的话），不管索引是否正被使用。在上面的方案中，因为存在主键而使得索引正被*使用*。解决方法是 DB2 Everyplace 将尝试使用句柄 #1 的当前位置重新定位其游标位置（在前进之前）。如果当前位置不再存在（例如，该行已被另一语句句柄删除），则游标在取装时仅需前进至下一位置。同样，如果下一位置被另一语句句柄删除，游标可以跳过该“空位”以进至下一位置。

另一语句句柄的写冲突下的可滚动游标:

考虑与上一节中类似的示例，但在此示例中读游标将是可滚动游标。如果它是“不敏感”的可滚动游标，这就不是问题，原因是结果集不会按定义更改。如果游标不是“不敏感”的，则采用的行为将与上面描述的常规读游标相似。实际上，发生冲突之后的读游标采用的行为是：根据最新表数据重新计算结果集并维持当前行集的起始位置。如果当前行已删除，则游标将进至下一行。

以下示例演示使用 CLP 的可滚动游标的实例。假定表 T 有六行:

```
create table T (a int, b int)
     create index idx1 on T(a)
     insert into T values (1, 1)
     insert into T values (2, 2)
     insert into T values (3, 1)
     insert into T values (3, 2)
     insert into T values (3, 3)
     insert into T values (4, 4)
```

一般来说，可以考虑这样一个示例：应用程序有两个语句句柄，一个用于读取，而另一个用于删除。

```
Statement handle 1: enable scrollable cursor;
Statement handle 1: execute "SELECT A FROM T WHERE a < 10"
Statement handle 2: prepare "DELETE FROM T WHERE a = ?"
Statement handle 1: fetchscroll with SQL_FETCH_FIRST
-- get (1, 1)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- get (2, 2)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
```

```
-- get (3, 1)
Statement handle 2: execute
--- suppose delete row (2, 2)
Statement handle 1: fetchscroll with SQL_FETCH_NEXT
-- re-compute previous rows, and return (3, 2)
Statement handle 1: fetchscroll with SQL_FETCH_PRIOR
-- get (3, 1)
Statement handle 1: fetchscroll with SQL_FETCH_PRIOR
-- get (1, 1)  note that (2, 2) is gone
Statement handle 1: fetchscroll with SQL_FETCH_ABSOLUTE, offset 2
-- get (3, 1)  note that (2, 2) is gone
Statement handle 1: fetchscroll with SQL_FETCH_ABSOLUTE, offset 5
-- get (4, 4)
```

落实和回滚下的游标，包括自动落实方式:

不管是事务或是自动落实方式，打开的游标在落实期间将保持打开状态，而在回滚时关闭。

对象相关性:

通过语句句柄 H 准备 SQL 语句可能会对特定对象设置某些相关性。例如，通过索引 Idx 从表 T 选择行需要表 T 和索引 Idx 的存在。如果这些对象被另一语句句柄删除（例如，如果索引 Idx 被删除），通过 H 重新执行该语句将强制重新编译 SQL 语句。因此，查询方案可能不同或者可能返回错误。

第 12 章 加密本地数据

本章说明如何加密 DB2 Everyplace 数据库中的本地数据。涉及下列主题:

本地数据加密的概述

DB2 Everyplace 中的加密设计成用于保护移动或嵌入式设备上的数据。本主题提供了本地数据加密的快速概述以帮助您入门。讨论了下列主题:

- 为什么使用本地数据加密?
- 本地数据加密目的
- 创建第一个加密表
- 对加密表的后继访问
- 管理用户特权

为何使用本地数据加密?:

考虑包含客户联系数据的公司销售应用程序。移动售货员可以使此数据出现在他们的 PDA 中以便客户访问。除非应用程序 PDA 提供安全存储系统, 可以使用应用程序或通过调查移动式设备的本地文件系统容易地存取数据。加密敏感数据成为保护公司信息的非常重要的方面。

本地数据加密目标:

DB2 Everyplace 提供允许应用程序实现公司安全策略的解决方案。第一个目标是加密存储在 DB2 Everyplace 表中的机密或敏感信息。使用实现加密密钥的标准加密方法(如 DES)来加密数据。第二个目标是提供安全框架以便能够管理用于加密数据的密钥。要求用户在连接数据库时提供用户标识和密码。有关更多信息, 请参阅第 78 页的『管理加密特权』。

有关使用数据加密的更多信息, 请参阅第 78 页的『使用 DB2eCLP 加密』。

先决条件:

本节描述如何对每个平台启用加密并列示需要的库以及 DB2 Everyplace 数据库必需的库。

对于 Win32:

- 插件库: CryptoPlugin.dll(由 DB2 Everyplace 提供)
- 加密库: Crypt32.dll(128 位加密强度“加密”包, 随 IE5.5 或更高版本一起提供)。转至 http://www.microsoft.com/windows/ie/downloads/critical /q313675/download.asp。

对于 Windows CE/Pocket PC

- 插件库: CryptoPlugin.dll（由 DB2 Everyplace 提供）
- 加密库: Microsoft High Encryption Pack for Pocket PC V1.0。转至 http://www.microsoft.com/mobile/pocketpc/downloads/ssl128.asp。

对于 Palm OS

- 插件库: CryptoPlugin.PRC（provided by DB2 Everyplace）
- 加密库: PBSPKcs11.prc（由 DB2 Everyplace 提供）

对于 Linux/Neutrino

- 插件库: libcryptoplugin.so（由 DB2 Everyplace 提供）
- 加密库: libpvcpkcs11.so（由 DB2 Everyplace 提供）

对于 Symbian

- 插件库: CRYPTOPLUGIN.DLL（由 DB2 Everyplace 提供）
- 加密库: ECSPKCS11.DLL（由 DB2 Everyplace 提供）

过程:

要使用数据加密:

1. 建立与 DB2 Everyplace 数据库的连接。
2. 授予用户加密特权。
3. 创建第一个加密表。

   对加密表的后续访问: 如果数据库包含 DB2eSYSUSERS 表，任何后续数据库连接将使用提供的用户标识和密码进行用户认证。如果认证失败，则应用程序仅可以访问非加密表。应用程序不能创建新的加密表，不能删除现有的加密表或存取和更新加密数据。
4. 管理加密特权。

建立与 DB2 Everyplace 数据库的连接

此任务是“加密本地数据”这一主任务的一部分。完成这些步骤后，请返回至第 75 页的『本地数据加密的概述』。

过程:

与 DB2 Everyplace 数据库的任何交互作用都要求建立连接。另外，为了用户访问或创建加密表，应用程序必须使用下列 CLI 函数用非空用户标识和密码连接至 DB2 Everyplace:

```
rc = SQLConnect(hdbc, "C:\temp\", SQL_NTS, "user1", SQL_NTS, "pwd1", SQL_NTS)
```

其中 *"C:\temp\"* 是应用程序连接至的数据库的目录，使用用户标识 *"user1"* 和密码 *"pwd1"*。

对于 JDBC 接口，可以用类似方法建立数据库连接。

相关概念:

- 第 63 页的『连接串行化』

授予用户加密特权

此任务是“加密本地数据”这一主任务的一部分。完成这些步骤后，请返回至第 75 页的『本地数据加密的概述』。

过程:

在创建第一个加密表之前，必须授予用户加密特权。例如，应用程序可以发出下列 SQL 语句:

```
rc = SQLExecDirect(..., "GRANT ENCRYPT ON DATABSE TO \" user1\"" +
                         " using \"pwd1\" new \"pwd1\"", SQL_NTS)
```

在执行此 SQL 语句时，DB2 Everyplace 将创建称为 DB2eSYSUSERS 的系统目录表，且将一行插入到此表。这意味着现在已用对应的密码注册用户 *user1*，且该用户将具有诸如创建和访问加密表的所有加密特权。

此表与数据库和加密数据有密切关系，因此它不能移动到另一个 DB2 Everyplace 数据库来存取加密数据。这是因为不同的数据库将具有不同的密钥用于加密或解密。因此，如果允许某人存取某个数据库中的加密表，则该人不能使用相同用户标识和密码存取不同数据库。与其它系统目录表一样，应用程序可以使用 SQS 选择语句来检索行，但它不能使用 INSERT、DELETE、 UPDATE、CREATE 或 DROP 语句修改此表中的数据。

相关概念:

- 第 78 页的『使用 DB2eCLP 加密』

相关任务:

- 第 78 页的『使用 DB2eCLP 加密』
- 『创建加密表』
- 第 78 页的『管理加密特权』

创建加密表

此任务是“加密本地数据”这一主任务的一部分。完成这些步骤后，请返回至第 75 页的『本地数据加密的概述』。

过程:

一旦建立了与 DB2 Everyplace 数据库的连接，且授予了用户加密特权，应用程序就可以使用扩展的 CREATE TABLE 语句创建加密表。例如，可以创建以下雇员表:

```
SQLExecDirect(..., "CREATE TABLE EMPLOYEES (EMPNO INT PRIMARY KEY, NAME VARCHAR(30),
SALARY DECIMAL(10,2)) WITH ENCRYPTION", SQL_NTS)
```

相关概念:

- 第 78 页的『使用 DB2eCLP 加密』

相关任务:

- 第 78 页的『使用 DB2eCLP 加密』
- 『授予用户加密特权』
- 第 78 页的『管理加密特权』

管理加密特权

此任务是“加密本地数据”这一主任务的一部分。完成这些步骤后，请返回至第 75 页的『本地数据加密的概述』。

过程:

一旦应用程序使用已认证的用户标识和密码连接到某个数据库，应用程序就可以创建新用户、更改密码或从系统除去已注册的用户。创建新用户或更改密码的语法是:

```
GRANT ENCRYPT ON DATABASE TO "newuser" USING "grantorpassword" NEW "newpassword"
```

除去注册用户的语法是:

```
REVOKE ENCRYPT ON DATABASE FROM "user"
```

注: 如果从 DB2eSYSUSERS 表除去所有注册用户（使用 REVOKE 语句），则不再能执行加密操作，包括访问现有的加密表。没有恢复机制。

相关概念:

- 『使用 DB2eCLP 加密』

相关任务:

- 『使用 DB2eCLP 加密』
- 第 77 页的『创建加密表』
- 第 77 页的『授予用户加密特权』

使用 DB2eCLP 加密

本节包含交互式会话的示例，该交互式会话设计为显示如何在应用程序中使用数据加密。已添加注释说明每个操作。

```
-- Encryption using DB2eCLP
--
-- This is an example encryption session using the provided sample
-- command line interface program DB2eCLP.
--
-- We only show the return code of a statement if it
-- failed, if it completed successfully we only show the results
-- of selects.
-- Commands which can be typed into DB2 Everyplace are
-- prefixed by the string "CLP:> ".
--
-- -- (CLI:SQLConnect, SQL:CREATE TABLE, SQL:GRANT, SQL:REVOKE)
--
-- When you start DB2eCLP you are automatically
-- connected to the default database (in the current directory).
-- This is equivalent to:
--
CLP:> CONNECT TO anything;

-- Since no specific path is given, just a name "anything", it connects
-- to the current directory.
--
-- We will now create a non-encrypted table containing a mapping of
-- some numbers to Swedish counting words.

CLP:> CREATE TABLE swedish(nummer INT, ord VARCHAR(32));
CLP:> INSERT INTO swedish VALUES(1, 'ett');
CLP:> INSERT INTO swedish VALUES(3, 'tre');
CLP:> INSERT INTO swedish VALUES(4, 'fyra');
CLP:> INSERT INTO swedish VALUES(5, 'fem');
CLP:> INSERT INTO swedish VALUES(7, 'sju');
CLP:> INSERT INTO swedish VALUES(99, 'nittionio');

-- Just have a look at the data
CLP:> SELECT * FROM swedish;
```

```
NUMMER      ORD
----------- --------------------------------
          1 ett
          3 tre
          4 fyra
          5 fem
          7 sju
         99 nittionio
6 row(s) returned.

-- We will now try to create the corresponding table for English,
-- but using encryption.
--
CLP:> CREATE TABLE english(number INT, word VARCHAR(32)) WITH ENCRYPTION;
Statement failed [sqlstate = 42501].

-- This fails because we are not authorized yet. As indicated by the error code.
-- So we need to connect again:
--
CLP:> CONNECT TO something USER jsk USING hemligt;

-- This connects to the same database (default/current directory) but with
-- a specific user identity "jsk" and using the password "hemligt".
-- The CONNECT TO command is not an SQL statement, thus is
-- interpreted by the DB2eCLP application. It will
-- disconnect and connect again to the DB2 Everyplace database
-- using:
--    SQLConnect(hdbc, "something", SQL_NTS, "jsk", SQL_NTS, "hemligt", SQL_NTS);
--
-- Now, we have to create the first authorized user. When the
-- first user is created it has to have the same name as the
-- logged in user and the same password:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "hemligt" NEW "hemligt";

-- Notice that for GRANT the name and passwords need to be inside
-- double quotes. This is because they are case-sensitive, and
-- the statement is passed directly to DB2 Everyplace.
--
-- Now that we have an authorized encryption user we can create the
-- encrypted table:
--

CLP:> CREATE TABLE english(number INT, word VARCHAR(32)) WITH ENCRYPTION;
CLP:> INSERT INTO english VALUES(1, 'one');
CLP:> INSERT INTO english VALUES(3, 'three');
CLP:> INSERT INTO english VALUES(4, 'four');
CLP:> INSERT INTO english VALUES(5, 'five');
CLP:> INSERT INTO english VALUES(7, 'seven');
CLP:> INSERT INTO english VALUES(99, 'ninety nine');

-- Just have a look at the data.
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- Select a large random number in Swedish:
--
CLP:> SELECT * FROM swedish WHERE nummer > 42;

NUMMER      ORD
----------- --------------------------------

         99 nittionio
1 row(s) returned.

-- Select a large random number in English:
--
CLP:> SELECT * FROM english WHERE number > 42;

NUMBER      WORD
----------- --------------------------------
         99 ninety nine
1 row(s) returned.
```

```
-- Translate 'fyra' to english:
--
CLP:> SELECT word FROM swedish, english WHERE number = nummer AND ord = 'fyra';

WORD
--------------------------------
four
1 row(s) returned.

-- Get a translation table:
--
CLP:> SELECT number, ord, word FROM swedish, english WHERE number = nummer;

NUMBER      ORD                            WORD
----------- ------------------------------ --------------------------------
          1 ett                            one
          3 tre                            three
          4 fyra                           four
          5 fem                            five
          7 sju                            seven
         99 nittionio                      ninety nine
6 row(s) returned.

--Attempt to authorize another user to access the encrypted data with her
-- own password:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "notKnown" NEW "notKnown";
Statement failed [sqlstate = 42506].

-- Failed because the user who is logged in must validate himself
-- in order to add a new user, this is done by providing his password
-- after the USING clause.
--
CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "hemligt" NEW "notKnown";

-- Let's reconnect with the new user:
--
CLP:> CONNECT TO samething USER xin USING notknown;
Statement failed [sqlstate = 42505].

-- This fails, because the password is not the same, thus will not generate
-- the same key and access is denied.
--
CLP:> CONNECT TO samething USER ksin USING notKnown;

-- This will not fail, because the user ksin does not exist, and we therefore
-- do not attempt to authenticate the user.
-- However, using SQLGetInfo one can distinguish this case
-- from the case where an user was successfully authenticated.
--
CLP:> SELECT * FROM swedish;

NUMMER      ORD
----------- --------------------------------
          1 ett
          3 tre
          4 fyra
          5 fem
          7 sju
         99 nittionio
6 row(s) returned.

-- Selecting non-encrypted data works fine, however encrypted data cannot
-- be read/updated unless an authorized user is connected:
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- Connect as the new user, finally with correct username and password.
--
CLP:> CONNECT TO samething USER xin USING notKnown;

-- Verify that we are authenticated and can access the data.
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
```

```
6 row(s) returned.

-- Add another user:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "thf" USING "notKnown" NEW "heimlich";

-- List currently existing users:
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
jsk                 jsk
xin                 jsk
thf                 xin
3 row(s) returned.

-- Again connect as "jsk":
--
CLP:> CONNECT TO itagain USER jsk USING hemligt;
Statement completed successfully.

-- Attempt to change the password to "secret":
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "secret" NEW "secret";
Statement failed [sqlstate = 42506].

-- Ah, we failed because we need to supply first our old password and then
-- the new password:
--
CLP:> GRANT ENCRYPT ON DATABASE TO "jsk" USING "hemligt" NEW "secret";

-- Try the new password:
--
CLP:> CONNECT TO itagain USER jsk USING secret;

-- Make sure we can access encrypted ata:
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- Let's remove encryption privilege from "xin":
--
CLP:> REVOKE ENCRYPT ON DATABASE FROM "xin";

-- List users
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
jsk                 jsk
thf                 xin
2 row(s) returned.

-- Connect again to the now non-existing user, without error.
--
CLP:> CONNECT TO thedatabase USER xin USING idontknow;

-- Attempt to do encryption operations without authorization:
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

CLP:> DROP TABLE english;
Statement failed [sqlstate = 42501].

CLP:> REVOKE ENCRYPT FROM "jsk";
Statement failed [sqlstate = 42601].

CLP:> GRANT ENCRYPT ON DATABASE TO "xin" USING "idontknow" NEW "idontknow";
Statement failed [sqlstate = 42502].

-- Connect as "thf":
--
```

```
CLP:> CONNECT TO thedatabase USER thf USING heimlich;

-- Check that we can read encrypted data:
--
CLP:> SELECT * FROM english;

NUMBER      WORD
----------- --------------------------------
          1 one
          3 three
          4 four
          5 five
          7 seven
         99 ninety nine
6 row(s) returned.

-- Let's remove the connected user's privilege:
--
CLP:> REVOKE ENCRYPT ON DATABASE FROM "thf";

-- Make sure he cannot access the data anymore:
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- If we connect to the database as the only remaining user "jsk"
--
CLP:> CONNECT TO thedatabase USER jsk USING secret;

-- We remove the connected user, that user can not access the data anymore.
-- Actually, there is no way to access the encrypted data ever again.
--

CLP:> REVOKE ENCRYPT ON DATABASE FROM "jsk";

-- Make sure there are no users left:
--
CLP:> SELECT username, grantorname FROM "DB2eSYSUSERS";

USERNAME            GRANTORNAME
------------------- -------------------
0 row(s) returned.

-- We should now not be able to access the encrypted data.
--
CLP:> SELECT * FROM english;
Statement failed [sqlstate = 42501].

-- This concludes the example session.
```

相关任务:

- 第 75 页的『本地数据加密的概述』
- 第 77 页的『授予用户加密特权』
- 第 77 页的『创建加密表』
- 第 78 页的『管理加密特权』

第 3 部分 样本应用程序

第 13 章 样本 C/C++ 应用程序

为每个操作系统提供了至少一个样本 C/C++ 应用程序。查看适当的客户机目录以获取带有源代码的完整的样本应用程序。

相关任务:

- 第 9 页的『开发 DB2 Everyplace C/C++ 应用程序』

相关参考:

- 第 9 页的『支持的 C/C++ 开发工具』
- 第 11 页的『支持 C/C++ 的操作系统』

第 14 章 样本 Java 应用程序

本章提供有关样本 Java 应用程序的信息。涉及的主题包括:

- 『样本 Java 应用程序的概述』
- 第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』
- 第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』

样本 Java 应用程序的概述

本主题描述 DB2eAppl.java 和 DB2eJavaCLP.java 样本应用程序。

样本 1: DB2eAppl.java:

DB2eAppl.java 演示如何编写用于 DB2 Everyplace 的 JDBC 程序。

DB2eAppl.java 应用程序的主要步骤包括:

步骤 1:
导入 java.sql 包。

步骤 2:
装入 DB2 Everyplace JDBC 驱动程序 com.ibm.db2e.jdbc.DB2eDriver。

步骤 3:
连接至当前目录(DB2eAppl.java 应用程序将在该目录中运行)中的数据库。

步骤 4:
创建“语句”对象。

步骤 5:
设置非常简单的样本数据库,该数据库由包含两个记录的 EMPLOYEE 表组成。这是通过使用 java.sql.Statement 接口的 executeUpdate(String sql) 方法完成的。

步骤 6:
从 EMPLOYEE 表中选择所有记录,然后通过使用 java.sql.ResultSet 接口的 next() 方法来检索这些行。

步骤 7:
从数据库删除 EMPLOYEE 表并释放数据库和 JDBC 资源。

以下 DB2eAppl.java 源代码包含一些注释,这些注释显示正在何处使用以上说明的步骤。

```
import java.sql.*; //Step 1

public class DB2eAppl
{
    public static void main(String[] args) {

        String driver ="com.ibm.db2e.jdbc.DB2eDriver";
String url = "jdbc:db2e:mysample";
            try {
            Class.forName(driver);//Step 2
            Connection con = DriverManager.getConnection(url);//Step 3
            Statement st = con.createStatement();//Step 4
```

```
            //Create table: employee  //Step 5
            st.executeUpdate("CREATE TABLE employee (EMPNO CHAR(6), FIRSTNAME VARCHAR(12))");
            System.out.println("*** Created table: employee");

    //Add records to employee
            st.executeUpdate("INSERT INTO employee VALUES ('112233','John')");
            st.executeUpdate("INSERT INTO employee VALUES ('445566','Mary')");
            System.out.println("*** Inserted two records");

            //Query and display results //Step 6
            ResultSet rs = st.executeQuery("SELECT * FROM employee");
            System.out.println("*** Query results:");
            while (rs.next()) {
                System.out.print("EMPNO=" + rs.getString(1) + ", ");
                System.out.println("FIRSTNAME=" + rs.getString(2));
            }

            //Delete table: employee  //Step 7
            st.executeUpdate("Drop table employee");
            System.out.println("*** Deleted table: employee");

    rs.close();
    st.close();
    con.close();

         } catch (SQLException sqlEx) {
             while(sqlEx !=null)
             {
                 System.out.println("[SQLException] " +
                                    "SQLState: " + sqlEx.getSQLState() +
                     ",Message:"+sqlEx.getMessage()+
                     ",Vendor:"+sqlEx.getErrorCode());
                 sqlEx =sqlEx.getNextException();
             }
         } catch (Exception ex) {
             ex.printStackTrace();
         }
     }
}
```

样本 2: DB2eJavaCLP.java:

DB2eJavaCLP.java 是用于 DB2 Everyplace 的 Java 命令行处理器。

限制: 在 Palm OS 上，不支持 DB2eJavaCLP.java 样本应用程序。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』
- 第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』
- 第 90 页的『为 Palm OS 目标创建用于 DB2eAppl.java 的 WSDD 项目』
- 第 94 页的『为非 Palm OS 目标创建 WSDD 项目并将 JAR 文件添加至 DB2eAppl.java 的构建路径』
- 第 95 页的『在 Win32 上运行 DB2eAppl.java』
- 第 96 页的『在 Windows CE 上运行 DB2eAppl.java』
- 第 91 页的『在 Palm OS 仿真器上运行 DB2eAppl.java』
- 第 98 页的『在 QNX Neutrino 或嵌入式 Linux 上运行 DB2eAppl.java』
- 第 99 页的『在 Symbian 上运行 DB2eAppl.java』
- 第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』

相关概念:

- 第 111 页的『样本 Java 本机同步应用程序』

在 Palm OS 目标上编译和运行样本 Java 应用程序

以下一组主题描述如何在 Palm OS 目标上编译和运行样本 DB2eAppl.java Java 代码。

建议使用 WebSphere Studio Device Developer（WSDD）作为开发环境。WSDD 使用 J9 VM，它可能不支持您的设备的处理器类型。如果使用不同的开发环境和 JVM，确保 JVM 支持 JNI，因为 DB2 Everyplace JDBC 驱动器使用 JNI。其它兼容的 JVM 包括 Sun PersonalJava、Insignia Jeode 和 NSIcom CrEme。如果当前目标为 Palm OS，则*必须*使用 WSDD 附带的 J9 VM。WSDD 的试用版本可从 http://www.ibm.com/software/pervasive/products/wsdd/ 下载。

先决条件:

1. 确保安装了以下软件:
   - WSDD 5.5，它包括 J9 Java 虚拟机
2. 根据 WSDD 文档准备目标和开发环境。通过构建和运行 WSDD 样本应用程序验证 WSDD 安装。
3. 在目标设备上安装 DB2 Everyplace。有关详细指示信息，请参阅《*DB2 Everyplace 安装与用户指南*》。

过程:

要在 Palm OS 目标上编译和运行样本 Java 代码:

1. 安装 WCE Tooling for WSDD。
2. 为 DB2eAppl.java 创建 WSDD 项目。
3. 将 DB2 Everyplace JDBC 驱动程序和 java.sql 包添加至构建路径。
4. 将 DB2eAppl.java 导入到 WSDD 中。
5. 在 Palm OS 仿真器上运行 DB2eAppl.java。

相关任务:

- 第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』

为 Palm OS 目标安装 WCE Tooling for WSDD

此任务是“在 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

1. 在 WSDD 中，单击**帮助 —> 软件更新 —> 更新管理器**以打开“安装／更新透视图”
2. 在“安装／更新透视图”的“功能部件更新视图”中，展开下列各项: **要访问的站点 —> WebSphere 定制环境 —> WebSphere 定制环境**。
3. 选择 WCE Tooling for WSDD 5.5.0。
4. 在“预览”视图中，单击**安装**。
5. 遵循安装指示信息以安装 WCE Tooling for WSDD 功能部件。
6. 如果选择使用 jclXtr 类库，则还必须通过遵循类似步骤安装“WCE jclXtr 类库”。

返回至『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

为 Palm OS 目标创建用于 DB2eAppl.java 的 WSDD 项目

有两个版本的 DB2 Everyplace JDBC Driver for Palm OS 可用。一个版本与 “J2ME CLDC 配置” 兼容。另一个版本与 WSDD 提供的 “JCL Extreme Palm 定制配置” 兼容。遵循相应的步骤来创建您需要的项目类型。

此任务是 “在 Palm OS 目标上编译和运行样本 Java 应用程序” 这一主任务的一部分。完成这些步骤后，请返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

要使用 jclCldc 配置创建 WSDD Palm OS 项目: :

1. 在 WSDD 中，单击**窗口 —> 打开透视图 —> Java** 以切换至 “Java 透视图”。
2. 创建 DB2 Everyplace Sample for Palm OS CLDC 项目:
   a. 单击**文件 —> 新建 —> 其它**。
   b. 在 “新建项目” 窗口的 “选择” 页上，在左窗格中选择 J2ME for J9，在右窗格中选择 “创建 J2ME 项目”，然后单击**下一步**。
   c. 在 “新建项目” 窗口的 “J2ME 项目” 页上，在 “项目名” 字段中输入 `DB2 Everyplace Sample for Palm OS CLDC`，然后单击**下一步**。
   d. 在 “新建项目” 窗口的 “库选择” 页上，选择 WME jclCldc（jclCldc），然后单击**完成**。

要使用 jclXtr 配置创建 WSDD Palm OS 项目: :

1. 在 WSDD 中，单击**窗口 —> 打开透视图 —> Java** 以切换至 “Java 透视图”。
2. 创建 DB2 Everyplace Sample for Palm OS XTR 项目:
   a. 单击**文件 —> 新建 —> 其它**。
   b. 在 “新建项目” 窗口的 “选择” 页上，在左窗格中选择 WCE for J9，在右窗格中选择 “创建 WCE 项目”，然后单击**下一步**。
   c. 在 “新建项目” 窗口的 “定制项目” 页上，在 “项目名” 字段中输入 `DB2 Everyplace Sample for Palm OS XTR`，然后单击**下一步**。
   d. 在 “新建项目” 窗口的 “库选择” 页上，选择 WCE jclXtr（jclXtr），然后单击**完成**。

返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

将 DB2 Everyplace JDBC 驱动程序和 java.sql 包添加至构建路径

下列步骤适用于 DB2 Everyplace Sample for Palm OS CLDC 项目。DB2 Everyplace Sample for Palm OS XTR 项目涉及类似步骤。

此任务是 “在 Palm OS 目标上编译和运行样本 Java 应用程序” 这一主任务的一部分。完成这些步骤后，请返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

要将 DB2 Everyplace JDBC 驱动程序（和 `java.sql` 包）添加至构建路径:

1. 在“Java 透视图”的“程序包资源管理器”视图中右键单击 DB2 Everyplace Sample for Palm OS CLDC 项目，然后在弹出菜单上单击**属性**。
2. 在打开的属性窗口中，在左窗格中单击“Java 构建路径”，然后在右窗格中单击“库”选项卡。
3. 单击**添加外部 JAR**。在“JAR 选择”窗口中，浏览至 <DB2Everyplace>\Clients\PalmOS\database\JDBC\cldc\db2ejdbc.jar，然后单击**打开**。
4. 重复先前的步骤以将 database_enabler_cldc.jar 和 DB2eJDBC_Cldc_maps.jar 添加至构建路径。
5. 回到“DB2 Everyplace Sample for Palm OS CLDC 的属性”窗口中，单击**确定**。

返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

将 DB2eAppl.java 导入到 Palm OS 的 WSDD 中

下列步骤适用于 DB2 Everyplace Sample for Palm OS CLDC 项目。DB2 Everyplace Sample for Palm OS XTR 项目包括类似步骤。

此任务是“在 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

要将 DB2eAppl.java 导入到 WSDD 中:

1. 在“Java 透视图”的“程序包资源管理器”视图中，右键单击 DB2 Everyplace Sample for Palm OS CLDC 项目中的 src 文件夹，然后在弹出菜单上单击**导入**。
2. 在“导入”窗口的“选择”页上，选择“文件系统”作为导入源，然后单击**下一步**。
3. 在“导入”窗口的“文件系统”页上，单击**浏览**。
4. 浏览至 <DB2Everyplace>\Clients\PalmOS\database\JDBC\cldc\sample 文件夹，然后单击**确定**。
5. 在右窗格中选择 **DB2eAppl.java** 复选框，然后单击**完成**。

返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

在 Palm OS 仿真器上运行 DB2eAppl.java

下列步骤适用于 DB2 Everyplace Sample for Palm OS CLDC 项目。DB2 Everyplace Sample for Palm OS XTR 项目涉及类似步骤。

此任务是“在 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

先决条件:

如果尚未设置系统来使用 DB2 Everyplace JDBC 驱动程序，对设备上的 JDBC 驱动程序安装下列文件:

<*DB2 Everyplace*>\Clients\PalmOS\database\JDBC\cldc\DB2eJDBC.prc
<*DB2 Everyplace*>\Clients\PalmOS\database\JDBC\cldc\DB2eJDBC_Cldc.prc

如果在使用 DB2 Everyplace Sample for Palm OS XTR 项目，对设备上的 JDBC 驱动程序安装下列文件:

<*DB2 Everyplace*>\Clients\PalmOS\database\JDBC\xtr\DB2eJDBC.prc
<*DB2 Everyplace*>\Clients\PalmOS\database\JDBC\xtr\DB2eJDBC_Xtr.prc

过程:

要在 Palm OS 仿真器上运行 DB2eAppl.java:

1. 配置 Palm OS 仿真器:
   a. 单击**设备 —> 配置**。
   b. 在“设备配置”窗口的左窗格中选择“Palm 仿真器”然后单击**新建**。
   c. 在右边出现的配置中，输入以下信息:
      - 在**设备名**字段中，输入 DB2 Everyplace Palm 仿真器。
      - 在 **Palm 仿真器可执行文件**字段中，浏览至 <*PalmEmulator*>\Emulator.exe，其中 <*PalmEmulator*> 是安装“Palm 仿真器”的目录。
      - 在**仿真器运行自变量**字段中，输入 -psf <*file*>.psf，其中 <*file*>.psf 是安装了 DB2 Everyplace 和 J9 VM 的 .psf 文件。
   d. 单击“应用”，然后单击“确定”。
2. 构建 DB2eAppl.java。
   a. 在“Java 透视图”的“程序包资源管理器”视图中，双击 DB2 Everyplace Sample for Palm OS CLDC 项目的 wsddbuild.xml 文件。
   b. 在 wsddbuild.xml 的编辑器中，单击**添加构建**。
   c. 从**平台**列表中选择 J9 for Palm 68k，保留 **Main 类**和**构建名**字段中的缺省值，然后单击**下一步**。
   d. 在“PalmOS 设置”页上，在**创建者标识**字段中输入 DB2e，然后在**应用程序名**字段中输入 DB2eAppl，然后单击**下一步**。
   e. 在“Jxelink 选项”页上，保留缺省值并单击**完成**。
3. 修改 DB2eAppl.jxeLinkOptions 文件:
   a. 在“Java 透视图”的“程序包资源管理器”中，展开 palm68k 文件夹以获取 DB2 Everyplace Sample for Palm OS CLDC 项目。双击 DB2eAppl.jxeLinkOptions 以打开 DB2eAppl.jxeLinkOptions 的编辑器。
   b. 在编辑器中，单击“输入”选项卡。单击**新建**以从映射文件（prereq）部分获取 Read 类。在出现的“添加 prereq”窗口中，对 prereq 输入 DB2eJDBC_Cldc，然后单击**确定**。**注意**: 如果在使用 DB2 Everyplace Sample for Palm OS XTR 项目，则对 prereq 输入 DB2eJDBC_Xtr，然后跳过接下来的两个步骤。
   c. 在编辑器中，单击 Jxe 选项卡。在“Jxe 平台信息”下，单击**新建**以在运行 jxe 部分时获取“使用 VM 选项”。
   d. 在出现的“添加 VM 选项”窗口中，对 VM 选项输入 -jcl:cldc:loadlibrary=db2ejdbc，然后单击**确定**。
   e. 输入 Ctrl+S 以保存更改。
   f. 在 wsddbuild.xml 的编辑器中，选择 jxe2prc palm68k/DB2eAppl，然后单击**执行构建**。

4. 运行 DB2eAppl.java。
   a. 从主菜单中单击**运行 —> 运行**。 “启动配置”窗口打开。
   b. 在“启动配置”窗口的左窗格中选择“设备 Java 应用程序”，然后单击**新建**。
   c. 在出现在右窗格中的配置中，在**名称**字段中输入 DB2eAppl Palm CLDC。
   d. 在“Java 应用程序”面板中，输入以下信息:
      1) 在“项目”字段中，浏览至 DB2 Everyplace Sample for Palm OS CLDC。
      2) 在“Java 应用程序”字段中单击**搜索**。
      3) 在“选择目标”窗口中，选择 DB2eAppl.prc（在 wsddbuild.xml 中目标为“jxe2prc palm68k/DB2eAppl”），然后单击**完成**。
      4) 从**设备或 JRE** 列表中选择“DB2 Everyplace Palm 仿真器”。
      5) 回到“启动配置”窗口中，单击**应用**，然后单击**运行**。 “Palm 仿真器”应启动并运行 DB2eAppl。应该会在“Palm 仿真器”屏幕或 .psf 文件所在目录中的 j9stdout.txt 文件中看到样本应用程序的输出。如果未修改 J9 Java VM 的“在屏幕上显示 Stdout”首选项，输出将在 j9stdout.txt 文件中。还应检查 j9stderr.txt 是否有错误。

返回至第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』。

在非 Palm OS 目标上编译和运行样本 Java 应用程序

以下一组主题描述如何使用 WebSphere Studio Device Developer（WSDD）5.5 和 J9 Java 虚拟机编译和运行样本 Java 代码。

建议使用 WSDD 作为开发环境。WSDD 使用 J9 VM，它可能不支持您的设备的处理器类型。如果使用不同的开发环境和 JVM，确保 JVM 支持 JNI，因为 DB2 Everyplace JDBC 驱动器使用 JNI。其它兼容的 JVM 包括 Sun PersonalJava、Insignia Jeode 和 NSIcom CrEme。WSDD 的试用版本可从 http://www.ibm.com/software/pervasive/products/wsdd/ 下载。

先决条件:

1. 确保安装了以下软件:
   - WSDD 5.5，包括 J9 Java 虚拟机或某些其它兼容 JVM
2. 根据 WSDD 文档准备目标和开发环境。通过构建和运行 WSDD 样本应用程序验证 WSDD 安装。
3. 在目标设备上安装 DB2 Everyplace。有关详细指示信息，请参阅《*DB2 Everyplace 安装与用户指南*》。

过程:

要在非 Palm OS 目标上编译和运行样本 Java 代码:

1. 安装 WCE Tooling for WSDD。
2. 创建 WSDD 项目并将 JAR 文件添加至 DB2eAppl.java 的构建路径。
3. 将 DB2eAppl.java 导入到 WSDD 中。
4. 运行 DB2eAppl.java。这些步骤根据操作系统的不同而变化。
   - 第 95 页的『在 Win32 上运行 DB2eAppl.java』
   - 第 96 页的『在 Windows CE 上运行 DB2eAppl.java』

- 第 98 页的『在 QNX Neutrino 或嵌入式 Linux 上运行 DB2eAppl.java』
- 第 99 页的『在 Symbian 上运行 DB2eAppl.java』

相关任务:

- 第 89 页的『在 Palm OS 目标上编译和运行样本 Java 应用程序』

为非 Palm OS 目标安装 WCE Tooling for WSDD

此任务是“在非 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

1. 在 WSDD 中，单击**帮助 —> 软件更新 —> 更新管理器**以打开“安装 / 更新透视图”
2. 在“安装 / 更新透视图”的“功能部件更新视图”中，展开下列各项: **要访问的站点 —> WebSphere 定制环境 —> WebSphere 定制环境**。
3. 选择 WCE Tooling for WSDD 5.5.0。
4. 在“预览”视图中，单击**安装**。
5. 遵循安装指示信息以安装 WCE Tooling for WSDD 功能部件。
6. 通过遵循类似步骤安装“WCE 数据库使能器库”和“WCE jclMax 类库”。

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

为非 Palm OS 目标创建 WSDD 项目并将 JAR 文件添加至 DB2eAppl.java 的构建路径

此任务是“在非 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

1. 在 WSDD 中，单击**窗口 —> 打开透视图 —> Java** 以切换至“Java 透视图”
2. 创建 DB2 Everyplace Sample 项目:
   a. 单击**文件 —> 新建 —> 其它**。
   b. 在“新建项目”窗口的“选择”页上，在左窗格中选择 WCE for J9，在右窗格中选择“创建 WCE 项目”，然后单击**下一步**。
   c. 在“新建项目”窗口的“定制项目”页上，对项目名输入 `DB2 Everyplace Sample`，然后单击**下一步**。
   d. 在“新建项目”窗口的“库选择”页上，选择 WCE jclMax（jclMax），然后单击**下一步**。
   e. 在“新建项目”窗口的“Java 设置”页上，单击**库**选项卡。
   f. 将 `db2ejdbc.jar` 添加至构建路径:
      1) 单击**添加外部 JAR**。

2) 在 “JAR 选择” 窗口中，浏览至 ***<DB2Everyplace>***\Clients\Win32\database\jdbc\db2ejdbc.jar，然后单击**打开**。

.

g. 回到 “Java 设置” 页上，将 database_enabler.jar 添加至构建路径。

1) 单击**添加外部 JAR**。

2) 在 “JAR 选择” 窗口中，浏览至 *<WSDD>*\wsdd5.0\ive\lib\jclMax\database_enabler.jar，然后单击**打开**。

h. 回到 “新建项目” 窗口的 “Java 设置” 页上，单击**完成**。

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』

将 DB2eAppl.java 导入到非 Palm OS 目标的 WSDD 中

此任务是 “在非 Palm OS 目标上编译和运行样本 Java 应用程序” 这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

要将 DB2eAppl.java 导入到非 Palm OS 目标的 WSDD 中:

1. 在 “Java 透视图” 的 “程序包资源管理器” 中，展开 DB2 Everyplace Sample 项目，选择 src 文件夹，然后单击**文件 —> 导入**。
2. 在 “导入” 窗口的**选择**页上，选择 “文件系统” 作为导入源，然后单击**下一步**。
3. 在 “导入” 窗口的**文件系统**页上，单击**浏览**以获取**目录**字段，浏览至 *<DB2Everyplace>*\Clients\Win32\database\jdbc 文件夹，然后单击**确定**。
4. 回到 “导入” 窗口的**文件系统**页上，在右窗格中选择 **DB2eAppl.java** 复选框，然后单击**完成**。

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』

运行样本 Java 应用程序

本章描述如何运行样本 Java 应用程序。涉及的主题包括:

- 『在 Win32 上运行 DB2eAppl.java』
- 第 96 页的『在 Windows CE 上运行 DB2eAppl.java』
- 第 98 页的『在 QNX Neutrino 或嵌入式 Linux 上运行 DB2eAppl.java』
- 第 99 页的『在 Symbian 上运行 DB2eAppl.java』

在 Win32 上运行 DB2eAppl.java

此任务是 “在非 Palm OS 目标上编译和运行样本 Java 应用程序” 这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

先决条件:

如果尚未设置系统来使用 DB2 Everyplace JDBC 驱动程序:

1. 使用设置命令，在 PATH 系统变量中包括以下目录:
   <DB2Everyplace>\Clients\Win32\database\x86
2. 使用设置命令，在 CLASSPATH 系统变量中包括以下文件:
   <DB2Everyplace>\Clients\Win32\database\jdbc\db2ejdbc.jar

注意: 如果 WSDD 是打开的，则需要重新启动它以使这些更改反映在 WSDD 中。

过程:

要在 Windows 工作站上运行 DB2eAppl.java:

1. 构建 DB2eAppl.java:
   a. 在“Java 透视图”的“程序包资源管理器”视图中，双击 DB2 Everyplace Sample 项目的 wsddbuild.xml 文件。
   b. 在 wsddbuild.xml 的编辑器中，单击**添加构建**。
   c. 在“创建新的 Ant 构建目标”窗口中，单击**浏览**以获取 **Main 类**字段。
   d. 在打开的窗口中，选择 DB2eAppl - （缺省包） - DB2 Everyplace Sample/src，然后单击**完成**。
   e. 返回到“创建新的 Ant 构建目标”窗口中，在平台列表中选择 J9 for Windows X86，保留“构建名”字段中的缺省值，然后单击**下一步**。
   f. 在“Jxelink 选项”页上，保留缺省值并单击**完成**。
   g. 回到 wsddbuild.xml 的编辑器中，选择 smartlink winx86/DB2eAppl，然后单击**执行构建**。
2. 运行 DB2eAppl.java。
   a. 单击**运行 —> 运行**。“启动配置”窗口打开。
   b. 在“启动配置”窗口的左窗格中选择“Java 应用程序”，然后单击**新建**。
   c. 在出现在右窗格中的配置中，在**名称**字段中输入 DB2eAppl Win32。
   d. 在主页上，完成下列步骤:
      - 单击**浏览**以获取**项目**字段。在“项目选择”窗口中，选择 DB2 Everyplace Sample，然后单击**确定**。
      - 单击**搜索**以获取 **Main 类**字段。在“选择 Main 类型”窗口中，选择 DB2eAppl，然后单击**确定**。
   e. 回到“启动配置”窗口中，单击**应用**，然后单击**运行**。应该会在“WSDD 控制台”中见到样本应用程序的输出。

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

在 Windows CE 上运行 DB2eAppl.java

此任务是“在非 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

先决条件:

如果尚未设置系统来使用 DB2 Everyplace JDBC 驱动程序，完成下列步骤:

1. 将下列文件复制至设备上的 \Windows 目录: <DB2Everyplace>\Clients \WinCE \database *proc* *ver* \db2ejdbc.dll<DB2Everyplace>\Clients \WinCE \database \jdbc \db2ejdbc.jar 其中 *proc* 是处理器类型，而 *ver* 是设备上的 Windows CE 操作系统的版本号。
2. 使用 “Windows CE 远程注册表编辑器”，修改设备的注册表，以在设备的类路径上包括下列文件:

   ```
   \Windows\db2ejdbc.jar
   \wsdd\lib\jclMax\database_enabler.jar
   ```

   （假定您在设备的根目录下安装了 J9）。
   或者，可更新由 WSDD 生成的 DB2eAppl 快捷方式以在类路径上包括以上文件:

```
256#"\wsdd\bin\j9.exe" "-Xbootclasspath:\Windows\db2ejdbc.jar;
\wsdd\lib\jclMax\database_enabler.jar;\wsdd\lib\jclMax\classes.zip;
\wsdd\lib\jclMax\locale.zip;\wsdd\lib\charconv.zip" "-jcl:max" "-jxe:\Temp\DB2eAppl.jxe"
```

过程:

要在 Windows CE 设备上运行 DB2eAppl.java:

1. 配置 Windows CE 设备。
   a. 单击**设备 —> 配置**。
   b. 在 “设备配置” 窗口的左窗格中选择 PocketPC Handheld 然后单击**新建**。
   c. 在右边出现的配置中，完成下列步骤:
      1) 在**设备名**字段中，输入 DB2 Everyplace PocketPC Handheld。
      2) 单击**浏览**以获取 **J9 运行时位置**字段。在 “浏览以查找设备上的文件夹” 窗口中，选择 wsdd（假定您在设备的根目录下安装了 J9），然后单击**确定**。
      3) 单击**浏览**以获取**应用程序安装位置**字段。在 “浏览以查找设备上的文件夹” 窗口中，选择 “临时”，然后单击**确定**。
      4) 单击**浏览**以获取**快捷方式安装位置**字段。在 “浏览以查找设备上的文件夹” 窗口中，选择 “临时”，然后单击**确定**。
   d. 回到 “设备配置” 窗口中，单击**应用**，然后单击**确定**。
2. 构建 DB2eAppl.java。
   a. 在 “Java 透视图” 的 “程序包资源管理器” 窗格中，双击 DB2 Everyplace Sample 项目的 wsddbuild.xml 文件。
   b. 在 wsddbuild.xml 的编辑器中，单击**添加构建**。
   c. 在 “创建新的 Ant 构建目标” 窗口中，单击**浏览**以获取 **Main 类**字段。
   d. 在打开的窗口中，选择 DB2eAppl - （缺省包） - DB2 Everyplace Sample/src，然后单击**完成**。
   e. 回到 “创建新的 Ant 构建目标” 窗口中，从**平台**列表中选择 J9 for PocketPC ARM，保留**构建名**字段中的缺省值，然后单击**下一步**。
   f. 在 **Jxelink 选项**页上，保留缺省值并单击**完成**。
   g. 在 wsddbuild.xml 的编辑器中，选择 smartlink ppcarm/DB2eAppl，然后单击**执行构建**。
3. 运行 DB2eAppl.java:
   a. 单击**运行 —> 运行**。 “启动配置” 窗口打开。
   b. 在 “启动配置” 窗口的左窗格中选择 “设备 Java 应用程序”，然后单击**新建**。
   c. 在出现在右窗格中的配置中，在**名称**字段中输入 DB2eAppl WinCE。

d. 在主页上，完成下列步骤:

1) 单击**浏览**以获取**项目**字段。在“项目选择”窗口中，选择 DB2 Everyplace Sample，然后单击**确定**。

2) 单击**搜索**以获取“Java 应用程序”字段。在“选择目标”窗口中，选择 DB2eAppl.jxe（在 wsddbuild.xml 中目标为“smartlink ppcarm/DB2eAppl”），然后单击**完成**。

3) 从**设备或 JRE** 列表中选择 DB2 Everyplace PocketPC Handheld。

e. 回到“启动配置”窗口中，单击**应用**，然后单击**运行**。应该会在设备上的“J9 控制台”中见到样本应用程序的输出。

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

在 QNX Neutrino 或嵌入式 Linux 上运行 DB2eAppl.java

此任务是“在非 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

先决条件:

如果尚未设置系统来使用 DB2 Everyplace JDBC 驱动程序:

1. 使用导出命令，在设备上的 LD_LIBRARY_PATH 系统变量中包括包含适当 libdb2e.so 和 libdb2ejdbc.so 本机库的一个或多个目录。

过程:

要在 QNX Neutrino 或嵌入式 Linux 设备上运行 DB2eAppl.java:

1. 构建 DB2eAppl.java。
   a. 在“Java 透视图”的“程序包资源管理器”窗格中，双击 DB2 Everyplace Sample 项目的 wsddbuild.xml 文件。
   b. 在 wsddbuild.xml 的编辑器中，单击**添加构建**。
   c. 在“创建新的 Ant 构建目标”窗口中，单击**浏览**以获取 main 类。在打开的“选择目标”窗口中，选择 DB2eAppl - （缺省包）- DB2 Everyplace Sample/src，然后单击**完成**。
   d. 回到“创建新的 Ant 构建目标”窗口的“设置构建”页上，从**平台**列表中选择相应平台，保留“构建名”字段中的缺省值，然后单击**下一步**。
   e. 在“Jxelink 选项”页上，保留缺省值并单击**完成**。
   f. 回到 wsddbuild.xml 的编辑器中，选择相应的构建并单击**执行构建**。
2. 运行 DB2eAppl.java。
   a. 将相应的 DB2eAppl.jxe 文件从

      ```
      <WSDD>\workspace\DB2 Everyplace Sample\<target>
      ```

      复制至设备，其中 <target> 表示目标设备和处理器类型。
   b. 通过使用以下命令来启动应用程序:

      ```
      j9 -Xbootclasspath:/wsdd/lib/jclMax/classes.zip:/wsdd/lib/jclMax/database_enabler.jar -cp:
        /DB2e/db2ejdbc.jar:. -jxe:DB2eAppl.jxe
      ```

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

在 Symbian 上运行 DB2eAppl.java

此任务是“在非 Palm OS 目标上编译和运行样本 Java 应用程序”这一主任务的一部分。完成这些步骤后，请返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

过程:

某些 Symbian 设备与 JVM 一起提供。如果想要运行基于文本的 Java 应用程序（例如，样本 Java 程序），需要安装“重定向”（在 Symbian SDK for Java 中以 `Redirect.sis` 形式提供）并在启动基于文本的应用程序之前启动“重定向”应用程序。“重定向”将捕获文本输出。

返回至第 93 页的『在非 Palm OS 目标上编译和运行样本 Java 应用程序』。

第 15 章 样本 Visual Basic 应用程序

本章提供有关样本 Visual Basic 应用程序的信息。涉及的主题包括:

样本 Visual Basic 应用程序的概述

Visual Basic 样本应用程序显示如何使用 Visual Basic 来存取 DB2 Everyplace 数据。您既可以在 Pocket PC（WinCE）操作系统上也可以在 Win32 操作系统上开发具有相同应用程序逻辑和用户界面的应用程序。DB2 Everyplace 附带提供了两个 Visual Basic 样本应用程序。一个用于 Pocket PC（WinCE）操作系统，另一个用于 Win32 操作系统。这两个样本应用程序的应用程序逻辑和用户界面是相同的。在这两个操作系统之间，db2evb.bas 文件（它包含应用程序逻辑）是公共的。有关更多详细信息，请参阅 101。

样本应用程序中包括的文件:

Visual Basic 项目目录（它包含样本应用程序）位于 DB2 Everyplace 安装目录之下。对于 Pocket PC（WinCE），可以在 \db2everyplace\clients**wince**\database\visualbasic 中找到这些文件。对于 Win32 操作系统，可以在 \db2everyplace\clients**win32**\database\visualbasic 中找到这些文件。

Visual Basic 样本应用程序包括下列文件:

db2evb.bas
db2evb.bas 文件包含 Visual Basic 样本应用程序。可以使用该样本应用程序来帮助您编写您自己的 Visual Basic 应用程序。

db2ecli.bas
db2ecli.bas 文件是用于连接至 DB2 Everyplace 数据库的 Visual Basic 接口。它还定义 sqlcli.h、sqlcli1.h、sqlext.h 和 sqlsystm.h 中的各种 DB2 Everyplace 约束。此文件只包含最常用的约束。如果有需要的话，还可以添加 sqlcli.h、sqlcli1.h、sqlext.h 和 sqlsystm.h 中的其它约束。

DB2eForms（扩展名随操作系统的不同而有所变化）
应用程序用户界面文件。

DB2eSample.exe（对于 WinCE，为 DB2eSample.vb）
应用程序可执行文件。

DB2eSample.vbw
应用程序项目文件。

DB2eSample.vbp（对于 WinCE，为 DB2eSample.ebp）
应用程序项目文件。

Visual Basic 示例: db2evb.bas:

样本应用程序（db2evb.bas）中使用的主要步骤包括:

连接至 DB2 Everyplace 数据库。

– 步骤 1: 分配环境句柄。

– 步骤 2: 分配数据库句柄。

– 步骤 3: 连接至数据库。

– 步骤 4: 分配语句句柄。

存取 DB2 Everyplace 数据。

– 步骤 5: 创建表。

– 步骤 6: 插入数据到表中。

– 步骤 7: 从表中检索数据。

终止应用程序。

注: 确保应用程序在退出之前关闭与 DB2 Everyplace 数据库的连接。

已向此示例添加注释来阐明样本应用程序步骤。

```
Option Explicit

Public henv    As Long          ' Environment handle
Public hdbc    As Long          ' Database handle
Public hstmt   As Long          ' Statement handle
Public rc      As Integer       ' Return code

Public dbpath As String        ' filesystem path where DB2e will create tables.
Public userid As String        ' Userid: not used by DB2 Everyplace.
Public pass   As String        ' Password: not used by DB2 Everyplace

'--------------------------------------------------------------------------------
' Function: DB2eTest
'
' Description: Function illustrating how calls to DB2 Everyplace can be made.
'
'--------------------------------------------------------------------------------

Public Function DB2eTest() As Integer
    Dim errmsg As String
    Dim numCols As Integer
    Dim i As Integer
    Dim retLen As Long
    Dim data As String
    Dim crtStmt As String
    Dim insStmt1 As String
    Dim insStmt2 As String
    Dim selStmt As String
    On Error Resume Next 'Important: don't ask me why, but this line is needed
                         'in every function that calls functions from db2e.dll
                         'otherwise visual basic does strange mysterious things.
    '
    dbpath = ""
    userid = ""
    pass = ""
    '
    crtStmt = "CREATE TABLE x(a INT, b TIMESTAMP)"
    insStmt1 = "INSERT INTO x VALUES(1, CURRENT TIMESTAMP)"
    insStmt2 = "INSERT INTO x VALUES(2, CURRENT TIMESTAMP)"
    selStmt = "SELECT * FROM x"
    '
    data = String(80, " ")
    ' Step 1: allocate an environment handle.
    '

    DB2eForm.DB2eText.Text = vbCrLf & vbCrLf & " Allocating an environment handle"

    rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HENV, henv)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If

    '
```

```
' Step 2: allocate database handle
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         " Allocating a database handle"

rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, hdbc)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Step 3: connect to the database
'
DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         "  Connecting to the database"

rc = SQLConnect(hdbc, dbpath, SQL_NTS, userid, SQL_NTS, pass, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Step 4: allocate a statment handle.
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf &
                         "  Allocating a statement handle"

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, hstmt)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf

'
' Now we can use CLI function calls to execute SQL statements.
'
' Step 5: Create a table
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & crtStmt
rc = SQLExecDirect(hstmt, crtStmt, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function
End If

'
' Create the same table again to force an error message and
' see if DB2eError works.
'
'rc = SQLExecDirect(hstmt, "create table p(a int)", SQL_NTS)
'If (rc <> 0) Then
'   testmsg = MsgBox("BLA1", 1, "DB2 Everyplace Visual Basic")
'   rc = DB2eError()
'   testmsg = MsgBox("BLA2", 1, "DB2 Everyplace Visual Basic")
'   rc = DB2eTerminate()
'   testmsg = MsgBox("BLA3", 1, "DB2 Everyplace Visual Basic")
'   Exit Function
'End If
'

'
' Step 6: Insert data into the table.
'

DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & insStmt1
rc = SQLExecDirect(hstmt, insStmt1, SQL_NTS)
If (rc <> 0) Then
    rc = DB2eError()
    rc = DB2eTerminate()
    Exit Function

End If
```

```
    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & insStmt2
    rc = SQLExecDirect(hstmt, insStmt2, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function

    End If

    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf

    '
    ' Step 7: Retrieve data from table.
    '

    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf & "  " & selStmt
                           & vbCrLf

    rc = SQLExecDirect(hstmt, selStmt, SQL_NTS)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If


    rc = SQLNumResultCols(hstmt, numCols)
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If

    Do While (SQLFetch(hstmt) = SQL_SUCCESS)
        For i = 1 To numCols
            rc = SQLGetData(hstmt, i, SQL_C_CHAR, data, 80, retLen)
            DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & "  " & data & vbCrLf
    If (rc <> 0) Then
        rc = DB2eError()
        rc = DB2eTerminate()
        Exit Function
    End If
        Next
    data = String(80, " ")
    DB2eForm.DB2eText.Text = DB2eForm.DB2eText.Text & vbCrLf
    Loop

    '
    ' Step 8: Close connection to DB2e database before application terminates.
    '
        rc = DB2eTerminate()

    DB2eTest = 0
End Function
```

相关任务:

- 第 29 页的『开发 DB2 Everyplace Visual Basic 应用程序』

编译和测试样本 Visual Basic 程序

过程:

要编译和测试 DB2 Everyplace 样本程序:

1. 打开 Visual Basic 项目文件 DB2eSample.vbp（对于 WinCE, 为 DB2eSample.ebp）。.
2. 构建样本程序。
   - 对于 Win32: 选择**文件 –> DB2eSample.exe**。将构建 DB2eSample.exe。
   - 对于 WinCE: 选择**文件 –> DB2eSample.vb**。将构建 DB2eSample.vb。
3. 复制下列文件:
   - 对于 Win32: 将 DB2e.dll（对于 Win32 操作系统）复制到当前项目目录中或环境变量 PATH 中 DB2e.dll 的路径中。

- 对于 WinCE: 将 DB2eSample.vb、DB2e.dll（对于 Pocket PC 操作系统）和 “Visual Basic 运行时”复制到您选择的目录中。

4. 运行 DB2Sample.exe（对于 WinCE, 为 DB2Sample.vb）。

相关概念:

- 第 101 页的『样本 Visual Basic 应用程序的概述』

相关参考:

- 第 175 页的『DB2 CLI 函数摘要』
- 第 30 页的『支持 Visual Basic 接口的操作系统』

第 16 章 样本 JSP 应用程序

下面列出的文件与 <DB2Everyplace>\SDK\JSP\sample\jsp 目录相关。所有样本 JSP 应用程序使用 <DB2Everyplace>\SDK\JSP\sample\data 目录中的 Visiting Nurse 样本数据库。教程在 <DB2Everyplace>\SDK\JSP\doc 目录中。

使用 WebSphere Studio Professional/Entry Edition V4.0 开发的应用程序
Visiting Nurse Schedule

描述 此样本将动态查询 Visiting Nurse 数据库并将结果显示在表中。

开始页 VNSchedule_ws40\scheduleHTMLResults.jsp

教程 ws40.pdf

使用 WebSphere Studio Application Developer V4.0 开发的应用程序
Visiting Nurse Schedule

描述 此样本将动态查询 Visiting Nurse 数据库并将结果显示在表中。此样本应用程序需要 JDBC 2.0 且不能在 Symbian OS V6 上运行。

开始页 VNSchedule_wsad40\scheduleMasterView.jsp

其它文件

VNSchedule_wsad40\web.xml
VNSchedule_wsad40\dbbeans.jar

教程 wsad40.pdf

使用 WebSphere Studio Application Developer V5.0 开发的应用程序
Visiting Nurse Schedule

描述 此样本将动态查询 Visiting Nurse 数据库并将结果显示在表中。

开始页 VNSchedule_wsad50\scheduleMasterView.jsp

其它文件

VNSchedule_wsad50\web.xml
VNSchedule_wsad50\dbbeans.jar
VNSchedule_wsad50\scheduleMasterViewBean.class

教程 wsad50.pdf

在 WebSphere Studio 外部开发的应用程序
Visiting Nurse

描述

有关 Visiting Nurse 样本应用程序的描述，请参阅《*DB2 Everyplace 安装与用户指南*》。

开始页 VisitingNurse\schedule.jsp

其它文件

VisitingNurse\contact.jsp
VisitingNurse\medrecord.jsp
VisitingNurse\person.jsp

相关任务:

- 第 33 页的『在 Windows 工作站上验证 JSP 支持』
- 第 34 页的『在 Windows CE 设备上为 JSP 开发进行设置』

相关概念:

- 第 31 页的『开发 DB2 Everyplace JSP 应用程序』
- 第 32 页的『DB2 Everyplace JSP 支持概述』

第 17 章 样本同步应用程序

本章提供有关样本同步应用程序的信息。涉及的主题包括:

样本 Sync Client C/C++ 应用程序

下列示例举例说明如何使用 DB2 Everyplace Sync Client API 函数的选择号来构建应用程序。可以在 \DB2e\Clients\clientapisample\C_API 中找到更多源代码示例。

```
/*****************************************************************/
/**
 * This function defines the sync listener. See isyncore.h for more
 * information.
 * param: listenerData, your personal data.
 * param: event, event object
 * param: pExtraInfo (reserved)
 * return: integer, when event type is ISCEVTTYPE_Retry:
 *   . ISCRTNCB_ReplyYes    : retry less than 3 times
 *   . ISCRTNCB_ReplyNo     : retry more than or equal to 3 times
 *     when event type is ISCEVTTYPE_Info:
 *   . ISCRTNCB_Done
 *     when event type is ISCEVTTYPE_Query and its event code is ISCEVT_QueLogin:
 *   . ISCRTNCB_Done        : username and password are entered correctly
 *   . ISCRTNCB_Default     : username and password are not entered
 *     others (ISCEVTTYPE_Fatal, ISCEVTTYPE_Error, ISCEVTTYPE_Query,
 *             and ISCEVTTYPE_Conflict)
 *   . ISCRTNCB_Default     : take default action
 **/
static isy_INT32 syncListener(
                 isy_UINT32            listenerData,
       ISCEVT    *event,
       isy_VOID  *pExtraInfo)
{
    // appEventCodeToMessage is some user function to map an event code to
    // some descriptive event message
    char *statusMsg = appEventCodeToMessage(event);
    int   timesRetried;

    switch (event->type) {
        case ISCEVTTYPE_Fatal:
        case ISCEVTTYPE_Error:
                  printf("Error: %s\n", statusMsg);
            return ISCRTNCB_Default ;

        case ISCEVTTYPE_Retry:
            timesRetried = event->retry;
            if (timesRetried >= 3)
                        return ISCRTNCB_ReplyNo;
            else {
                char ans;
                printf("%s [Y/N] ", statusMsg);
                ans = getchar();
                getchar();
```

```
                if(tolower(ans) == 'y')
                        return ISCRTNCB_ReplyYes;
                else
                        return ISCRTNCB_ReplyNo;
            }

          case ISCEVTTYPE_Info:
              switch (event->code) {
                  case ISCEVT_InfSucceeded:
                  case ISCEVT_InfFailed:
                  case ISCEVT_InfCanceled:
                      printf("Conclusion: %s\n", statusMsg);
                       break;
                  case ISCEVT_InfGeneral:
                  case ISCEVT_InfCancelingSync:
                  case ISCEVT_InfPrepMsg:
                  case ISCEVT_InfSendMsg:
                  case ISCEVT_InfWaitMsg:
                  case ISCEVT_InfApplyMsg:
                      printf("Status: %s\n", statusMsg);
                       break;
                  default: // ignore other event code
                       break;
              } // switch (event->code)
                  return ISCRTNCB_Done;

          case ISCEVTTYPE_Query:
              if (event->code == ISCEVT_QueLogin) {
                  ISCLISTENARG *args = event->info;
                  isy_TCHAR *target = args->argv[0];
                  // Just an example, not intended to be free of memory leaks.
                    isy_TCHAR *username = (isy_TCHAR *) calloc(18, sizeof(isy_TCHAR));
                    isy_TCHAR *password = (isy_TCHAR *) calloc(254, sizeof(isy_TCHAR));
                  char c;
                  int  i;

                  printf("Query on target data(%s): %s ...\n", target, statusMsg);
                  // Ask for the username
                  printf("Username: ");
                  for(i = 0; (c = getchar()) != '\n'; i++) username[i] = c;
                  username[i] = '\0';
                  if (i == 0) return ISCRTNCB_Default; // username not entered
                  // Ask for the password
                  printf("Password: ");
                  for(i = 0; (c = getchar()) != '\n'; i++) password[i] = c;
                  password[i] = '\0';
        args->argv[1] = username;
                    args->argv[2] = password;
                  return ISCRTNCB_Done;
              }
              return ISCRTNCB_Default;

          // all other event types, don't care
          default:
              return ISCRTNCB_Default;
      } // switch (event->type)
}

// Sample SyncClient
main()
{
    isy_TCHAR user[]     = isy_T("user1");
    isy_TCHAR password[] = isy_T("password");
    HISCSERV  hServ;
    HISCCONF  hConf;
    HISCENG   hEngine;
    isy_INT32 rc;

    rc = iscConfigOpen(hServ, isy_T(".\isyncPath"), &hConf;);
    rc = iscEngineOpen(hConf, &hEngine;);
    iscEngineSetListener(hEngine, syncListener, NULL);

    iscEngineSyncConfig(hEngine); // get the configuration first
    iscConfigEnableSubsSet(hConf, NULL); // enable all subscription sets
```

```
        rc = iscEngineSync(hEngine); // sync config + subscription sets

    if (rc == ISCRTN_Failed) {
        HISCCSR   hCursor;
        isy_TCHAR  id[ISCLEN_SubsSetID];
        isy_TCHAR  name[ISCLEN_SubsSetName];
        isy_INT32  enabled;

        iscConfigOpenCursor(hConf, &hCursor;);
        while (iscConfigGetNextSubsSet(hConf, hCursor, id, name)
               == ISCRTN_Succeeded) {
            enabled = iscConfigSubsSetIsEnable(hConf, id);
            if (enabled != ISCRTN_True) continue; // forget about those which have
                                                  // been disabled
            rc = iscConfigGetSubsSetStatus(hConf, id);
            if (rc != ISCRTN_Succeeded)
// Then, the application can have some code
// processing the failing subscription sets here.
// To disable the subscription set, call:

                iscConfigDisableSubsSet(hConf, id);
        }
        iscConfigCloseCursor(hConf, hCursor);
        rc = iscEngineSync(hEngine); // sync config + subscription sets
    }
    // close all handles
    iscEngineClose(hEngine);
    iscConfigClose(hConf);
    iscServiceClose(hServ);
} // main
```

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 9 页的『支持的 C/C++ 开发工具』

样本 Java 本机同步应用程序

有一些样本 Java 程序可用于帮助您编写用于 DB2 Everyplace 的 Java 同步应用程序。

有关样本所在位置的信息，请参阅*《DB2 Everyplace 安装与用户指南》*中的『样本应用程序的概述』一节。

样本程序 ISyncSample.java 演示如何为 DB2 Everyplace 本机同步提供程序编码 Sync Client 应用程序。

ISyncSample.java 样本应用程序的主要步骤如下:

步骤 1:

导入 DB2 Everyplace 同步包。

```
import com.ibm.mobileservices.isync.*;
```

```
import com.ibm.mobileservices.isync.event.*;
```

对于基于 JNI 的同步提供程序，导入 com.ibm.mobileservices.isync.db2e.jni.*;
对于基于陷阱的同步提供程序，导入 com.ibm.mobileservices.isync.db2e.sti.*;

步骤 2:

实现 ISyncListener 接口的 eventIssued 方法，以在同步期间获取事件通知。

步骤 3:

获取实例 DB2eISyncProvider

步骤 4:

从提供程序对象获取同步服务的实例

步骤 5:

从服务对象获取配置存储的实例

步骤 6:

从配置存储对象获取同步驱动程序的实例

步骤 7:

注册实现 ISyncListener 接口的应用程序侦听器对象，以在同步期间从同步驱动程序对象获取事件通知

步骤 8:

在所有启用的预订集上执行同步。检查返回码和异常以获取同步的状态。

步骤 9:

关闭并释放由同步提供程序分配的所有资源

```
// Example 1:  ISync Java - Simple API usage
//

 // Step 1:  import the Sync Client Java packages
      //
      import com.ibm.mobileservices.isync.*;
import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.db2e.jni.*;


      // Step 2:  implement the eventIssued() method in the ISyncListener
      //    interface if you are interested in event notification (optional)
      //
      public class ISyncSample implements ISyncListener {

         public ISyncSample () {}

         public int eventIssued(ISyncEvent evt) {

            int evtType = evt.getEventType();

            switch(evtType) {

            // display event status
            case ISync.EVTTYPE_INFO:
            case ISync.EVTTYPE_ERROR:

               System.out.println ("**********************");
               System.out.println ("SubsSet:    " + evt.getSubscriptionSetName() );
               System.out.println ("Subs:       " + evt.getSubscriptionName() );
               System.out.println ("SubsType:   " + evt.getSubscriptionType() );
               System.out.println ("Event Type: " + evtType );
               System.out.println ("Event Code: " + evt.getEventCode() );
               System.out.println ("Progress:   " + evt.getSyncProgress());
               System.out.println ("**********************\n");

               return ISync.RTNCB_DONE;

            case ISync.EVTTYPE_RETRY:
               return ISync.RTNCB_REPLY_YES;

            case ISync.EVTTYPE_CONFLICT:
               return ISync.RTNCB_DONE;

            // ignore other event types
          default:
                     break;
           }

           // let sync engine take default action
           return ISync.RTNCB_DEFAULT ;

      }

      public void runSample(String host, String port,
                            String userID, String passwrd) {

         ISyncProvider provider = null;
```

```
         ISyncService  service  = null;
         ISyncConfigStore config = null;
         ISyncDriver syncer = null;
         String      path = "data";         // a data directory under current dir
         ISyncSubscriptionSet ssArr[] = null;
         int rc = 0;

            try {

            // Step 3:  get an instance DB2eISyncProvider
            //
            provider = DB2eISyncProvider.getInstance();

         // Step 4:  get an instance of synchronization service from the provider
            //
/*
 For the DB2j sync client, the JDBC driver and url are required

 String driver = "com.ibm.db2j.jdbc.DB2jDriver";
 String jdbcUrl = jdbc:db2j:crtlDb;create=true;
*/

if (driver != null)
    userProps.put("target.db.driver", driver);
if (jdbcUrl != null)
    userProps.put("target.db.url", jdbcUrl);

            Properties userProps = new Properties();

    userProps.put("isync.user", user);
    userProps.put("isync.password", password);
    userProps.put("isync.trace", "detailed");

    service = provider.createSyncService(uri, userProps);

         // Step 5:  get an instance of the configuration store
            //
            config = service.getConfigStore(path);

            // Step 6:  get an instance of the sync driver to perform
            //                                  synchronization
           syncer = config.getSyncDriver();

            // Step 7:  set the listener object for event notification from the
                  syncer object
            //          during synchronization (optional)
            syncer.setSyncListener(this);

            // Step 8:  perform synchronization on all enabled subscription sets
            //
            rc = syncer.sync();

            switch (rc) {
               case ISync.RTN_SUCCEEDED:
                  System.out.println("Synchronization succeeded");
                     break;

               case ISync.RTN_CANCELED:
                  System.out.println ("Synchronization canceled");
                     break;

         default:
                  System.out.println ("Synchronization failed");
                     break;
            }

            ssArr = config.getSubscriptionSets();

            for (int i=0; i < ssArr.length; i++ ) {

               System.out.print ("Subscription Set: " +
                                 ssArr[i].getName() + " Status: ");

               switch(ssArr[i].getStatus()) {

                  case ISync.STATUS_READY:
                     System.out.println("READY");
                     break;

                  case ISync.STATUS_COMPLETED:
                     System.out.println ("COMPLETED");
                     break;

                  case ISync.STATUS_CANCELED:
```

```
                System.out.println ("CANCELED");
                break;

        default:
                System.out.println ("FAILED");
                break;
            }
         }
         catch (ISyncException ie) {
            System.out.println("Exception code: " + ie.getCode());
            ie.printStackTrace();
         }
         catch (Exception e) {
            e.printStackTrace();
         }
         finally {
            // Step 9:  close and free all allocated resources
            //

            try {

               if (syncer != null) {
                  syncer.close();
                  syncer = null;
               }

               if (config != null) {
                  config.close();
                  config = null;
               }

               if (service != null) {
                  service.close();
                  service = null;
               }
            }
            catch(ISyncException ie2) {

               System.out.println("Exception code: " + ie2.getCode());
               ie2.printStackTrace();
            }
         }
      }  // end runSample()


      public static void main(String args[]) {

         String host     = "localhost";
         String port     = "8080";
         String userID   = "nurse1";
         String passwrd  = "nurse1";

         ISyncSample isa = new ISyncSample();

         if (args.length > 0) {
         if (args.length == 4)
         {
           host     = args[0];
           port     = args[1];
           userID   = args[2];
           passwrd  = args[3];
         }
  else
           System.out.println("Usage: java ISyncSample [host] [port] " +
                              "[userid] [password]");
      }
         isa.runSample(host, port, userID, passwrd);

      }  // end main()

    }  // end ISyncSample class
```

相关任务:

- 第 23 页的『安装和验证基于陷阱的本机同步提供程序』

相关概念:

- 第 115 页的『样本 Java MIDP 同步应用程序』

相关参考:

• 第 19 页的『IBM Java Sync API』

样本 Java MIDP 同步应用程序

有一些样本 Java 应用程序可帮助您编写用于 DB2 Everyplace 的 Java 同步应用程序。对于 MIDP 同步提供程序，样本位于以下位置:

```
%DSYINSTDIR%/Clients/Midp/samples
```

主要的样本是 com/ibm/mobileservices/demo、VNurse.java 和 NursesAid.jar 下的 Visiting Nurse 应用程序。在同一样本目录下有两个构成简单应用程序的文件。此应用程序不提供“记录存储管理”（RMS）代码或固定的用户界面。这两个文件是:

- ISyncSample.java: 驱动 MIDlet
- ISyncWorker.java: 负责使数据同步的工作程序

有关 ISyncWorker.java 文件的详细信息:

样本程序 SyncWorker.java 演示如何为 DB2 Everyplace MIDP 同步提供程序编码 Sync Client 应用程序。

Java 样本应用程序执行下列步骤:

1. 导入 DB2 Everyplace 同步包。

```
      import com.ibm.mobileservices.isync.*;
import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.midp.*;
```

2. 实现 ISyncListener 接口的 eventIssued 方法，以在同步期间获取事件通知。
3. 获取实例 MIDPISyncProvider
4. 从提供程序对象获取同步服务的实例
5. 从服务对象获取配置存储的实例
6. 从配置存储对象获取同步驱动程序的实例
7. 注册实现 ISyncListener 接口的应用程序侦听器对象，以在同步期间从同步驱动程序对象获取事件通知
8. 在所有启用的预订集上执行同步。检查返回码和异常以获取同步的状态。
9. 关闭并释放由同步提供程序分配的所有资源。

ISyncSample.java 示例:

以下示例包含注释，这些注释涉及先前节中的步骤。

```
// Example 1:  ISync Java - Simple API usage
//

 // Step 1:  import the Sync Client Java packages
      //
      import com.ibm.mobileservices.isync.*;
import com.ibm.mobileservices.isync.event.*;
      import com.ibm.mobileservices.isync.midp.*;



/**
 Supporting class which handles all of the synchronization tasks.
 Called by ISyncSample.
*/

public class SyncWorker extends Thread implements ISyncListener
{
 private ISyncSample midlet;
 private boolean mCancel;
```

```
private ISyncProvider provider;
private ISyncService service;
private ISyncConfigStore config;
private ISyncDriver syncer;
private String eventString;

public SyncWorker(ISyncSample midlet)
{
 this.midlet = midlet;
 mCancel = false;
}

  // Step 2:  implement the eventIssued() method in the ISyncListener interface
  //          if you are interested in event notification (optional)
  //

public int eventIssued(ISyncEvent evt)
{
 int evtType = evt.getEventType();
 int evtCode = evt.getEventCode();
 int evtProg = evt.getSyncProgress();
        String ssName = evt.getSubscriptionSetName();
 Object listenerInfo = evt.getEventInfo();

 Exception e = null;
 ConflictReader cr = null;

 if (listenerInfo instanceof Exception)
  e = (Exception) listenerInfo;
 else if (listenerInfo instanceof ConflictReader)
  cr = (ConflictReader) listenerInfo;

 eventString += evtCode + ":";

 switch(evtType)
 {
  // display event status
   case ISync.EVTTYPE_INFO:

             switch (evtCode)
     {
               case ISync.EVT_INF_SYNCING_SUBS:
                   midlet.updateSyncStat1("Synchronizing " + ssName);
                   midlet.updateSyncStat2(" ");
                     break;
               case ISync.EVT_INF_SYNC_STARTED:
                   midlet.updateSyncStat1("Synchronization started");
                   midlet.updateSyncStat2(" ");
                     break;
               case ISync.EVT_INF_PREP_MSG:
                   midlet.updateSyncStat2("Preparing message...");
                   break;
               case ISync.EVT_INF_SEND_MSG:
                   midlet.updateSyncStat2("Sending message...");
                   break;
               case ISync.EVT_INF_WAIT_MSG:
                  midlet.updateSyncStat2("Awaiting server reply...");
                  break;
               case ISync.EVT_INF_APPLY_MSG:
                   midlet.updateSyncStat2("Applying server message...");
                     break;
             case ISync.EVT_INF_SYNC_CANCELED:
      midlet.updateSyncStat1("Synchronization canceled");
      midlet.updateSyncStat2(" ");
      break;
       case ISync.EVT_INF_SYNC_SUCCEEDED:
      midlet.updateSyncStat1("Synchronization succeeded");
      midlet.updateSyncStat2(" ");
      break;
             case ISync.EVT_INF_SYNC_FAILED:
      midlet.updateSyncStat1("Synchronization failed");
      midlet.updateSyncStat2(" ");
      break;
       default:
                   break;
          }

  return ISync.RTNCB_DONE;

  case ISync.EVTTYPE_ERROR:
   midlet.updateSyncStat2("Error: " + evtCode);
   return ISync.RTNCB_DONE;

  case ISync.EVTTYPE_RETRY:
```

```
   midlet.updateSyncStat2("Retry: " + evtCode);
   return ISync.RTNCB_REPLY_YES;

  case ISync.EVTTYPE_CONFLICT:
  if (evtCode == ISync.EVT_CFT_REJECT)
  {
   String tabName = evt.getSubscriptionName();
                    midlet.updateSyncStat2("Conflict: " + tabName);

   /*
    Application needs to do the right thing with conflictRow.
   */

   // System.out.println("Conflict table " + tabName
    // + " row: " + conflictRow);
  }
  return ISync.RTNCB_DONE;

  // ignore other event types
  default:
   break;
  }

  // let sync engine take default action
 return ISync.RTNCB_DEFAULT ;

}  // end of eventIssued()
/*
 Synchronization is implemented in a thread to allow the
 user to cancel the request which single-threaded, might
 be hung on in IO request
*/

public void run()
{
 sync();
}

public void cancel()
{
 try
 {
  if (syncer != null)
   syncer.cancelSync();
 }
 catch (ISyncException iex)
 {}

 mCancel = true;
}

private void sync()
{
 try
 {
  eventString = " ";

  String user = "nurse1";
  String password = "nurse1";
  String host = "localhost";
  String port = "9080";

  /*
   If jad file has values, use them, see
   DeployManifest.java in tools.

   In the Sun WirelessToolkit, under Settings, you can enter
   values in the User Defined tab.
  */

  String x = midlet.getAppProperty("Db2eSyncUserName");
  if (x != null)
   user = x;
  x = midlet.getAppProperty("Db2eSyncPassword");
  if (x != null)
   password = x;
  x = midlet.getAppProperty("Db2eSyncHost");
  if (x != null)
   host = x;
  x = midlet.getAppProperty("Db2eSyncPort");
  if (x != null)
   port = x;
  midlet.appendForm(host + ":" + port + " " + user + "/" +
          password);
```

```
        // Step 3:  get an instance MIDPISyncProvider
        //
  provider = MIDPISyncProvider.getInstance();

        // Step 4:  get an instance of synchronization service from the provider
        //
   Hashtable ht = new Hashtable();
   ht.put("isync.user", userName);
   ht.put("isync.password", password);
   ht.put("isync.trace", "detailed");

   service = provider.createSyncService(URI, ht);

        // Step 5:  get an instance of the configuration store
        //
  config = service.getConfigStore(null);

        // Step 6:  get an instance of the sync driver to perform synchronization
        //
  syncer = config.getSyncDriver();

        // Step 7:  set the listener object for event notification
        //     from the syncer object during synchronization
        //
  syncer.setSyncListener(this);

          // Step 8:  perform synchronization on all enabled subscription sets
          //

  int rc = syncer.sync();

  switch (rc)
  {
    case ISync.RTN_SUCCEEDED:
   midlet.reportSyncStatus("Synchronization succeeded "
    + eventString);
   break;

    case ISync.RTN_CANCELED:
   midlet.reportSyncStatus("Synchronization canceled "
     + eventString);
   break;

    default:
   midlet.reportSyncStatus("Synchronization failed "
     + eventString);
   break;
  }
 //  Step 9: Close all resources
 //

  close();
 }
 catch (ISyncException iex)
 {
  midlet.reportSyncStatus("Exception Code: "
   + iex.getCode() + ", Event codes: " + eventString);
 }
 catch (Exception e)
 {
  midlet.reportSyncStatus(e.toString());
 }
 finally
 {
  mCancel = false;
 }
}

private void close() throws ISyncException
{
 if (syncer != null)
 {
  syncer.close();
  syncer = null;
 }
 if (config != null)
 {
  config.close();
  config = null;
 }

 if (service != null)
```

```
  {
   service.close();
   service = null;
  }

  provider = null;
 }
}
```

相关任务:

- 『使用 Sun Wireless Toolkit 开发 isync4j for MIDP 应用程序』
- 第 120 页的『使用 ANT 和 Sun Wireless Toolkit Command Line 开发 isync4j for MIDP 应用程序』

相关概念:

- 第 115 页的『样本 Java MIDP 同步应用程序』

使用 Sun Wireless Toolkit 开发 isync4j for MIDP 应用程序

本主题描述如何在 Sun Wireless Toolkit 应用程序中开发 DB2 Everyplace ISYNC4J for MIDP。本节中使用的示例基于 VNurse 样本应用程序。

先决条件:

有关使用 DB2 Everyplace J2ME MIDP 同步提供程序的硬件和软件先决条件，请参阅第 19 页的『DB2 Everyplace 同步提供程序的概述』。

过程:

1. 启动 Wireless Toolkit。从命令行提示符处，切换至安装了 Sun Wireless Toolkit 的 `bin` 目录。输入 **ktoolbar.bat**。

   注意: 建议使用命令提示符代替 Windows **开始**菜单。
2. 为 `isync4j` 样本应用程序创建新项目:
   a. 打开 J2ME Wireless Toolkit
   b. 选择**新建项目**。
   c. 输入项目名称（例如，VNurse）
   d. 输入 MIDlet 类名（例如，com.ibm.mobileservices.demo.VNurse）
   e. 单击**创建项目**。
   f. 将 `ISyncMidp.jar` 文件复制到 J2ME 项目库。例如:

      ```
      c:\>copy %DSYINSTDIR%\Clients\Midp\lib\ISyncMidp.jar \
        j2me_install_dir\apps\VNurse\lib.
      ```

   g. 可选: 如果要在 MIDlet 运行时查看跟踪输出，将 `ISyncMidpDebug.jar` 复制到 *j2me_install_dir* `\apps \VNurse \lib`。

      注: 当构建将安装在电话上的 JAR 文件时不要使用跟踪。产生的 JAR 文件将太大而不能安装。

   h. 可选: 要使用混淆（obfuscation）（以减少代码大小），将 `retroguard.jar` 文件复制到安装了 J2ME 的 bin 目录。
   i. 单击**设置**。“项目设置”窗口打开。单击**用户定义项**选项卡并单击**添加**来输入下列**键**和**值**条目:
      - Db2eSyncPassword, nurse1 [缺省值]

- Db2eSyncUserName, nurse1 [缺省值]
- PacketDownSize, 2800 [缺省值 30000]
- PacketUpSize, 1400 [缺省值 30000]
- Db2eSyncHost, localhost [缺省值]
- Db2eSyncPort, 9080 [缺省值]

Sun Wireless Toolkit 将这些值放置在 .jad 文件中，MIDlet 在运行时读它们的值。

j. DB2 Everyplace 样本应用程序（VNurse）将显示 PNG 图像文件。单击 **MIDlets** 选项卡并选择 **MIDlet-1**。单击**编辑**将 VNurse.png 更改为 ibm.png。您将需要将 ibm.png 从 Midp\samples\images 目录复制到 *J2ME_ install_ dir*\apps\VNurse\res 目录。

3. 将 DB2 Everyplace 样本 Java 文件导入该项目。例如，将目录结构从 %DSYINSTDIR%\Clients \Midp \samples \com 复制到 *J2ME_ install_ dir* \apps \VNurse \src 目录。
4. 构建并运行 VNurse 样本应用程序。从 Sun Wireless Toolkit 窗口，单击**构建**并单击**运行**。

相关任务:

- 『使用 ANT 和 Sun Wireless Toolkit Command Line 开发 isync4j for MIDP 应用程序』

相关概念:

- 第 19 页的『DB2 Everyplace 同步提供程序的概述』

使用 ANT 和 Sun Wireless Toolkit Command Line 开发 isync4j for MIDP 应用程序

本主题描述如何使用 ANT 和 Sun Wireless Toolkit Command Line 开发 DB2 Everyplace ISYNC4J for MIDP。

先决条件:

下载并安装下列软件来使用提供的示例:

- Sun Microsystems JavaTM 2 Platform Micro Edition, Wireless Toolkit
- Apache ANT
- RetroGuard Ofuscator

过程:

1. 可选: 如果想要修改演示，则重新编译它们。

   lib 目录包含预编译的 JAD 和 JAR 文件。提供 build.bat 和 build.xml 脚本来举例说明如何使用 Apache ANT、DeployManifest 工具和 RetroGuard obfuscator。

   a. 将 retroInstallDir\lib\retroguard.jar 添加到 CLASSPATH 变量。在环境中设置下列变量:
      - ANT_HOME - 至 ANT 安装的根目录
      - DB2m_HOME - 至 %DSYINSTDIR%\Clients\Midp 目录
      - J2MEWTK_HOME - 至 Sun Wireless Toolkit 安装的根目录
      - JAVA_HOME - 至 jdk13 或 jdk131（仅）安装的根目录

- JAVA14_HOME – 设置为 jdk14 目录的根目录。

b. 执行 MIDP 客户机目录的根目录中的 build.bat 文件，以便用新的 JAR 和 JAD 文件重新放入 MIDP 下的 lib 目录。每个用户和设备标识配置都有一个 JAR 文件和若干 JAD 文件。

您将发现几个新的 build*classes 目录，它们用于预验证和混淆（obfuscation）。每个用户和设备标识配置都有一个 JAR 文件和若干 JAD 文件。查看 JAD 文件来了解如何设置用户标识、密码和设备标识以及如何将它们传送到 MIDLet 应用程序。

c. DeployManifest 类包含在 lib\FilterServlet*.jar 中，从 build.xml 文件调用它。使用此类来生成“JAR 清单”文件和 JAD 文件。生成文件时使用下列语法。

要生成“清单”文件:

```
java DeployManifest -m <midletName> <className> <imageFileName> <outputfileName>
```

要生成 JAD 文件:

```
java DeployManifest -j <midletJarName> -U <uploadMaxPacket> -D \
  <downloadMaxPacket> -n <numClients> <JadBaseName> <outputFileName>
```

d. Apache ANT 从 build.xml 内部调用 DeployManifest 类。编辑 build.xml 文件内的 setJad 条目，以永久地更改用户标识、密码或其它属性。缺省值是 *nurse1* 和 *nurse1*。

2. 运行 insync4j 应用程序。

DB2 Everyplace 安装会创建具有预订集、用户和组的 Vnurse 数据库。

a. 选择**开始 –> DB2 Everyplace –> 启动 MDAC** 并验证名为 *nurse1* 的用户是否存在。此用户的密码设置为 *nurse1*。可以使用此用户名，也可以编辑正在传送至运行脚本的 lib\<midlet>.jad 文件。注意，每次编译更改时，会覆盖 JAD 文件。请参阅 samples\DeployManifest.java 以永久地更改用户和密码。

b.

必须使用 Tomcat 或 Websphere 版本 4.0 或更高版本启动 Sync Server。来自 MIDP 电话的“HTTP 连接”使用 HTTP 编码传送，它需要支持 HTTP Servlet 2.3 规范和 HTTP 1.1 的 servlet 引擎。

c. 执行 Midp\bin 目录中的批处理文件，从 Midp\lib 目录中将某个 JAD 文件的名称传送给它:

- 要运行演示程序的非调试版本，输入:

```
run VNurse
```

- 要使用“nurse3”作为设备 # 213 的用户标识和密码运行调试版本，输入:

```
run VNurseDebug3
```

J2ME MIDP Sync Client 使用在 com.ibm.mobileservices.isync.midp、com.ibm.mobileservices.isync 和 com.ibm.mobileservices.isync.event 包内定义的接口和类。

相关任务:

- 第 119 页的『使用 Sun Wireless Toolkit 开发 isync4j for MIDP 应用程序』

相关参考:

- 第 19 页的『IBM Java Sync API』
- 第 19 页的『支持 Java Sync API 的操作系统』

编译和运行 GoISyncConsole 样本 Java 同步应用程序

GoISyncConsole 是一个 Java 样本应用程序，用来演示 DB2 Everyplace Sync Client Java API 的使用。

GoISyncConsole 的文件内容:

- `GoISyncConsole.java`
- `GoISyncConstants.java`
- `GoISyncListener.java`
- `isyncdb2.properties`
- `isyncdb2e.properties`
- `isyncdb2j.properties`

先决条件:

- 安装并配置了 DB2 Everyplace Sync Server。
- 在设备上安装了 Sync Client 二进制文件。它们位于 `Clients\`*`platform`*`\sync` 目录中。
- 在设备上安装了 Cloudscape（如果在使用 Cloudscape 客户机的话）。

过程:

1. 编译 GoISyncConsole 应用程序:

   这需要 `isync4j.jar` 文件，它是 Sync Client 二进制文件之一。

   a. 打开命令提示符。

   b. 输入以下命令:

   ```
   javac -classpath isync4j.jar *.java
   ```

2. 设置环境:

   必须设置路径环境才能定位 Sync Client 二进制文件。

   - 对于 Win32: 将 PATH 变量设置为包括 Sync Client 二进制文件所在的文件夹。
   - 对于 Linux 或 Neutrino: 导出 LD_LIBRARY_PATH 以包括 Sync Client 二进制文件所在的文件夹。

3. 运行该样本:

   GoISyncConsole 可与 C 客户机或 Java DB2j 客户机配合使用。属性文件用于确定使用哪个客户机。已经提供了 DB2e 和 DB2j 的样本属性文件。

   - 要使用 C 客户机，通过输入以下命令使用提供的 `isyncdb2e.properties` 文件:

     ```
     java -classpath isync4j.jar;. GoISyncConsole isyncdb2e.properties
     ```

   - 要使用 Java DB2j 客户机，加入 DB2j Sync Client JAR 和 Cloudscape JAR 文件，并通过输入以下命令来进入 `isyncdb2j.properties` 文件（如果 Cloudscape 安装目录不同，则修改斜体文本部分）:

```
java -classpath c:\cloudscape_5.1\lib\db2j.jar;
db2jisync.jar GoISyncConsole isyncdb2j.properties
```

   应用程序以包含下列选项的文本菜单开始:

   - （1）执行同步
   - （2）启用、禁用或复位预订集
   - （3）更改服务器设置

- （4）查看日志
- （5）关于 Sync Client
- （6）退出

4. 指定选项（3）以配置服务器设置。此选项将允许您指定 Sync Server 的 IP 地址，同步用户名和密码以及其它选项。
5. 指定选项（1）以执行同步。

   GoISyncConsole 应用程序创建称为 ISync.properties 的另一属性文件以保存首选项。如果更改 isyncdb2e.properties 或 isyncdb2j.properties 中的属性，则在重新运行 GoISyncConsole 之前应删除 ISync.properties 以确保新更改生效。

相关概念:

- 第 19 页的『DB2 Everyplace 同步提供程序的概述』

第 4 部分 参考

第 18 章 应用程序编程接口（API）

DB2 Everyplace SQL 语句支持

本章包含使用 DB2 Everyplace 支持的 SQL 语句的受支持语法图、语义描述、规则和示例。涉及的主题包括:

- 『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』
- 第 165 页的『SQL 报告的 SQLState 消息』
- 第 168 页的『CLI 报告的 SQLState 消息』

DB2 Everyplace SQL 语句支持的概述

可以通过使用命令行处理器（CLP）以交互方式从移动式设备发出受支持的可执行 SQL 语句，也可以在应用程序中使用这些语句来存取 DB2 Everyplace 数据库中的数据。表 15 列出了 DB2 Everyplace 支持的 SQL 语句。

表 15. 支持的 SQL 语句

SQL 语句	函数
CALL	使用 DB2 Everyplace Sync Server 远程查询和存储过程适配器（AgentAdapter）调用远程存储过程
CREATE INDEX	创建索引。
CREATE TABLE	定义表。
DELETE	从表中删除一行或多行。
DROP	从数据库中删除表或索引。
EXPLAIN	获取关于 SELECT 语句的存取路径选择的信息。
GRANT	将加密特权授予用户。
INSERT	将一行或多行插入表。
REORG TABLE	除去或减少与指定的表相关联的浪费了的存储器。
REVOKE	撤销用户的加密特权。
SELECT	指定从一个或多个表查询的结果表。
UPDATE	更新一个或多个表行中一列或多列的值。

第 165 页的『SQL 报告的 SQLState 消息』列出了 DB2 Everyplace SQL 引擎报告的所有 SQLSTATE。

SQL 语句的长度不能超过 64000 个字符。

该目录包含下列由 DB2 Everyplace 管理的 DB2 Everyplace 系统表: DB2eSYSTABLES、DB2eSYSRELS 和 DB2eSYSCOLUMNS。

相关参考:

- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

CALL

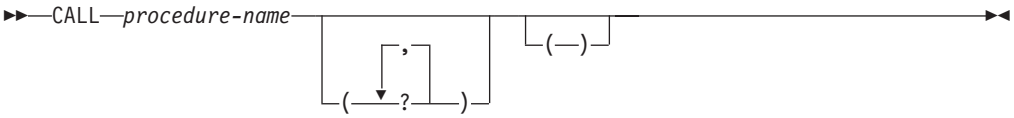
调用使用 DB2 Everyplace Sync Server 的“远程查询和存储过程适配器”定义的存储过程。例如，在远程数据库位置执行了存储过程，并将数据返回至 DB2 Everyplace 客户机应用程序。

使用 SQL CALL 语句的程序被设计为分两部分运行，一部分在客户机上运行，而另一部分在服务器上运行。

调用:

远程存储过程是通过将以下 CALL 语句语法传送至后跟 `SQLExecute()` 的 `SQLPrepare()` 从 DB2 Everyplace 应用程序中调用的。

语法:



描述:

procedure-name
标识要在远程服务器上调用的过程。必须在当前 Sync Server 上的 AgentAdapter 预订中定义标识的过程。

? CALL 语句语法图中的 *?* 指示对应于存储过程的自变量的参数标记。所有自变量必须使用参数标记来传送。

规则:

无

注意事项:

CALL 语句使用 DB2 Everyplace Sync Server 附带包括的远程查询和存储过程适配器。DB2 Everyplace Sync Server 是在 DB2 Everyplace 应用程序中使用 CALL 语句所必需的。DB2 Everyplace 不支持本地存储过程。

有关附加信息，请参阅*《DB2 Everyplace Sync Server 管理指南》*的『数据源』一节。

示例:

*《DB2 Everyplace Sync Server 管理指南》*中提供了如何使用 CALL 语句和远程查询和存储过程适配器的完整示例。以下样本仅显示样本应用程序中的 CALL 语句的编码。

存储过程 `MYPROC()` 是在数据库 `mysample` 的源服务器上定义的。AgentAdapter 预订是在带有下列属性的 DB2 Everyplace Sync Server 上定义的:

```
User ID:  db2admin
Password: db2admin
Other:    dbname=mysample;procname= db2e.MYPROC
```

使用 CALL 语句的样本程序:

```
int main(int argc, char * argv[])
{
  SQLHENV  henv;
  SQLHDBC  hdbc;
  SQLHSTMT hstmt;
  SQLRETURN rc;
  SQLCHAR  strSQL[] = "CALL db2e.MYPROC(?,?,?,?,?)";
  int   nInd4, nInd5;
  int   nSaving = 0, nChecking =0 ;
  int   nCmd =0, nAmount=0;
  SQLCHAR  strConnect[254];

 //*************************************************************
 //* Check input parameters
 //*************************************************************
 if ( argc < 4 ){
   printf("\nUsage : myClient AccountName Cmd Amount");
   printf("\n     cmd 1 : query balance");
   printf("\n     cmd 2 : Transfer from Saving to Checking");
   printf("\n     cmd 3 : Trnasfer from Checking to Saving");
   return (99);
 }
 nCmd = atoi(argv[2]);
 nAmount = atoi(argv[3]);

 //*************************************************************
 //* Allocate handles
 //*************************************************************
 rc = SQLAllocHandle( SQL_HANDLE_ENV,
      SQL_NULL_HANDLE,
      &henv; //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_DBC,
      henv,
      &hdbc); //checkerror
 if (argc == 5){
 strcpy(strConnect,"http://");
 strcat(strConnect,argv[4]);
   strcat(strConnect,"/servlet/com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample");
 }else{
  strcpy(strConnect,
             "http://127.0.0.1:8080/db2e/servlet/

              com.ibm.mobileservices.adapter.agent.AgentServlet?DB=mysample");
 }

 //*************************************************************
 //* Connect to remote database
 //*************************************************************
 rc = SQLConnect(hdbc,
  strConnect,
  SQL_NTS,
  "userex", SQL_NTS,
  "userex", SQL_NTS  ); //checkerror
 rc = SQLAllocHandle( SQL_HANDLE_STMT,
      hdbc,
      &hstmt); //checkerror
 //*************************************************************
 //* Prepare, Bind , and Execute the statement
 //*************************************************************
 rc = SQLPrepare(hstmt,strSQL, SQL_NTS); //checkerror
 rc = SQLBindParameter(hstmt,
   1,
   SQL_PARAM_INPUT,
   SQL_C_CHAR,
   SQL_CHAR,
   0,
   0,
   (SQLPOINTER)argv[1],
   0,
   NULL ); //checkerror
 rc = SQLBindParameter(hstmt,
   2,
   SQL_PARAM_INPUT,
   SQL_C_LONG,
   SQL_INTEGER,
   0,
   0,
```

```
  (SQLPOINTER)&nCmd,
  sizeof(int),
  NULL); //checkerror
rc = SQLBindParameter(hstmt,
  3,
  SQL_PARAM_INPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nAmount,
  sizeof(int),
  NULL ); //checkerror
rc = SQLBindParameter(hstmt,
  4,
  SQL_PARAM_OUTPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nSaving,
  sizeof(int),
  &nInd4 ); //checkerror
rc = SQLBindParameter(hstmt,
  5,
  SQL_PARAM_OUTPUT,
  SQL_C_LONG,
  SQL_INTEGER,
  0,
  0,
  (SQLPOINTER)&nChecking,
  sizeof(int),
  &nInd5 ); //checkerror
rc = SQLExecute(hstmt); //checkerror
//*************************************************************
//* Print the balance
//*************************************************************
printf("\nSaving = %d",nSaving);
printf("\nChecking = %d",nChecking);

SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
return 0;
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

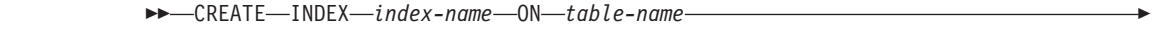
CREATE INDEX

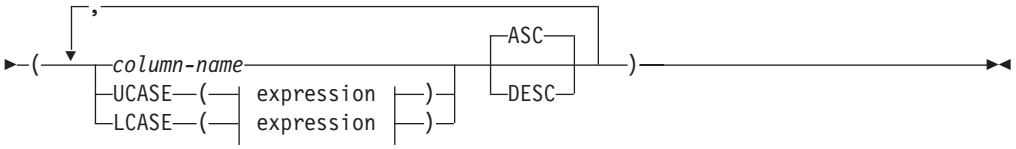
CREATE INDEX 语句用来为 DB2 Everyplace 表创建索引。

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:





描述:

INDEX *index-name*
命名索引。

ON *table-name*
table-name 命名要对其创建索引的表。

column-name
对于索引，列名标识要作为索引键一部分的列。

每个列名都必须是标识一个表列的未限定名。使用 8 列或更少的列；不能重复列名（SQLSTATE 42711）。

每个指定列的长度一定不能超过 1024 字节。

ASC 按列以升序顺序排列索引条目。这是缺省值。

DESC 按列以降序顺序排列索引条目。

LCASE / UCASE

LCASE 或 LOWER 函数返回一个字符串，其中所有 SBCS 字符都已转换为小写字符。即，字符 A 至 Z 将转换为字符 a 至 z，而带有区分标记的字符将转换为其小写等效项（如果它们存在的话）。

自变量必须是其值为 CHAR 或 VARCHAR 数据类型的表达式。

该函数的结果的数据类型和长度属性与自变量的数据类型和长度属性相同。如果自变量可以为空，则结果可以为空；如果自变量为空，则结果为空值。

确保 EMPLOYEE 表中的列 JOB 的值中的字符将以小写字符的形式返回。例如:

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

规则:

- 对于没有主键的表，最多可创建 15 个索引。对于有主键的表，最多可创建 14 个索引。
- 如果尝试创建与现有索引匹配的索引，则 CREATE INDEX 语句将失败。在下列情况下，两个索引描述被认为是重复的:
  - 索引中的列集及其顺序与现有索引的列集和顺序相同。
  - 排序属性是相同的。
- 不能在 CREATE INDEX 语句中使用具有 BLOB 数据类型的列。

注意事项:

- CREATE INDEX 语句可包含最多 8 列。
- DB2 Everyplace 支持索引的双向扫描。尽管下面两个索引的定义不同，但它们的作用是相同的。

```
CREATE INDEX IDX1 ON EMPLOYEE (JOB ASC)
CREATE INDEX IDX1 ON EMPLOYEE (JOB DESC)
```

通常，应在不指定排序方向的情况下创建索引。通常，索引越少，索引维护的成本也就越低。

- DB2 Everyplace 支持索引的前缀扫描。考虑以下示例。创建了以下索引。

```
CREATE INDEX J1 ON T (A, B, C, D, E, F, G, K)
```

不需要为 T (A,B,C,D) 创建另一索引。

- 如果该表未包含数据，则 CREATE INDEX 创建索引的描述；在对表插入数据时创建索引条目。
- 要为脏位索引创建索引，请使用以下示例:

```
CREATE INDEX <index name>
  ON <table name>
  ($dirty)
```

有关脏位的更多信息，请参阅 246。

示例:

为 EMPLOYEE 表创建名为 JOB_BY_DPT 的索引。按每个部门（WORKDEPT）中的职位（JOB）以升序顺序排列索引条目。

```
CREATE INDEX JOB_BY_DPT
ON EMPLOYEE (WORKDEPT, JOB)
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

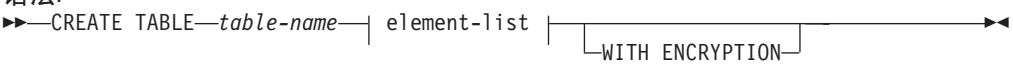
CREATE TABLE

CREATE TABLE 语句定义表。此定义必须包括表名和表列的名称和属性。此定义还可以包括表的其它属性，如它的主键。

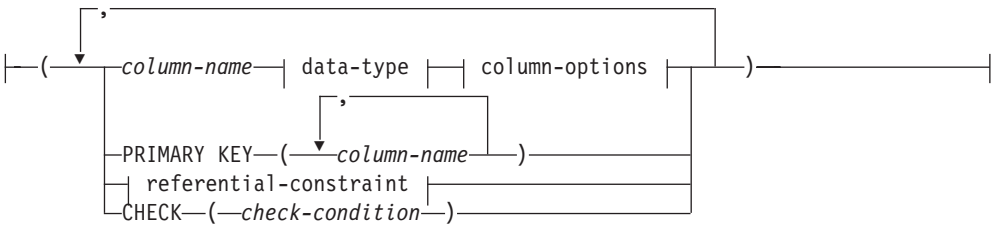
调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

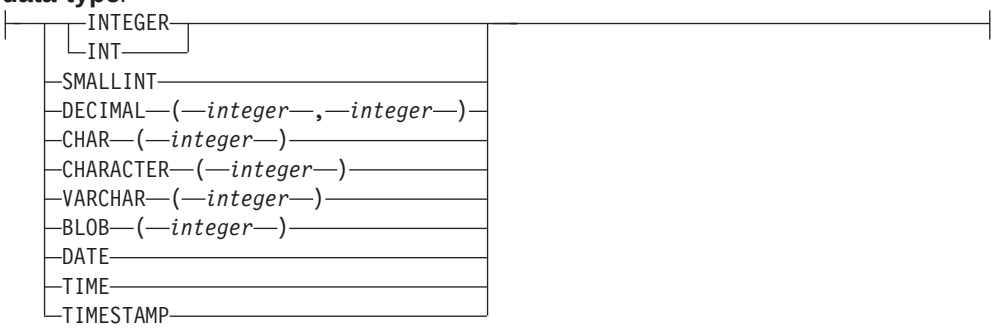
语法:



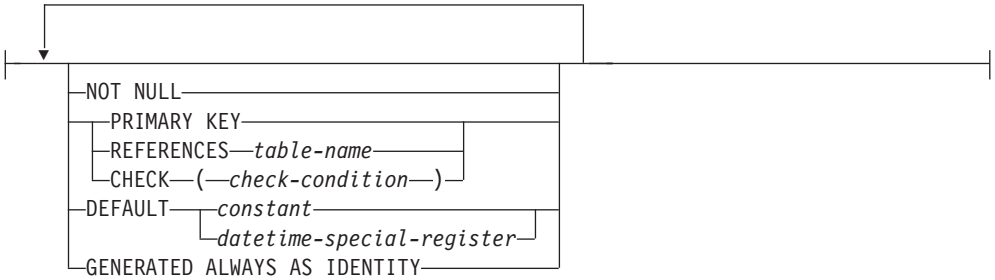
element-list:



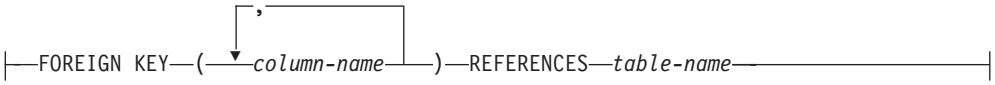
data-type:



column-options:



referential-constraint:



描述:

table-name

给表命名。名称最长可有 18 字节。此名称不得标识目录中的表。此名称对移动式设备必须是唯一的。

在存储在目录中之前，表名被转换成大写。可以使用定界标识（带双引号）来防止这样的转换。当表名包含空白或特殊字符时，必须使用定界标识。

表名可以包含“双字节字符集”字符。

限制: 与按用户名创建和命名的表相对应的系统创建的数据文件不区分大写和小写字符。例如，名为 TB 的表的数据文件命名为 DSY_TB。名为“tb”的表的数据文件也是 DSY_TB。因此，要确保数据完整性，强烈建议您不要使用与现有的表名称除字符大小写外完全相同的一系列字符来命名表。

WITH ENCRYPTION

创建加密的用户表。要对表进行加密，必须已认证并连接。必须显示地获得加密授权。（有关更多信息，请参阅第 143 页的『GRANT』。）

用户表仅在创建时才可对它加密。创建表后，除删除表以外，不能添加或除去加密。

column-name

命名表列。名称最长可有 18 字节。名称不能被限定，且同一名称不能用于多个表列。

在存储在目录中之前，列名被转换成大写。可以使用定界标识（带双引号）来防止这样的转换。当列名包含空白或特殊字符时，还必须使用定界标识。

列名可以包含 DBCS 字符。

data-type
是以下列表中的其中一种类型。使用:

INTEGER 或 **INT**
表示四字节带符号整数，其范围在 2147483647 到 -2147483648 之间。

SMALLINT
表示双字节带符号整数，其范围在 -32768 到 32767 之间。

DECIMAL*(precision-integer, scale-integer)*
表示十进制数。第一个整数是数字的精度（即总位数）；其范围在 1 到 31 之间。第二个整数是数字的小数位；即小数点右边的位数；其范围在 0 到该数字的精度之间。

CHAR*(integer)*
表示长度为 *integer* 的定长字符串，其范围在 1 到 32767 之间。

CHARACTER*(integer)*
表示长度为 *integer* 的定长字符串，其范围在 1 到 32767 之间。

VARCHAR*(integer)*
表示最大长度为 *integer* 的变长字符串，其范围在 1 到 32767 之间。

BLOB*(integer)*
表示具有指定最大长度（以字节计）的二进制大对象字符串。

长度的范围可在 1 个字节到 32767 个字节之间。

integer 是最大长度。

DATE
表示日期。输入值可为下列其中一种格式: MM/DD/YYYY、YYYY-MM-DD 或 DD.MM.YYYY。日期值只能以 ISO 格式（YYYY-MM-DD）显示出来。

专用寄存器 CURRENT DATE 也生成 ISO 格式的当前日期。

TIME
表示时间。输入值可为下列其中一种格式: HH:MM AM（或 PM）、HH:MM:SS、HH.MM AM（或 PM）或 HH.MM.SS。SS（秒数）可选择使用 HH:MM:SS 格式或 HH.MM.SS 格式。时间值只能以 ISO 格式（即 HH:MM:SS）显示出来。

专用寄存器 CURRENT TIME 也生成 ISO 格式的当前时间。

TIMESTAMP
表示时间戳记。输入值必须是以下格式: YYYY-MM-DD-HH.MM.SS.ZZZZZZ。时间戳记值可以以下格式打显示出来: YYYY-MM-DD-HH.MM.SS.ZZZZZZ。

专用寄存器 CURRENT TIMESTAMP 也生成当前时间戳记。

column-options
定义与表列相关的附加选项。

NOT NULL

防止该列包含空值。

如果未指定 NOT NULL，则该列可以包含空值，且其缺省值是空值或 DEFAULT 子句提供的值。

PRIMARY KEY

这提供定义由单一列组成的主键的快捷方法。因此，如果列 C 的定义中指定了 PRIMARY KEY，则效果与 PRIMARY KEY(C) 子句被指定为单独子句相同。

请参阅 136 页上 PRIMARY KEY 的描述。

REFERENCES *table-name*

请参阅 136 页上 REFERENCES 的描述。

CHECK (*check-condition***)**

请参阅 136 页上 CHECK 的描述。

DEFAULT

在 INSERT 语句中未提供值的情况下提供缺省值。

在列定义中省略 DEFAULT 会导致使用空值作为列的缺省值。如果这样的列被定义为 NOT NULL，则该列不具有有效的缺省值。

constant

指定常量作为列的缺省值。指定的常量:

- 必须表示一个可赋给列的值。
- 不得具有超出列数据类型的小数位的非零小数位，例如，1.234 不能是 DECIMAL(5,2) 列的缺省值（如果该常量是一个十进制常量的话）。

datetime-special-register

指定执行 INSERT 时日期时间专用寄存器（CURRENT DATE、CURRENT TIME 或 CURRENT TIMESTAMP）的值作为列的缺省值。该列的数据类型必须是与指定的专用寄存器相对应的数据类型（例如，当指定 CURRENT DATE 时，数据类型必须是 DATE）。

GENERATED ALWAYS AS IDENTITY

在创建表时，用户可将列指定为 “GENERATED ALWAYS AS IDENTITY”。随后，每当用户执行 INSERT 或带有子选择的 INSERT 时，DB2 Everyplace 都会生成此列的值。此列必须为数字类型（INTEGER、SMALLINT 或 DECIMAL 类型），而且 DB2 Everyplace 将自动生成唯一序列号，从 1 开始，每次增加 1。

IDENTITY 列的生成值从 1 开始，每次将一行插入到表中都会增加 1。因此，唯一性得到了保证，尽管 DB2 Everyplace 不会自动对 IDENTITY 列创建索引。如果想要具有 IDENTITY 列的索引，必须显式创建索引或将该列指定为 PRIMARY KEY。当 IDENTITY 列的值的范围耗尽（到达最大值）时，继续使用 INSERT 语句将导致错误（SQLSTATE 23522）。INT 和 SMALLINT 类型的 IDENTITY 列的最大值为是这两种类型允许的最大值。DECIMAL 类型的 IDENTITY 列的最大值由下列各项确定: （1）数据类型的定义（精度和标度）以及（2）IDENTITY 列允许的最大值: 2.15* (10∧18)（19 个小数位数）。（1）和（2）中较小的那一项即范围限制。对于 DECIMAL 类型的 IDENTITY 列，该值的分数部分始终为 0，而整数部分每次增加 1。

只能对其数据类型为下列三种数字类型之一的列定义 IDENTITY 规范: INT、SMAIINT 和 DECIMAL。否则，将生成错误（SQLSTATE 42815）。每个表至多有一个 IDENTITY 列（否则生成错误 SQLSTATE 428C1）。用户既不能在 INSERT 语句中提供 IDENTITY 列的值（必须缺省为 DB2 Everyplace 系统生成的值），也不能更新 IDENTITY 列。

PRIMARY KEY (*column-name***, ...)**

定义由标识的列组成的主键。不得多次指定此子句，且标识的列必须被定义为 NOT NULL。每个列名都必须标识一个表列，且不得多次标识同一个列。

标识的列数不得超过 8。

将对指定列自动创建唯一索引。

只能对表定义一个主键。

每个指定列的长度属性一定不能超过 1024 字节。

referential-constraint

定义引用约束。

FOREIGN KEY (*column-name***, ...)**

定义具有指定约束名的引用约束。

让 T1 指示语句的对象表。引用约束的外键由标识的列组成。列名列表中的每个名称都必须标识 T1 的一个列，且不得多次标识同一个列。标识的列数不得超过 8。DB2 Everyplace 不强制使用外键。

REFERENCES *table-name*

REFERENCES 子句中指定的表必须标识目录中描述的基本表，但不得标识目录表。

如果引用约束的外键与先前指定的引用约束的外键表相同，则该引用约束是重复的。

在下面的讨论中，让 T2 指示标识的父表，而让 T1 指示正在创建的表。

指定的外键的列数必须与 T2 的父键的列数相同，且外键的第 n 列的描述必须与该父键的第 n 列的描述相匹配。根据此规则，日期时间列不被认为与字符串列相匹配。DB2 Everyplace 不强制使用外键。

CHECK (*check-condition***)**

定义检查约束。*check-condition* 是搜索条件。列引用必须是正在创建的表的一个列。插入或更新到表中的值必须满足所有检查约束。

如果检查约束被指定为列定义的一部分，则只能对同一个列进行列引用。作为表定义一部分指定的检查约束可具有标识先前在 CREATE TABLE 语句中定义的列的列引用。不检查检查约束的不一致性、重复条件或等效条件。因此，可以定义矛盾的或冗余的检查约束。

可以指定检查条件 “IS NOT NULL”，然而，建议使用列的 NOT NULL 属性直接强制使用可空性。例如，如果 salary 设置为 NULL，则接受 CHECK (salary + bonus > 30000)，这是因为 CHECK 约束必须被满足或是未知的，在此情况下，salary 是未知的。然而，CHECK (salary IS NOT NULL) 将被认为是假的，而如果 salary 设置为 NULL，则会发生约束违例。

当在表中插入或更新行时，便强制实施检查约束。

在 CREATE TABLE 语句中定义的所有检查约束都被组合并存储在系统目录中。DB2 Everyplace 将此组合的检查约束限制为不能超过 512 个字节。

规则:

- 一行的实际总字节计数一定不能超过 65 536。

  有关更多信息，请参阅 137。
- 具有 BLOB 数据类型的列不能带有检查、缺省、引用或外键约束（SQLSTATE 42962）。
- 具有 BLOB 数据类型的列不能用于 CREATE TABLE 语句的主键中。

注意事项:

- 表和列应使用大写名称来创建。大小写混合的名称可能会导致在使用某些语言时出错。
- 如果在移动式设备上创建一个新表，则在将移动式设备与服务器同步时，在企业数据库上并不自动创建该表。必须在同步之前在企业数据库上创建该表。
- 数据的字节计数: 以下列表包含按数据类型分类的列的字节计数。这可能会随每个发行版而更改。每个记录还包括有关 NULL 的信息。对于 32 列的每个组，NULL 信息需要 4 个字节。NULL 值仍然使用固定大小的列大小。

数据类型	列字节计数
INTEGER	4
SMALLINT	4
DECIMAL(n, m)	4 – 20
CHAR(n)	n+1
VARCHAR(n)	i+5，其中 i 是实际长度
BLOB	i+4，其中 i 是实际长度
DATE	4
TIME	4
TIMESTAMP	12

示例:

创建具有列名 EMPNO、FIRSTNAME、LASTNAME、DEPT、PHONENO、SALARY 和 HIREDATE 的 EMPLOYEE 表。CHAR 表示该列将包含字符数据。NOT NULL 表示该列不能包含空值。VARCHAR 表示该列将包含变长字符数据。主键由 EMPNO 列组成。

```
CREATE TABLE EMPLOYEE
  (EMPNO       CHAR(3)    PRIMARY KEY,
   FIRSTNAME   VARCHAR(12)   NOT NULL,
   LASTNAME    VARCHAR(15)   NOT NULL,
   DEPT        CHAR(3),
   PHONENO     CHAR(4),
   SALARY      INT,
   HIREDATE    DATE)
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』

- 第 164 页的『SQLState 类代码的摘要』

DELETE

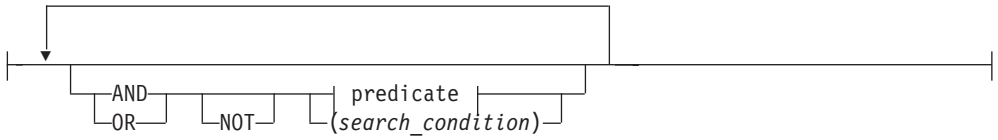
DELETE 语句从表中删除一行或多行。

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

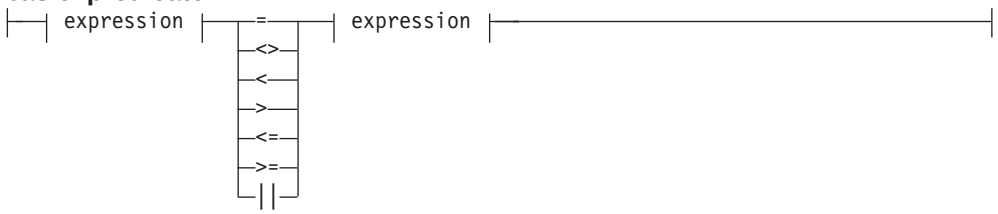
语法:

search_condition:

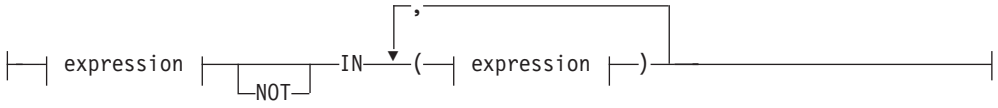


predicate:

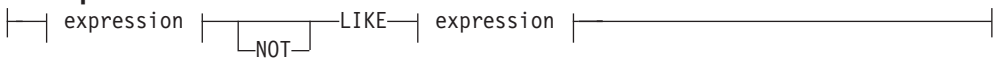
basic predicate:



IN predicate:

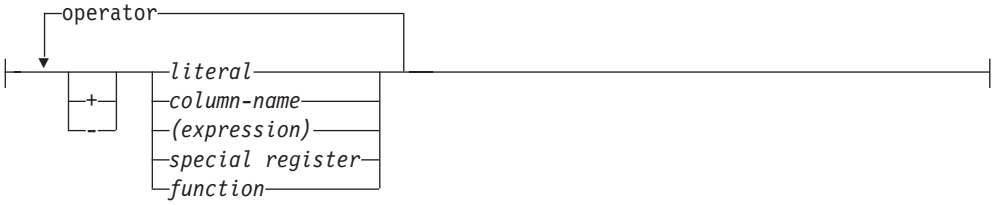


LIKE predicate:

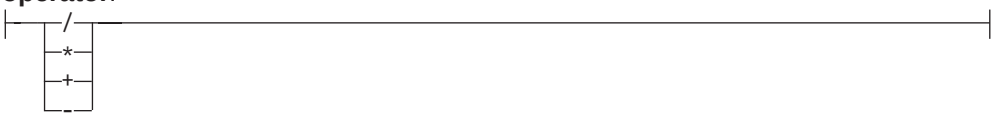


NULL predicate:

expression:



operator:



注:

1 仅 NULL 谓词中允许 BLOB 表达式。

描述:

FROM *table-name*

标识要从中删除行的表。名称必须标识目录中存在的表，但不能用它标识目录表。

WHERE

指定选择要删除的行的条件。可以省略此子句，或指定搜索条件。如果省略此子句，则会删除所有表行。

search_condition

search_condition 指定关于给定行的为真、为假或未知的条件。

search_condition 的结果通过对每个指定谓词的结果应用指定的*逻辑运算符*（AND、OR 或 NOT）而导出。谓词对两个值作比较。如果未指定逻辑运算符，则搜索条件的结果就是指定的谓词的结果。

首先对圆括号内的搜索条件求值。如果圆括号未指定求值顺序，则 NOT 优先于 AND，而 AND 优先于 OR。同一优先级别的运算符的求值顺序未定义，这样可以优化搜索条件。

将对每一表行应用 *search_condition*，而删除掉的行就是 *search_condition* 的结果为真的那些行。

搜索条件中的每个 *column-name* 都必须标识一个表列。

NOT

若指定 NOT，则谓词的结果相反。

expression

标识谓词的操作数。*expression* 可以是文字、列名、专用寄存器或函数。

不支持对 BLOB(n)、DATE、TIME 和 TIMESTAMP 数据类型执行算术运算。

literal

literal 可以是数据类型为 INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME 或 TIMESTAMP 的值。

column-name

标识作为谓词操作数的列。

special register
标识作为谓词操作数的专用寄存器。专用寄存器 CURRENT DATE、CURRENT TIME 和 CURRENT TIMESTAMP 可用来生成当前的日期、时间或时间戳记。

function
只能包括 MOD、LENGTH 和 RTRIM 函数。

relational operator
可以是下列任何运算符:

= 等于。

<> 不等于。

< 小于。

> 大于。

<= 小于或等于。

>= 大于或等于。

LIKE 与一个字符串相匹配。使用单字节字符集（SBCS）下划线字符来引用一个 SBCS 字符。使用双字节字符集（DBCS）下划线字符来引用一个 DBCS 字符。例如，条件 `WHERE PART_NUMBER LIKE '_0'` 返回所有以 0 结束的两位部件号（例如，20、30 和 40）。使用 %（SBCS 或 DBCS）来引用具有零个或更多个 SBCS 或 DBCS 字符的字符串。例如，条件 `WHERE DEPT_NUMBER LIKE '2%'` 返回所有以数字 2 开始的部门号（例如，20、27 或 234）。

NOT LIKE
没有一个字符相同。

IS NULL
包含空值。

IS NOT NULL
不包含空值。

AND
如果指定的话，将逻辑运算符 AND 应用于每个指定谓词的结果。

OR
如果指定的话，将逻辑运算符 OR 应用于每个指定谓词的结果。

规则:

无。

注意事项:

- 逻辑 DELETE 不会应用于以逻辑方式删除的记录。

示例:

从 EMPLOYEE 表中删除雇员号（EMPNO）003002。

```
    DELETE FROM EMPLOYEE
 WHERE EMPNO = '003002'
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

DROP

DROP 语句删除一个表或索引。

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:

描述:

TABLE *table-name*
标识要删除的基本表。*table-name* 必须标识目录中描述的表 （SQLSTATE 42704）。

INDEX *index-name*
标识要删除的索引。*index-name* 必须标识目录中描述的索引（SQLSTATE 42704）。它不能是系统对于主键所需的索引 （SQLSTATE 42704）。

规则:

无。

注意事项:

- 不应在使用表时删除表和索引（语句句柄在使用该表或索引的查询上是活动的）。删除正在使用的表和索引将使涉及表或索引的语句句柄无效。

示例:

删除表 EMPLOYEE。

```
DROP TABLE EMPLOYEE
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

EXPLAIN

EXPLAIN 语句获取关于 SELECT 语句的存取路径选择的信息。获取的信息放在名为 DB2ePLANTABLE 的用户表中。

下列平台支持 EXPLAIN 语句:

- Win32（Windows 95、Windows 98、Windows NT、Windows 2000 和 Windows XP）
- Linux

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:

```
►►—EXPLAIN—SET QUERYNO=integer—FOR—SELECT-statement————————————————►◄
```

描述:

SET QUERYNO = integer

将 *integer* 与 SELECT 语句相关联。在由 EXPLAIN 语句插入到计划表中的每一行中，对列 QUERYNO 给出 *integer* 值。

SELECT-statement

可以按 SELECT 语句的结果表的格式来指定一些新行。

规则:

integer 值必须是正数。

注意事项:

- 在使用 EXPLAIN 语句时，缺省情况下，如果 DB2ePLANTABLE 不存在，则自动创建它。
- 要以显式方式创建 DB2ePLANTABLE，请使用以下示例:

```
create table "DB2ePLANTABLE"
 (query_no int, plan_no int, table_name char(18), index_name char(18), sort_temp char(1),
   expl_timestamp timestamp, remarks varchar(300))
```

表 16 描述了 DB2ePLANTABLE 列。

表 16. DB2ePLANTABLE 列信息

列名	描述
query_no	用于将 EXPLAIN 语句连接至 DB2ePLANTABLE 中的输出的整数。
plan_no	用于表示在其中执行该语句的步骤的整数（升序）。
table_name	表名或用于唯一标识表的相关名，如果不适用，则为空。
index_name	表访问上的索引的名称（如果使用的话）。如果未使用索引，则返回空。
sort_temp	'Y' 表示需要对临时表进行排序才能处理 GROUP BY 或 ORDER BY。如果返回空，则指示不必对临时表进行排序。
expl_timestamp	执行 EXPLAIN 语句时的时间戳记值。
remarks	注释列包含空值。您可以向此列添加注释以进行薄记。

- DB2ePLANTABLE 是可以由任何应用程序修改或删除的用户表。

示例:

在开发新应用程序时，我们都希望确定为 SELECT 语句选择了什么存取路径。在本示例中，一个新应用程序查询 SALES 和 EMPLOYEES 表。EXPLAIN 语句显示是否为 SELECT 语句选择了适当的索引。

```
EXPLAIN SET QUERYNO = 100 FOR
 SELECT E.EMPNAME, S.SALES_AMOUNT
```

```
      FROM SALES S, EMPLOYEES E
            WHERE S.EMPNO = E.EMPNO
                  AND S.MONTH = ?

Index XSALES on SALES(MONTH)
Index XEMP on EMPLOYEES(EMPNO)

SELECT QUERY_NO, PLAN_NO, TABLE_NAME, INDEX_NAME,  SORT_TEMP
   FROM "DB2ePLANTABLE"

QUERY_NO  PLAN_NO  TABLE_NAME  INDEX_NAME  SORT_TEMP
--------------------------------------------------------------------------------------------
   100     1        SALES        XSALES        -
   100     2        EMPLOYEE     XEMP          -
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

GRANT

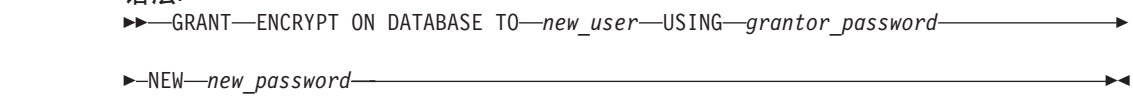
GRANT 语句给予您创建、查询和处理数据库内加密表的许可权。要执行 GRANT 操作，当前必须已连接和认证。如果数据库未加密，您（作为第一个用户）可以授权您自己进行执行 GRANT 操作所必要的认证。（有关如何执行此操作的更多信息，请参阅以下的示例 1。）

要更改自己的密码，应该用自己的用户标识执行 GRANT 操作。

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:



描述:

new_user
标识正在被授予加密特权的用户。

grantor_password
正在给新用户授予加密特权的认证用户的密码。

new_password
正在被授予其加密特权的用户的密码。

规则:

- 用户名和密码参数的长度都限制在 254 个字节内。
- 对于多字节字符，对存储器在内部使用 UTF-8 编码。因此，限制使用国际字符集写入的用户名的长度。

- DB2 Everyplace 要求授权者（即当前连接的用户）重新输入授权者密码以便能够给新用户授予特权。此限制确保授权者物理上在设备中存在。
- 密码和用户标识必须用双引号限定。

注意事项:

- 如果您是现有的用户，则必须经过连接和认证才能更改自己的密码。您仅可以更改自己的密码。
- GRANT 语句不能与参数标记或 SQLPrepare() 函数一起使用。
- 当利用未授权用户连接时尝试使用 GRANT 特权将返回 SQLSTATE 42502。使用 GRANT 语句指定错误的密码将导致 SQLSTATE 42506。

示例:

示例 1: 第一个用户授权她自己进行对尚未加密的数据库执行 GRANT 操作所必要的认证:

```
GRANT ENCRYPT ON DATABASE TO "jsk" USING "foo" NEW "foo"
```

示例 2: 现在用户 “jsk”（在以上*示例 1* 中）已被创建和认证且拥有连接。为了 “jsk” 添加另一个用户:

```
GRANT ENCRYPT ON DATABASE TO "xin" USING "foo" NEW "bar"
```

示例 3: 当前连接的用户 “jsk” 更改她自己的密码:

```
GRANT ENCRYPT ON DATABASE TO "jsk" USING "foo" NEW "fie"
```

示例 4: 当前仍然连接的用户 “jsk” 使用她的新密码来添加另一个用户:

```
GRANT ENCRYPT ON DATABASE TO "thf" USING "fie" NEW "fum"
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

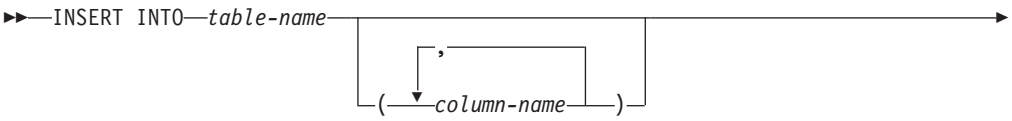
INSERT

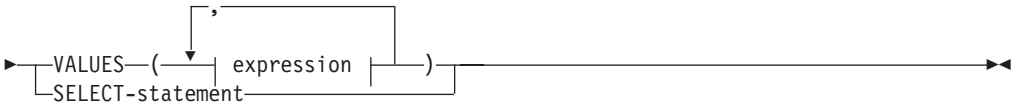
INSERT 语句使用提供的值将一行或多行插入表中。

调用:

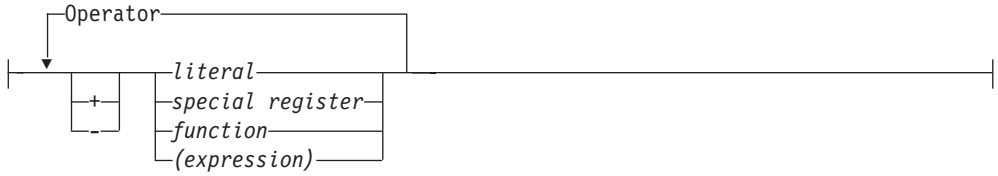
此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:





expression:



operator:

描述:

INTO *table-name*

标识插入操作的表。名称必须标识现有的表，但不能用来标识目录表。

(*column-name***,...)**

指定为其提供插入值的列。每个名称都必须是标识一个表列的未限定名。不得多次标识同一个列。

省略列列表是列表的隐式规范，在此列表中，每个表列都按从左到右的顺序标识。

VALUES

引入要插入的一行值。

每行的值的数目必须等于列列表中的名称数目。第一个值被插入到列表中的第一列中，第二个值被插入到第二列中，依此类推。

expression

表达式（expression）可以是文字、专用寄存器、函数或复杂表达式。

不支持对 CHAR、VARCHAR、BLOB(n)、DATE、TIME 和 TIMESTAMP 数据类型执行算术运算。

literal

文字（literal）可以是任何支持的数据类型为 INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME 或 TIMESTAMP 的值。

special register

专用寄存器 CURRENT DATE、CURRENT TIME 和 CURRENT TIMESTAMP 可用来生成当前的日期、时间和时间戳记。

SELECT-statement

可以按 SELECT 语句的结果表的格式来指定一些新行。可以是一行、多行，或者不指定任何行。如果结果表是空的，则 SQLCODE 会被设置为 +100，SQLSTATE 被设置为 '02000'。选择语句的基本对象不能是 INSERT 的基本对象。

规则:

缺省值 将缺省值或空值插入到不在列列表中的任何列中。必须在列列表中包括不允许缺省值或空值的列。

长度 如果一个列的插入值是数字，则该列必须是数字列，且具有表示该数字的整数部分的容量。如果一个列的插入值是字符串，则该列必须是字符串列，且长度属性至少与该字符串的长度一样大。

赋值 根据 *DB2 Universal Database SQL Reference* 中描述的赋值规则，将插入值赋给各列。

示例:

示例 1: 将一个具有下列信息的雇员插入到 EMPLOYEE 表中:

- 雇员号（EMPNO）是 002001
- 名（FIRSTNAME）是 John
- 姓（LASTNAME）是 Harrison
- 部门号（DEPT）是 600
- 电话号码（PHONENO）是 4900
- 薪水（SALARY）是 50000
- 聘用日（HIREDATE）是 01/12/1989

```
INSERT INTO EMPLOYEE
 VALUES ('002001', 'John', 'Harrison', '600', '4900', 50000, '01/12/1989')
```

示例 2: 将一个具有下列信息的新雇员插入到 EMPLOYEE 表中:

- 雇员号（EMPNO）是 003002
- 名（FIRSTNAME）是 Jim
- 姓（LASTNAME）是 Gray

```
    INSERT INTO EMPLOYEE (EMPNO, FIRSTNAME, LASTNAME)
      VALUES ('003002', 'Jim', 'Gray')
```

示例 3: 创建表 EMP_ACT_COUNT。将 EMP_ACT 表中具有涉及到项目号的雇员号（EMPNO）的那些行装入 EMP_ACT_COUNT 中。

```
CREATE TABLE EMP_ACT_COUNT
   ( EMPNO CHAR(6)  NOT NULL,
     COUNT          INTEGER)

INSERT INTO EMP_ACT_COUNT
    SELECT EMPNO, COUNT(*)
   FROM EMP_ACT
   GROUP BY EMPNO
```

限制:

1. SELECT 语句的列数据类型必须与目标表的列定义完全相同（除了可空性之外）。
2. 不允许使用 ORDER BY 和 LIMIT 子句。

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』

- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

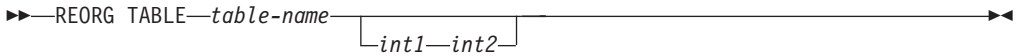
REORG TABLE

REORG TABLE 语句压缩与指定的表相关联的数据。

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:



描述:

REORG TABLE *table-name*
标识重组操作的表。名称必须标识现有的表。

int1
需要恢复的字节的可选最小百分比。

int2
需要为将要执行的表压缩恢复的最小字节数。

规则:

- 可选的值 *int1* 和 *int2* 必须一起使用，或全都不用。
- 可选的值 *int1* 必须是非负数。
- 可选的值 *int1* 必须介于 0 与 100 之间。

注意事项:

- DB2 Everyplace 可以以内部方式调用表重组。
- 第一个可选参数是表必须包含的不可用的字节的百分比（即百分之十（10）意味“至少百分之十的空间不可用”。）第二个可选参数是表必须包含的不可用的字节数（即 1000 将意味“至少 1000 个字节必须是不可用的空间”。）必须符合两个条件，才可以进行表的实际重组。
- 如果没有指定参数，DB2 Everyplace 对这此选项使用缺省值。缺省百分比是 30 且缺省字节是 6144。因此，“reorg table t1”与“reorg table t1 30 6144”相同。
- 如果重组方式设置为已启用，则 DB2 Everyplace 将自动重组表。如果在 DELETE 或 UPDATE 上启用了重组，则在执行语句之后，会对目标表执行“reorg table table_name 50 30270”。如果在 DROP TABLE 上启用了重组，则在删除表处理结束时执行“reorg table DB2eSYSTABLES 30 10240”（对于 DB2eSYSCOLUMNS 和 DB2eSYSRELS 也是如此）。
- 在 C/C++ 程序中，通过使用具有属性 SQL_ATTR_REORG_MODE 的 CLI/ODBC 函数 SQLSetStmtAttr 设置重组方式。在 JAVA 程序中，通过 DB2eStatement 接口 enableReorg 方法设置重组方式。缺省值是启用重组。
- 重组表时，通过物理上回收删除和更新创建的不可用空间来压缩包含表的数据文件。然后将表的索引更新为指向行的新物理位置。
- 可以重组“DB2 Everyplace 系统目录”基本表。

- 在执行 REORG TABLE 语句时，数据库中不应发生任何其它活动。

示例:

使用缺省值压缩 VNNURSE 表。

```
REORG TABLE VNNURSE
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

REVOKE

REVOKE 语句允许已连接的认证用户撤销现有用户的加密特权。

调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:

描述:

user
标识正在被撤销其加密特权的用户。

规则:

- 用户参数必须是定界标识。它的长度限制在 254 个字节内。
- 对于多字节字符，对存储器在内部使用 UTF-8 编码。因此，限制使用国际字符集写入的用户名的长度。
- 如果除去了所有具有加密特权的用户，在当前会话期间，可以继续访问加密表。在终止当前会话之后，加密表不再是可访问的。

注意事项:

- 用户必须已连接并认证才能撤销现有的用户的特权。如果您是已连接的认证用户，可以撤销包括您自己在内的任何用户的特权。
- REVOKE 语句不能与参数标记或 SQLPrepare() 函数一起使用。
- 当利用未授权用户连接时尝试使用 REVOKE 特权将返回 SQLSTATE 42502。尝试撤销非现有用户的特权将导致 SQLSTATE 42501。

示例:

当前已连接且认证的用户除去用户 “jsk” 的加密特权:

```
REVOKE ENCRYPT ON DATABASE FROM "jsk"
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』

- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

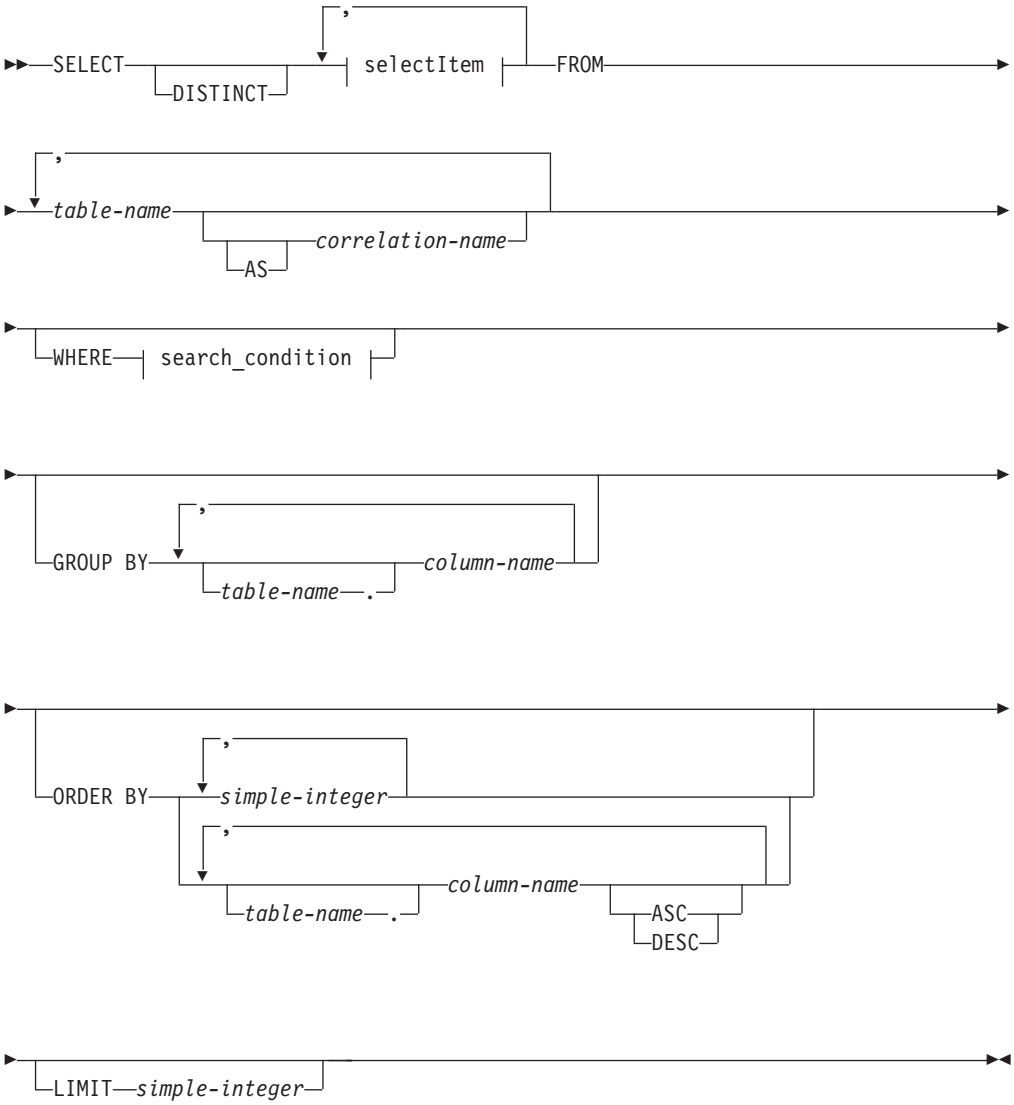
SELECT

SELECT 语句是一种查询形式。

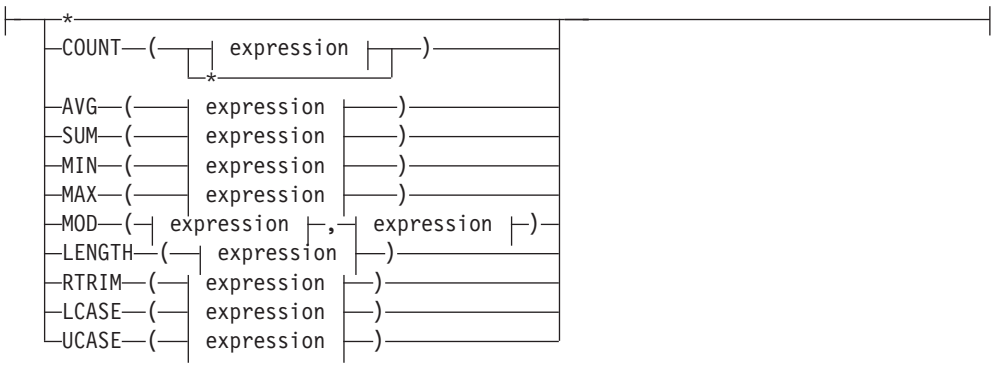
调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

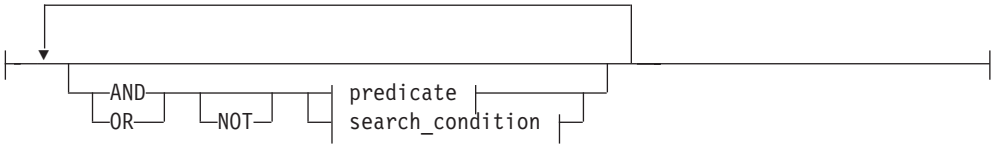
语法:



selectItem:

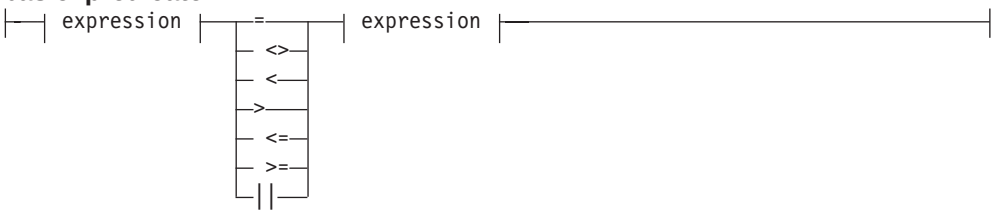


search_condition:



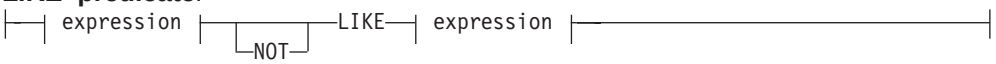
predicate:

basic predicate:



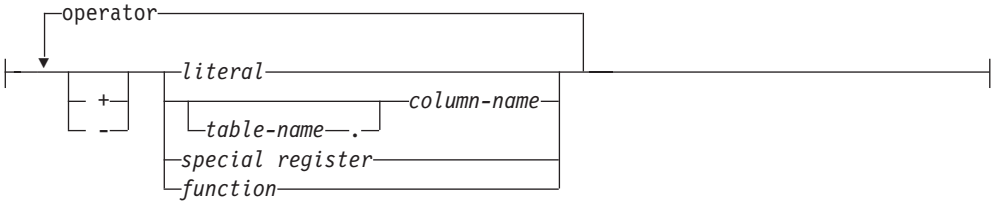
IN predicate:

LIKE predicate:

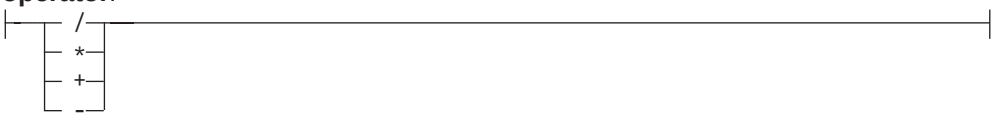


NULL predicate:

expression:



operator:



注:

1 仅 NULL 谓词中允许 BLOB 表达式。

描述:

selectItem

***** 指定所有列。若指定了 *, 则它必须是唯一的选择项。

COUNT(*)
COUNT 函数返回一组行或值中的行或值的数目。COUNT(*) 的自变量是一组行。结果是集合中的行数。只包含 NULL 值的行也被计入在内。

expression
expression 可以是文字、列名、函数或专用寄存器。有效函数包括: COUNT、AVG、SUM、MIN、MAX、MOD、LENGTH 和 RTRIM。

不支持对 CHAR、VARCHAR、BLOB(n)、DATE、TIME 和 TIMESTAMP 数据类型执行算术运算。

literal
literal 可以是数据类型为 INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME 和 TIMESTAMP 的值。

table-name
标识包含您正在查询的列的表。

. 包含两部分的列标识 *table-name.column-name* 中的分隔符。

column-name
标识您正在查询的列。

COUNT(*expression*)
COUNT(*expression*) 的自变量是一组行。此函数应用于通过消去空值而从自变量值派生的行集。结果是集合中的非空值（包括重复的值）的数目。

AVG(*expression*)
AVG(*expression*) 函数返回 *expression* 的平均值。自变量值必须是数字, 它们的和必须在结果数据类型的范围之内。此函数应用于通过消去空值而从自变量值派生的值的集合。结果可以为空。

SUM(*expression*)

SUM(*expression*) 函数返回 *expression* 的值的和。自变量值必须是数字，它们的和必须在结果数据类型的范围之内。此函数应用于通过消去空值而从自变量值派生的值的集合。

MIN(*expression*)

MIN(*expression*) 函数返回 *expression* 值集合中的最小值。自变量值可为除 BLOB 外的任何内置类型。此函数应用于通过消去空值而从自变量值派生的值的集合。

MAX(*expression*)

IAX*expression*) 函数返回 *expression* 值集合中的最大值。自变量值可为除 BLOB 外的任何内置类型。此函数应用于通过消去空值而从自变量值派生的值的集合。

MOD(*expression, expression*)

MOD(*expression, expression*) 函数返回第一个自变量除以第二个自变量的余数。仅当第一个自变量是负数时，结果才是负数。

第一个和第二个自变量可以是 SMALLINT 或 INTEGER。

如果两个自变量都是 SMALLINT，则此函数的结果是 SMALLINT；否则，它是 INTEGER。结果可以为空；如果任何自变量为空，则结果是空值。

(*expression* || *expression*)

(*expression* || *expression*) 返回两个字符串自变量的并置。这两个自变量必须是兼容类型。

函数的结果是字符串。其长度是两个自变量的长度的和。如果自变量可以为空，则结果可以为空；如果自变量为空，则结果为空值。

LENGTH(*expression*)

LENGTH(*expression*) 函数返回值的长度。

自变量可以是返回下列内置数据类型的值的表达式:

- VARCHAR
- CHAR
- BLOB

该函数的结果是整数。如果自变量可以为空，则结果可以为空；如果自变量为空，则结果为空值。

结果是自变量的长度。变长字符串的长度是实际长度，而不是最大长度。

BLOB 的长度是用来表示值的字节数。

考虑称为 ADDRESS 的 VARCHAR(50) 列，其值为 '895 Don Mills Road'。`LENGTH(ADDRESS)` 返回值 18。

RTRIM(*expression*)

RTRIM(*expression*) 函数从字符串末尾除去空格。

自变量可以是 CHAR 或 VARCHAR 数据类型。

该函数的结果数据类型始终是 VARCHAR。

所返回的类型的长度参数与自变量数据类型的长度参数相同。

字符串的结果的实际长度是字符串表达式的长度减去为空白字符除去的字节数。图形字符串的实际长度是字符串表达式的长度（以双字节字符数计）减去已除去的双字节空白字符数。如果所有字符都被除去了，则结果为空白变长字符串（长度为零）。

如果自变量可以为空，则结果可以为空；如果自变量为空，则结果为空值。

考虑称为 NAME 的 CHAR(50) 列，其值为 'Cliff '。RTRIM(NAME) 返回 'Cliff'。LENGTH(RTRIM(NAME)) 返回 5。

LCASE / UCASE

LCASE 或 LOWER 函数返回一个字符串，其中所有 SBCS 字符都已转换为小写字符。即，字符 A 至 Z 将转换为字符 a 至 z，而带有区分标记的字符将转换为其小写等效项（如果它们存在的话）。

自变量必须是其值为 CHAR 或 VARCHAR 数据类型的表达式。

该函数的结果的数据类型和长度属性与自变量的数据类型和长度属性相同。如果自变量可以为空，则结果可以为空；如果自变量为空，则结果为空值。

确保 EMPLOYEE 表中的列 JOB 的值中的字符将以小写字符的形式返回。例如:

```
SELECT LCASE(JOB)
     FROM EMPLOYEE
     WHERE EMPNO = '000020';
```

special register

专用寄存器 CURRENT DATE、CURRENT TIME 和 CURRENT TIMESTAMP 可用来生成当前的日期、时间和时间戳记。

FROM

FROM 子句指定中间结果表。

如果指定一个表引用，则中间结果表就是那个表引用的结果。如果指定了多个表引用，则中间结果表由指定表引用的行的所有可能组合组成（笛卡尔乘积）。每一结果行都是第一个表引用中的一行与第二个表引用中的一行的并置，接着与第三个表引用中的一行并置，依此类推。结果中的行数是所有独立的表引用中的行数的乘积。最多可以在 FROM 子句中指定 20 个表。

table-name

指定成表引用的每个 *table-name* 都必须标识现有的表。

AS

标识表定义。

correlation-name

每个 *correlation-name* 都定义为 *table-name* 紧前面的指示符。如果对表指定一个相关名，则对该表列的任何限定引用都必须使用该相关名，而不是使用表名。如果指定同一个 *table-name* 两次，则至少一个指定的后面应跟随有 *correlation-name*。*correlation-name* 用来限定对表列的引用。作为限定符，相关名可以用来避免不明确性或建立相关引用。它也可以仅仅用作表的简短名。

WHERE

指定选择行的条件。可以省略此子句，或指定搜索条件。如果省略此子句，则会选择所有表行。

search_condition

search_condition 指定关于给定行的为真、为假或未知的条件。

search_condition 的结果通过对每个指定谓词的结果应用指定的*逻辑运算符*（AND、OR 或 NOT）而导出。谓词对两个值作比较。如果未指定逻辑运算符，则搜索条件的结果就是指定的谓词的结果。

首先对圆括号内的搜索条件求值。如果圆括号未指定求值顺序，则 NOT 优先于 AND，而 AND 优先于 OR。同一优先级别的运算符的求值顺序未定义，这样可以优化搜索条件。

将对每一表行应用 *search_condition*，而所选择的行就是 *search_condition* 的结果为真的那些行。

搜索条件中的每个 *column-name* 都必须标识一个表列。

NOT

若指定 NOT，则谓词的结果相反。

expression

expression 可以是文字、列名、专用寄存器或函数。

不支持对 CHAR、VARCHAR、BLOB(n)、DATE、TIME 和 TIMESTAMP 数据类型执行算术运算。

literal

literal 可以是数据类型为 INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME 或 TIMESTAMP 的值。

table-name

标识包含作为谓词操作数的列的表。

. 两部分列标识 *table-name.column-name* 中的分隔符。

column-name

标识作为谓词操作数的列。

special register

标识作为谓词操作数的专用寄存器。专用寄存器 CURRENT DATE、CURRENT TIME 和 CURRENT TIMESTAMP 可用来生成当前的日期、时间和时间戳记。

function

可包括 LCASE、UCASE、MOD、LENGTH 和 RTRIM 函数。

operator

可以是下列任何运算符:

= 等于。

<> 不等于。

< 小于。

> 大于。

<= 小于或等于。

>= 大于或等于。

|| 返回两个字符串自变量的并置。

LIKE 与一个字符串相匹配。使用单字节字符集（SBCS）下划线字符来引用

一个 SBCS 字符。使用双字节字符集（DBCS）下划线字符来引用一个 DBCS 字符。例如，条件 `WHERE PART_NUMBER LIKE '_0'` 返回所有以 0 结束的两位部件号（例如，20、30 和 40）。使用 %（SBCS 或 DBCS）来引用具有零个或更多个 SBCS 或 DBCS 字符的字符串。例如，条件 `WHERE DEPT_NUMBER LIKE '2%'` 返回所有以数字 2 开始的部门号（例如，20、27 或 234）。

NOT LIKE
没有一个字符相同。

IN 与一组值相匹配。IN 谓词将一个值与一组值相比较。

示例:

SELECT lname, fname FROM emp WHERE state IN (’CA’, ’AZ’, ’OR’);

SELECT c1 FROM t1 WHERE c1*5-6 IN (mod(c2,2)+5,c3+4/2);

NOT IN
与一组值不匹配。NOT IN 谓词将一个值与一组值相比较。

示例:

SELECT empid FROM emp WHERE city NOT IN (’San Jose’, ’Morgan Hill’, ’Santa Clara’);

IS NULL
包含空值。

IS NOT NULL
不包含空值。

AND
如果指定的话，将逻辑运算符 AND 应用于每个指定谓词的结果。

OR
如果指定的话，将逻辑运算符 OR 应用于每个指定谓词的结果。

GROUP BY
指定由 R 的行分组组成的中间结果表。R 是子选择的前一子句的结果。

ORDER BY
指定结果表的行的排序。

column-name
通常标识结果表的列。在这种情况下，*column-name* 必须是选择列表中命名列的列名。

simple-integer
必须大于 0 且不大于结果表中的列数。整数 *n* 标识结果表的第 *n* 列。

ASC
以升序使用列的值。

DESC
以降序使用列的值。

LIMIT *simple-integer*
将要返回给应用程序的行数限制为结果集中的头 *n* 行，其中 *n* 是整数。必须大于 0。

关系运算符

可以是下列其中一个运算符

+ 加

- 减

***** 乘

/ 除

规则:

BLOB 数据类型列不能用于 GROUP BY、ORDER BY 和 DISTINCT 子句。

注意事项:

- SELECT DISTINCT 语句可包含最多 8 列。
- GROUP BY 子句可包含最多 8 列。
- ORDER BY 子句可包含最多 8 列。
- 在 ORDER BY 子句中指定的所有列必须出现在选择列表中。例如，以下查询无效:

  ```
  SELECT EMPNO, FIRSTNAME FROM EMPLOYEE ORDER BY LASTNAME
  ```

  以下查询有效:

  ```
  SELECT LASTNAME, EMPNO, FIRSTNAME FROM EMPLOYEE ORDER BY LASTNAME
  ```

示例:

示例 1: 从 EMPLOYEE 表中选择在 01/01/1980 后聘用的雇员（EMPNO 和 LASTNAME），并按他们的姓（LASTNAME）的顺序排列他们。

```
SELECT EMPNO, LASTNAME FROM EMPLOYEE
 WHERE HIREDATE > '01/01/1980'
 ORDER BY LASTNAME
```

示例 2: 计算 EMPLOYEE 表中每个部门的平均薪水。

```
SELECT DEPT, AVG(SALARY) FROM EMPLOYEE
 GROUP BY DEPT
```

示例 3: 计算每个销售区域的最大销售量，按区域从最高到最低销售量显示结果。

```
SELECT REGION, MAX(SALES_VOL) FROM SALES
 GROUP BY REGION ORDER BY 2 DESC
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

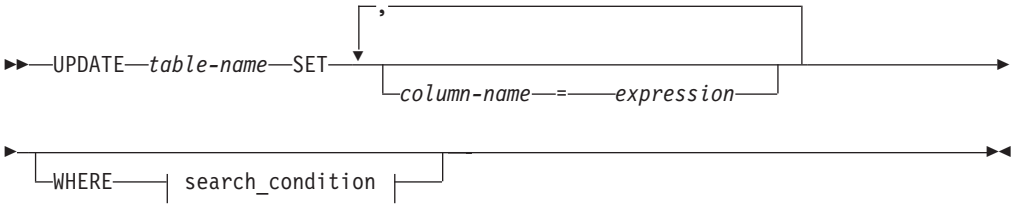
UPDATE

UPDATE 语句更新表行中指定列的值。

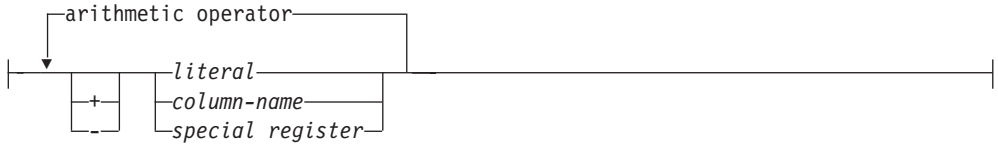
调用:

此语句可以在使用 DB2 CLI 函数的应用程序中使用，也可以通过 CLP 发出。

语法:

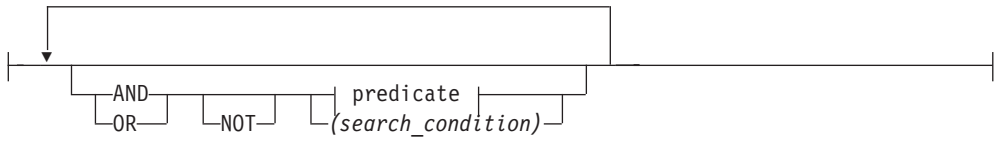


expression:



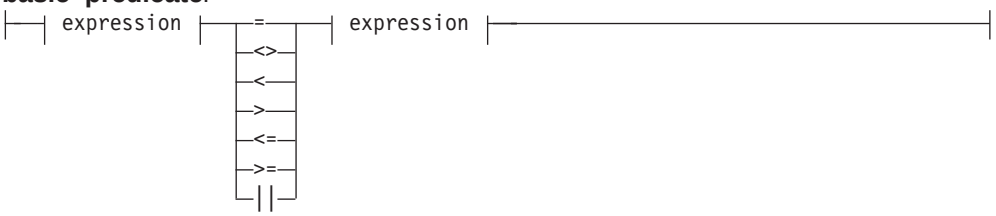
operator:

search_condition:

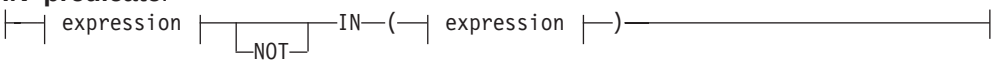


predicate:

basic predicate:

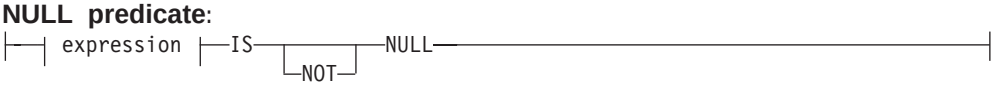


IN predicate:



LIKE predicate:

NULL predicate:



relational operator:

注:

1 仅 NULL 谓词中允许 BLOB 表达式。

描述:

table-name
是要更新的表的名称。此名称必须标识目录中描述的表，但不是目录表。

SET
对列名赋值。

column-name
标识要更新的列。*column-name* 必须标识指定表的一个列。不得多次指定一个列（SQLSTATE 42701）。

expression
expression 可以是文字、列名或专用寄存器。

不支持对 BLOB(n)、DATE、TIME 和 TIMESTAMP 数据类型执行算术运算。

literal

literal 可以是数据类型为 INTEGER、SMALLINT、DECIMAL、CHAR(n)、VARCHAR(n)、BLOB(n)、DATE、TIME 或 TIMESTAMP 的值。

special register

专用寄存器 CURRENT DATE、CURRENT TIME 和 CURRENT TIMESTAMP 可用来生成当前的日期、时间和时间戳记。

WHERE

引入一个条件，指示更新哪些行。可以省略此子句，也可以给出搜索条件。如果省略此子句，则更新所有表行。

search_condition

search_condition 指定关于给定行的为真、为假或未知的条件。

search_condition 的结果通过对每个指定谓词的结果应用指定的*逻辑运算符*（AND、OR 或 NOT）而导出。谓词对两个值作比较。如果未指定逻辑运算符，则搜索条件的结果就是指定的谓词的结果。

首先对圆括号内的搜索条件求值。如果圆括号未指定求值顺序，则 NOT 优先于 AND，而 AND 优先于 OR。同一优先级别的运算符的求值顺序未定义，这样可以优化搜索条件。

将对每一表行应用 *search_condition*，而更新的行就是 *search_condition* 的结果为真的那些行。

搜索条件中的每个 *column-name* 都必须标识一个表列。

可在搜索条件的谓词表达式中使用 CONCAT、MOD、LENGTH 和 RTRIM 函数。有关 MOD 函数的更多信息，请参阅 152 页。

NOT

若指定 NOT，则谓词的结果相反。

relational operator

可以是下列任何运算符:

= 等于。

<> 不等于。

< 小于。

> 大于。

<= 小于或等于。

>= 大于或等于。

LIKE 与一个字符串相匹配。使用单字节字符集（SBCS）下划线字符来引用一个 SBCS 字符。使用双字节字符集（DBCS）下划线字符来引用一个 DBCS 字符。例如，条件 `WHERE PART_NUMBER LIKE '_0'` 返回所有以 0 结束的两位部件号（例如，20、30 和 40）。使用 %（SBCS 或 DBCS）来引用具有零个或更多个 SBCS 或 DBCS 字符的字符串。例如，条件 `WHERE DEPT_NUMBER LIKE '2%'` 返回所有以数字 2 开始的部门号（例如，20、27 或 234）。

NOT LIKE

没有一个字符相同。

IS NULL
包含空值。

IS NOT NULL
不包含空值。

AND
如果指定的话，将逻辑运算符 AND 应用于每个指定谓词的结果。

OR
如果指定的话，将逻辑运算符 OR 应用于每个指定谓词的结果。

规则:

- *赋值*: 根据 *DB2 Universal Database SQL Reference* 中描述的赋值规则将更新值赋予列。
- UPDATE 不会应用于以逻辑方式删除的记录。

注意事项:

- 在系统方式下，缺省情况下设置脏位。如果以系统方式运行应用程序（SQL_DIRTYBIT_SET_BY_SYSTEM），则不能以手工方式设置脏位。如果您尝试设置脏位，则会出错。

  有关更多信息，请参阅第 246 页的 。

示例:

将 EMPLOYEE 表中雇员号（EMPNO）'003002' 的电话号码更改为 '1234'。

```
UPDATE EMPLOYEE
 SET PHONENO = '1234'
 WHERE EMPNO = '003002'
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 『赋值和比较的数据类型兼容性』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』
- 第 164 页的『SQLState 类代码的摘要』

赋值和比较的数据类型兼容性

赋值操作是在执行 INSERT 和 UPDATE 语句期间执行的。比较操作是在执行包含谓词的语句期间执行的。涉及的操作数的数据类型必须兼容，如从表 17 到第 161 页的表 19 中所示。

如果数据类型列包含:

X 操作数的数据类型相兼容。

空白 操作数的数据类型不兼容。

表 17. 数据类型兼容性，表 1

SQL 数据类型	INT	SMALLINT	DECIMAL	BLOB
INT	X	X	X	

表 17. 数据类型兼容性，表 1 (续)

SQL 数据类型	INT	SMALLINT	DECIMAL	BLOB
VARCHAR				
BLOB				X
DECIMAL	X	X	X	
CHAR				
SMALLINT	X	X	X	
DATE				
TIME				
TIMESTAMP				

表 18. 数据类型兼容性，表 2

SQL 数据类型	CHAR	VARCHAR
INT		
VARCHAR	X	X
BLOB		
DECIMAL		
CHAR	X	X
SMALLINT		
DATE	X	X
TIME	X	X
TIMESTAMP	X	X

表 19. 数据类型兼容性，表 3

SQL 数据类型	DATE	TIME	TIMESTAMP
INT			
VARCHAR	X	X	X
BLOB			
DECIMAL			
CHAR	X	X	X
SMALLINT			
DATE	X		
TIME		X	
TIMESTAMP			X

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 列表』

SQL 符号和缺省数据类型

表 20. SQL 符号和缺省数据类型

SQL 数据类型	符号 SQL 数据类型	缺省符号 C 数据类型
BLOB	SQL_BLOB	SQL_C_BINARY
CHAR	SQL_CHAR	SQL_C_CHAR
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
VARCHAR	SQL_VARCHAR	SQL_C_CHAR

数据类型属性

显示有关下列数据类型属性的信息:

- 精度
- 小数位
- 长度
- 显示大小

精度:

数字列或参数的精度指的是列或参数的数据类型使用的最大位数。非数字列或参数的精度通常是指列或参数的最大长度或定义长度。下表定义每个 SQL 数据类型的精度。

表 21. 精度

fSqlType	精度
SQL_CHAR SQL_VARCHAR	列或参数的定义长度。例如，定义为 CHAR(10) 的列的精度为 10。
SQL_DECIMAL	定义的最大位数。例如，定义为 DECIMAL(10,3) 的列的精度为 10。
SQL_SMALLINT a	5
SQL_INTEGER a	10
SQL_BLOB	列或参数的定义长度。例如，定义为 BLOB(10) 的列的精度为 10。
SQL_DATE a	10（yyyy-mm-dd 格式中的字符数）。
SQL_TIME a	8（hh:mm:ss 格式中的字符数）。
SQL_TIMESTAMP	26（TIMESTAMP 数据类型使用的 “yyyy-mm-dd-hh.mm.ss.ffffff” 格式中的字符数。）

a: 对于此数据类型，将忽略 `SQLBindParameter()` 的 *cbParamDef* 自变量。

小数位:

数字列或参数的小数位指的是小数点右边的最大位数。下表定义每种 SQL 数据类型的小数位。

表 22. 小数位

fSqlType	小数位
SQL_CHAR SQL_VARCHAR	不适用。
SQL_DECIMAL	小数点位置右边的定义位数。例如，定义为 DECIMAL(10, 3) 的列的小数位为 3。
SQL_SMALLINT SQL_INTEGER	0
SQL_BLOB	不适用。
SQL_DATE SQL_TIME	不适用。
SQL_TIMESTAMP	6（ “yyyy-mm-dd-hh.mm.ss.ffffff” 格式中的小数点右边的位数。）

长度:

列的长度是将数据转换至其缺省 C 数据类型时返回至应用程序的最大字节数。对于字符数据，该长度不包括 null 终止字节。注意，列的长度可能与在数据源上存储数据所需的字节数不同。

下表定义每种 SQL 数据类型的长度。

表 23. 长度

fSqlType	长度
SQL_CHAR SQL_VARCHAR	列的定义长度。例如，定义为 CHAR(10) 的列的长度为 10。
SQL_DECIMAL	最大数字是位数加上二。因为这些数据类型是作为字符串返回的，所以需要附加字符来表示附加位数（符号和小数点）。例如，定义为 DECIMAL(10,3) 的列的长度为 12。
SQL_SMALLINT	2（2 字节）。
SQL_INTEGER	4（4 字节）。
SQL_BLOB	列的定义长度。例如，定义为 BLOB(10) 的列的长度为 10。
SQL_DATE SQL_TIME	6（DATE_STRUCT 或 TIME_STRUCT 结构的大小）。
SQL_TIMESTAMP	16（TIMESTAMP_STRUCT 结构的大小）。

显示大小:

列的显示大小是以字符形式显示数据所需的最大字节数。下表定义每种 SQL 数据类型的显示大小。

表 24. 显示大小

fSqlType	显示大小
SQL_CHAR SQL_VARCHAR	列的定义长度。例如，定义为 CHAR(10) 的列的显示大小为 10。
SQL_DECIMAL	列的精度加上二（表示精度位的符号和小数点）。例如，定义为 DECIMAL(10,3) 的列的显示大小为 12。
SQL_SMALLINT	6（一个符号加上 5 位）。
SQL_INTEGER	11（一个符号加上 10 位）。
SQL_BLOB	列的定义长度乘以 2（每个二进制字节用 2 位十六进制数字表示）。例如，定义为 BLOB(10) 的列的显示大小为 20。
SQL_DATE	10（格式为 yyyy-mm-dd 的日期）。
SQL_TIME	8（格式为 hh:mm:ss 的时间）。
SQL_TIMESTAMP	26（格式为 yyyy-mm-dd-hh.mm.ss.ffffff 的时间戳记）。

SQLState 列表

本节帮助您解释 SQL 或 CLI 生成的错误消息。

- 『SQLState 类代码的摘要』包含一般错误类别的列表。
- 第 165 页的『SQL 报告的 SQLState 消息』、第 168 页的『CLI 报告的 SQLState 消息』和第 174 页的『JDBC 报告的 SQLState 消息』包含每个错误的描述，还为 SQL 提供了生成错误的函数的名称。

如果安装了 DB2 UDB，还可使用 DB2 命令行处理器查找 SQLSTATE 描述:

1. 要打开命令行处理器，选择**开始 —> 程序 —> DB2 —> 命令行处理器**。
2. 在命令行上，输入 ? [*SQLSTATE*]。

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 『SQLState 类代码的摘要』

SQLState 类代码的摘要

第 168 页的表 27 中的 SQLState 消息的前两个字符是**粗体**的，以指示类代码。表 25 对这些类代码作了*总结*。

表 25. SQLState 类代码

代码	类
00	取消限定成功完成
01	警告
02	没有数据
07	动态 SQL 错误
08	连接异常

表 25. SQLState 类代码 (续)

代码	类
09	触发操作异常
0A	功能部件不受支持
0F	记号无效
21	基数违例
22	数据异常
23	约束违例
24	游标状态无效
25	游标事务无效
26	SQL 语句标识无效
28	授权规范无效
2D	事务终止无效
2E	连接名无效
34	游标名无效
38	外部函数异常
39	外部函数调用异常
40	事务回滚
42	语法错误或访问规则违例
44	具有检查选项违例
46	Java DDL
51	无效的应用程序状态
54	超过了 SQL 或产品限制
55	对象未处于先决条件状态
56	其它 SQL 或产品错误
57	资源不可用或操作员介入
58	系统错误资源
59	“DB2 Everyplace 管理员”错误
HY	由 DB2 CLI 或 ODBC 驱动程序生成
IM	由 ODBC 驱动程序管理器生成

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 164 页的『SQLState 列表』
- 第 72 页的『DB2 Everyplace 支持的参数标记』

SQL 报告的 SQLState 消息

第 166 页的表 26 列出了 DB2 Everyplace SQL 引擎报告的 SQL 语句的所有 SQLSTATE。DB2 CLI 报告的 SQLSTATE 列示在第 175 页的『DB2 CLI 函数摘要』中的每个 DB2 CLI 函数描述下面。

表 26. SQL 报告的 SQLSTATE 消息

SQLSTATE	描述	解释
01000	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
01004	值被截断。	值被系统强制转换或调整函数截断。
01550	未创建索引。	因为已存在具有指定描述的索引，所以未创建索引。
02000	找不到任何行。	执行 FETCH、DELETE 或 UPDATE 语句期间，找不到任何行。
07001	参数数目不正确。	参数标记尚未绑定。
07005	参数无效。	游标的语句名所标识的已准备语句不能与游标相关联。
07006	变量无效。	因为输入主变量的数据类型不正确，所以不能使用它。
08002	连接已存在。	连接已存在。
22001	值需要截断。	值需要被系统强制转换或调整函数截断。
22002	未提供空指示符。	因为未提供存储器，所以不能赋予 NULL 值。
22003	数值超出范围。	数值不在其目标列的范围之内。
22007	日期时间格式无效。	日期时间值的字符串表示法语法不正确。
22008	日期时间值超出范围。	日期时间值的字符串表示法超出范围。
22012	除零。	尝试了除零操作。
22504	分段 MBCS 字符。	数据包含格式不正确的多字节字符。
23502	不允许空值。	不允许对 NOT NULL 列赋予 NULL 值。
23505	值不是唯一的。	因为可能会生成重复键，所以操作无效。
23513	值无效。	INSERT 或 UPDATE 语句的结果行不符合检查约束定义。
23515	指定了多个主键子句。	指定了多个主键子句。
24000	游标状态无效。	*StatementHandle* 处于已执行状态，但没有任何结果集与 *StatementHandle* 相关联。
24501	游标未打开。	因为未生成结果集，所以 FETCH 无效。
24505	游标未定位。	因为游标未定位在行上，所以 FETCH 无效。
34000	游标名无效。	游标名无效。
42501*	不允许授权标识对标识的对象执行指定的操作	当前用户正尝试从不存在的用户除去特权。
42502*	不允许授权标识执行指定的操作	当前用户不具有已认证的连接。当应用程序（它不具有加密库或 CryptoPlugin.dll）执行与加密相关的 SQL 命令（GRANT、REVOKE 和 CREATE TABLE）时，将返回 “42502” 错误。这将避免应用程序崩溃。
42505*	发生了连接授权故障。	已注册的用户试图连接，但无法认证。
42506*	所有者授权故障。	不能认证已连接的用户。（错误的密码。）
42601	语法错误。	在 SQL 语句中检测到语法错误。
42603	字符串常量没有结束定界符。	字符串常量或定界标识没有结束定界符。
42610	参数标记的使用无效。	语句包含无效的参数标记。请参阅第 72 页的表 14 以了解参数标记的有效用法。
42611	长度规范无效。	长度规范超过限制。
42614	重复的关键字无效。	重复的关键字无效。
42621	检查约束无效。	检查约束无效。
42622	名称太长。	标识的名称太长。
42702	列名引用混乱。	可能正在引用多个列。

表 26. SQL 报告的 SQLSTATE 消息 (续)

SQLSTATE	描述	解释
42703	列名未定义。	列名不在引用的表中。
42704	对象未定义。	表不存在。
42710	命名的对象已存在。	已存在同名的表。
42711	列名重复。	多次指定相同的列名。
42802	值的数目与列的数目不匹配。	指定的值的数目与指定或隐含的列数不同。
42803	SELECT 列表中的列引用未在 GROUP BY 子句中指定。	选择列表中包含列名和聚集函数，但没有 GROUP BY 子句。
42818	操作数的数据类型不兼容。	操作的操作数数据类型不兼容。
42820	文字值超出范围。	指定的数值不在可接受的范围之内。
42821	数据类型不兼容。	值与目标列的数据类型不兼容。
42822	ORDER BY 项无效。	ORDER BY 项不在选择列表中。
42824	LIKE 操作数无效。	LIKE 的操作数不是字符串，或第一个操作数不是列。
42829	FOR UPDATE OF 无效。	因为不能修改游标所指示的结果表，所以 FOR UPDATE OF 无效。
42830	外键不符合父键的描述。	外键不符合父键的描述。
42831	主键中存在可空的列。	主键子句中指定的列不能是可空的。
42832*	越权访问系统对象。	不允许对系统对象执行此操作。
42884	函数名未知。	找不到带有指定名称和兼容自变量的函数或过程。
42887	功能不受支持。	该功能在当前发行版中不受支持。
42894	DEFAULT 值无效。	DEFAULT 值无效。
42902	对象表引用重复。	FROM 子句中也标识了 INSERT 语句的对象表。
42903	WHERE 子句或 SET 子句包含无效的引用。	WHERE 子句或 SET 子句包含无效的引用，如列函数。
42962	LOB 列不能用作键。	LOB 列不能用作主键。
54001	语句太长。	查询语句太长。
54008	键太长。	主键、外键或索引中的列太多。
54010	表记录长度太长。	表的记录长度太长。
55002	DB2ePLANTABLE 未正确定义。	DB2ePLANTABLE 的声明不正确，不能执行 EXPLAIN。
55009	文件是只读的。	文件是只读的。在只读环境中，只能执行 SELECT 查询。
57001	表不可用。	不能对处于事务作用域之下的表执行 REORG。
57011	内存不足。	系统无法分配动态内存。
57014	处理因中断而被取消。	查询的执行因用户中断而被取消。
58004	内部系统错误（继续）。	发生不严重的系统错误。
58005	内部系统错误（停止）。	发生严重的系统错误。

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 164 页的『SQLState 列表』
- 第 72 页的『DB2 Everyplace 支持的参数标记』

• 第 164 页的『SQLState 类代码的摘要』

CLI 报告的 SQLState 消息

表 27. CLI 报告的 SQLState 消息

SQLSTATE	CLI 函数名	描述	解释
01000	SQLAllocHandle	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
01000	SQLFreeHandle	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
01002	SQLDisconnect	断开连接错误。	断开连接时出错。但是，断开连接成功。（函数返回 SQL_SUCCESS_WITH_INFO。）
01004	SQLDescribeCol	数据被截断。	*ColumnName* 自变量中返回的列名比 *BufferLength* 自变量中指定的值要长。*NameLengthPtr* 自变量包含完整列名的长度。（函数返回 SQL_SUCCESS_WITH_INFO。）
01004	SQLFetch	数据被截断。	对一个或多个列返回的数据被截断。字符串值或数值在右端被截断。（如果不出错，则返回 SQL_SUCCESS_WITH_INFO。）
01004	SQLGetData	数据被截断。	对指定列（*ColumnNumber*）返回的数据被截断。字符串或数值在右端被截断。（返回 SQL_SUCCESS_WITH_INFO。）
01S06*	SQLFetchScroll	试图在结果集返回第一个行集之前进行取装。	当当前位置在第一行之外，且 *FetchOrientation* 是 SQL_PRIOR，或 *FetchOrientation* 是 SQL_RELATIVE，而 F*etchOffset* 是绝对值小于或等于当前 SQL_ATTR_ROW_ARRAY_SIZE 的负数，则请求的行集与结果集的起始点重叠。（函数返回 SQL_SUCCESS_WITH_INFO。）
07005	SQLDescribeCol	语句未返回结果集。	与 *StatementHandle* 相关联的语句未返回结果集。没有要描述的列。（首先调用 `SQLNumResultCols()` 以确定结果集中是否有返回行。）
07006	SQLBindParameter	转换无效。	从 *ValueType* 自变量标识的数据类型到 *ParameterType* 自变量标识的数据类型的转换不是有意义的转换。（例如，从 SQL_C_DATE 转换至 SQL_DOUBLE。）
07006	SQLFetch	转换无效。	未能将数据类型以有意义的方式转换为 `SQLBindCol()` 中的 *fCType* 所指定的数据类型。
07006	SQLGetData	转换无效。	不能将数据类型转换为自变量 *TargetType* 指定的 C 数据类型。先前对同一 *ColumnNumber* 值调用了此函数，但 *TargetType* 值不同。
07009	SQLBindCol	描述符索引无效。	对 *ColumnNumber* 自变量指定的值超出结果集中的最大列数。
07009	SQLDescribeCol	描述符索引无效	对 *ColumnNumber* 指定的值等于或小于 0。对 *ColumnNumber* 自变量指定的值大于结果集中的列数。
08001	SQLConnect	无法连接数据源。	DB2 CLI 无法建立与数据源（服务器）的连接。
08002	SQLConnect	连接正在使用中。	已使用指定的 *ConnectionHandle* 来建立与数据源的连接，该连接仍处于打开状态。

表 27. CLI 报告的 SQLState 消息 (续)

SQLSTATE	CLI 函数名	描述	解释
08003	SQLAllocHandle	连接已关闭。	*HandleType* 自变量是 SQL_HANDLE_STMT，但由 *InputHandle* 自变量指定的连接未打开。连接过程必须成功完成（且连接必须是打开的），DB2 CLI 才能分配语句句柄。
08003	SQLDisconnect	连接已关闭。	*ConnectionHandle* 自变量中指定的连接未打开。
08004	SQLConnect	应用程序服务器拒绝建立连接。	数据源（服务器）拒绝建立连接。
08S01	SQLFreeHandle	通信链路故障。	*HandleType* 自变量是 SQL_HANDLE_DBC，在函数完成处理之前，DB2 CLI 与它尝试连接的数据源之间的通信链路失效。
22002	SQLFetch	指定的输出或指示符缓冲区无效。	对 SQLBindCol() 中 *pcbValue* 自变量指定的指针值是空指针，且相应列的值为空。报告 SQL_NULL_DATA 是没有意义的。
22002	SQLGetData	指定的输出或指示符缓冲区无效。	对 *StrLen_or_IndPtr* 自变量指定的指针值是空指针，列的值为空。报告 SQL_NULL_DATA 是没有意义的。
22003	SQLExecDirect	数值超出范围。	对数字类型列指定的数值导致在赋值期间或计算中间结果期间截断该数字的整个部分。
22005	SQLGetData	赋值出错。	返回值与 *TargetType* 自变量指示的数据类型不兼容
39001 *	SQLExecute	用户定义的函数返回了无效的 SQLSTATE。	用户定义的函数返回了无效的 SQLSTATE。
40003 **08**S01	SQLBindCol	通信链路错误。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
40003 **08**S01	SQLBindParameter	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
40003 **08**S01	SQLDescribeCol	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
40003 **08**S01	SQLFreeStmt	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
40003 **08**S01	SQLGetData	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
40003 **08**S01	SQLNumResultCols	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
40003 **08**S01	SQLRowCount	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
42nnn*	SQLPrepare	语法错误。	**42**nnn SQLSTATES 指示与语句相关的各种语法或访问问题。字符 nnn 指的是任何具有该类代码的 SQLSTATE。示例：**42**nnn 指的是 **42** 类中的任何 SQLSTATE。
42xxx	SQLExecDirect	语法错误或访问规则违例。	**42**xxx SQLSTATES 指示与语句相关的各种语法或访问问题。xxx 指的是任何具有该类代码的 SQLSTATE。示例：**42**xxx 指的是 **42** 类中的任何 SQLSTATE。
42xxx	SQLNumResultCols	语法错误。	**42**xxx SQLSTATES 指示与语句相关的各种语法或访问问题。xxx 指的是任何具有该类代码的 SQLSTATE。示例：**42**xxx 指的是 **42** 类中的任何 SQLSTATE。

表 27. CLI 报告的 SQLState 消息 (续)

SQLSTATE	CLI 函数名	描述	解释
58004	SQLBindCol	意外系统故障。	不可恢复的系统错误。
58004	SQLBindParameter	意外系统故障。	不可恢复的系统错误。
58004	SQLConnect	意外系统故障。	不可恢复的系统错误。
58004	SQLDescribeCol	意外系统故障。	不可恢复的系统错误。
58004	SQLDisconnect	意外系统故障。	不可恢复的系统错误。
58004	SQLExecDirect	意外系统故障。	不可恢复的系统错误。
58004	SQLFetch	意外系统故障。	不可恢复的系统错误。
58004	SQLFreeStmt	意外系统故障。	不可恢复的系统错误。
58004	SQLGetData	意外系统故障。	不可恢复的系统错误。
58004	SQLPrepare	意外系统故障。	不可恢复的系统错误。
58004	SQLNumResultCols	意外系统故障。	不可恢复的系统错误。
58004	SQLRowCount	意外系统故障。	不可恢复的系统错误。
59101*	SQLExecute	未定义用户。	在“移动式设备管理中心”控制数据库中未定义用户。
59102*	SQLExecute	密码不正确。	用户密码与“移动式设备管理中心”中定义的密码不匹配。
59103*	SQLExecute	未定义组。	在“移动式设备管理中心”中未定义组。
59104*	SQLExecute	未定义应用程序。	在“移动式设备管理中心”中未定义应用程序。
59105*	SQLExecute	未定义预订。	在移动式设备管理中心中未定义具有 AgentAdapter 的预订。
59106*	SQLExecute	预订不完整。	预订不具有调用远程存储过程所需的所有信息。
59120*	SQLExecute	XML 转换错误。	AgentAdapter 在将用户输入数据转换为 XML 文档时失败。
59121*	SQLExecute	一般 AgentAdapter 错误。	一般 AgentAdapter 错误。
59122*	SQLExecute	装入库失败。	系统上找不到某些必需的库。
HY000	SQLAllocHandle	一般错误。	出错，该错误没有特定的 SQLSTATE。**MessageText* 缓冲区中由 `SQLGetDiagRec()` 返回的错误消息描述了该错误及其原因。
HY000	SQLFreeHandle	一般错误。	出错，该错误没有特定的 SQLSTATE。**MessageText* 缓冲区中由 `SQLGetDiagRec()` 返回的错误消息描述了该错误及其原因。
HY001	SQLAllocHandle	内存分配出错。	DB2 CLI 无法为指定的句柄分配内存。
HY001	SQLBindCol	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLBindParameter	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLConnect	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLDescribeCol	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLDisconnect	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。

表 27. CLI 报告的 SQLState 消息 (续)

SQLSTATE	CLI 函数名	描述	解释
HY001	SQLExecDirect	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLFetch	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLFreeHandle	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLFreeStmt	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLGetData	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLPrepare	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLNumResultCols	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY001	SQLRowCount	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY002	SQLBindCol	列号无效。	对自变量 *ColumnNumber* 指定的值小于 0。对自变量 *ColumnNumber* 指定的值超过数据源支持的最大列数。
HY002	SQLDescribeCol	列号无效。	对 *ColumnNumber* 自变量指定的值小于 1。对自变量 *ColumnNumber* 指定的值大于结果集中的列数。
HY002	SQLGetData	列号无效。	指定的列小于 0 或大于结果列的数目。
HY003	SQLBindCol	程序类型超出范围。	*TargetType* 不是有效的数据类型或 SQL_C_DEFAULT。
HY003	SQLBindParameter	程序类型超出范围。	*ParameterNumber* 自变量指定的值不是有效的数据类型或 SQL_C_DEFAULT。
HY003	SQLGetData	程序类型超出范围。	*TargetType* 不是有效的数据类型或 SQL_C_DEFAULT。
HY004	SQLBindParameter	SQL 数据类型超出范围。	*ParameterType* 自变量指定的值不是有效的 SQL 数据类型。
HY009	SQLBindParameter	自变量值无效。	*ParameterValuePtr* 自变量是空指针，且 *StrLen_or_IndPtr* 自变量也是空指针，而 *InputOutputType* 不是 SQL_PARAM_OUTPUT。
HY009	SQLExecDirect	自变量值无效。	*StatementText* 是空指针。
HY009	SQLNumResultCols	自变量值无效。	*StatementText* 是空指针。
HY010	SQLDescribeCol	函数顺序错误。	在对 *StatementHandle* 调用 `SQLPrepare()` 或 `SQLExecDirect()` 之前调用了该函数。
HY010	SQLExecute	函数顺序错误。	指定的 *StatementHandle* 未处于已准备状态。调用 `SQLExecute()` 之前未首先调用 `SQLPrepare()`。
HY010	SQLFetch	函数顺序错误。	在对 *StatementHandle* 调用 `SQLPrepare()` 或 `SQLExecDirect()` 之前调用了该函数。

表 27. CLI 报告的 SQLState 消息 (续)

SQLSTATE	CLI 函数名	描述	解释
HY010	SQLFreeHandle	函数顺序错误。	*HandleType* 自变量为 SQL_HANDLE_ENV 且至少有一个连接处于已分配或已连接状态。在调用 *HandleType* 为 SQL_HANDLE_ENV 的 SQLFreeHandle() 之前，必须对每个连接调用 *HandleType* 为 SQL_HANDLE_DBC 的 SQLDisconnect() 和 SQLFreeHandle()。*HandleType* 自变量是 SQL_HANDLE_DBC，而在对连接调用 SQLDisconnect() 之前调用了该函数。*HandleType* 自变量是 SQL_HANDLE_STMT；使用语句句柄调用了 SQLExecute() 或 SQLExecDirect()，且返回了 SQL_NEED_DATA。（DM）在调用 SQLFreeHandle() 之前，未释放任何附属句柄和其它资源。
HY010	SQLGetData	函数顺序错误。	调用该函数之前未首先调用 SQLFetch()。
HY010	SQLNumResultCols	函数顺序错误。	在对 *StatementHandle* 调用 SQLPrepare() 或 SQLExecDirect() 之前调用了该函数。
HY010	SQLRowCount	函数顺序错误。	在对 *StatementHandle* 调用 SQLExecute() 或 SQLExecDirect() 之前调用了该函数。
HY013	SQLAllocHandle	发生意外的内存处理错误。	*HandleType* 自变量是 SQL_HANDLE_DBC 或 SQL_HANDLE_STMT；因为未能访问底层内存对象（可能是因为内存不足），所以未能处理函数调用。
HY013	SQLBindCol	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLBindParameter	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLConnect	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLDescribeCol	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLDisconnect	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLExecDirect	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLFetch	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLFreeHandle	发生意外的内存处理错误。	*HandleType* 自变量是 SQL_HANDLE_STMT；因为未能访问底层内存对象（可能是因为内存不足），所以未能处理函数调用。
HY013	SQLGetData	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLNumResultCols	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLNumResultCols	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY013	SQLRowCount	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。

表 27. CLI 报告的 SQLState 消息 (续)

SQLSTATE	CLI 函数名	描述	解释
HY014	SQLAllocHandle	不再有句柄。	已达到可对 *HandleType* 自变量所指示的句柄类型分配的句柄数限制。
HY014	SQLExecDirect	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY014	SQLNumResultCols	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY017	SQLFreeHandle	自动分配的描述符句柄的使用无效。	*Handle* 自变量设置为自动分配的描述符或实现描述符的句柄。
HY024	SQLSetStmtAttr	无效属性值。	给出指定的 *Attribute* 值，在 *ValuePtr* 中指定了无效值。
HY090	SQLBindCol	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 1，*TargetType* 自变量是 SQL_C_CHAR、SQL_C_BINARY 或 SQL_C_DEFAULT。
HY090	SQLBindParameter	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 0。
HY090	SQLDescribeCol	字符串或缓冲区长度无效。	自变量 *BufferLength* 中指定的长度小于 1。
HY090	SQLExecDirect	字符串或缓冲区长度无效。	*TextLength* 自变量小于 1，但不等于 SQL_NTS。
HY090	SQLGetData	字符串或缓冲区长度无效。	*BufferLength* 自变量的值小于 0，*TargetType* 自变量是 SQL_C_CHAR 或 SQL_C_BINARY；或者 *TargetType* 是 SQL_C_DEFAULT，而缺省类型是 SQL_C_CHAR、SQL_C_BINARY 或 SQL_C_DBCHAR 之一。
HY090	SQLNumResultCols	字符串或缓冲区长度无效。	*TextLength* 自变量小于 1，但不等于 SQL_NTS。
HY092	SQLAllocHandle	选项类型超出范围。	*HandleType* 自变量不是: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT
HY092	SQLFreeStmt	选项类型超出范围。	对 *Option* 自变量指定的值不是 SQL_DROP 或 SQL_RESET_PARAMS。
HY093	SQLBindParameter	参数号无效。	对 *ValueType* 自变量指定的值小于 1 或大于服务器支持的最大参数数目。
HY094	SQLBindParameter	小数位的值无效。	对 *ParameterType* 指定的值是 SQL_DECIMAL 或 SQL_NUMERIC，对 *DecimalDigits* 指定的值小于 0 或大于 *ParamDef* 自变量（精度）的值。
HY104	SQLBindParameter	精度值无效。	对 *ParameterType* 指定的值是 SQL_DECIMAL 或 SQL_NUMERIC，对 *ParamDef* 指定的值小于 1。
HY105	SQLBindParameter	参数类型无效。	*InputOutputType* 不是 SQL_PARAM_INPUT。
HY106	SQLFetchScroll	取装类型超出范围。	对自变量 *FetchOrientation* 指定的值无效。SQL_CURSOR_TYPE 语句属性的值是 SQL_CURSOR_FORWARD_ONLY，而 *FetchOrientation* 自变量的值不是 SQL_FETCH_NEXT。

表 27. CLI 报告的 SQLState 消息 (续)

SQLSTATE	CLI 函数名	描述	解释
HY107	SQLFetchScroll	行值超出范围。	用 SQL_ATTR_CURSOR_TYPE 语句属性指定的值是 SQL_CURSOR_KEYSET_DRIVEN，但用 SQL_ATTR_KEYSET_SIZE 语句属性指定的值大于 0 且小于用 SQL_ATTR_ROW_ARRAY_SIZE 语句属性指定的值。
HY501	SQLConnect	*DataSource* 名无效。	指定的 *DataSource* 名称无效。
HYC00	SQLBindCol	驱动程序不起作用。	DB2 CLI 识别但不支持 *TargetType* 自变量中指定的数据类型。
HYC00	SQLBindParameter	驱动程序不起作用。	DB2 CLI 或数据源不支持由对自变量 *ValueType* 和 *ParameterType* 指定的值的组合所指定的转换。DB2 CLI 或数据源不支持对自变量 *ParameterType* 指定的值。
HYC00	SQLDescribeCol	驱动程序不起作用。	DB2 CLI 不识别 *ColumnNumber* 列的 SQL 数据类型。
HYC00	SQLGetData	驱动程序不起作用。	DB2 CLI 识别但不支持指定数据类型的 SQL 数据类型。DB2 CLI 或数据源不能执行所请求的从 SQL 数据类型到应用程序数据 *TargetType* 的转换。

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』
- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 164 页的『SQLState 列表』
- 第 72 页的『DB2 Everyplace 支持的参数标记』
- 第 164 页的『SQLState 类代码的摘要』

JDBC 报告的 SQLState 消息

表 28. JDBC 报告的 SQLState 消息

SQLSTATE	描述	解释
0100C	返回了一个或多个附加结果集。	对于 *ResultSet* 对象的并行性方式，DB2 Everyplace 不支持 ResultSet.CONCUR_UPDATABLE。将使用 ResultSet.CONCUR_READ_ONLY 代替。
0641E	批处理中有 SELECT 语句。	SELECT 语句在批处理中是不允许的。
0643E	批处理中没有任何语句。	批处理没有任何语句。
22005	赋值出错。	参数类型与目标数据类型不兼容。
22011	发生了子串错误。	BLOB 值中要抽取的第一个字节的顺序位置无效。
S1010	函数顺序错误。	调用了 `CallableStatement` get 方法，但没有先调用 `registerOutParameter`。

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』

支持的 DB2 CLI 函数

本章描述 DB2 Everyplace 支持的“DB2 调用层接口”（DB2 CLI）函数。

- 『DB2 CLI 函数摘要』提供每个函数用途的简要描述以及 DB2 Everyplace 支持的 DB2 CLI 函数与标准 DB2 CLI 函数之间差异的简要摘要。
- 第 177 页的『DB2 CLI 函数描述的关键』提供每个 CLI 函数的函数描述的解释。
- 第 252 页的『由 DB2 CLI 函数进行的数据转换』包含一个表，该表列出了在 C 与 SQL 数据类型之间支持的数据转换

DB2 CLI 函数摘要

表 29 是有关 DB2 Everyplace 支持的 DB2 CLI 函数的摘要，包括每个函数的用途以及 DB2 Everyplace 支持的 DB2 CLI 函数与标准 DB2 CLI 函数之间差异的摘要。

表 29. DB2 CLI 函数列表

函数名	用途	差别摘要
SQLAllocConnect	获取连接句柄。	
SQLAllocEnv	获取环境句柄。	
SQLAllocHandle	获取句柄。	
SQLAllocStmt	分配语句句柄。	
SQLBindCol	指定结果列的存储器并指定数据类型。	目标类型限制为支持的数据类型。不支持 LOB 定位器。
SQLBindParameter	在 SQL 语句中为参数指定存储器。	不支持绑定至应用程序变量或 LOB 定位器的数组。因为不支持 SQLPutData()，所以在调用 SQLExecute() 之前应用程序应将参数的值放在 *ParameterValuePtr* 中。因为不支持存储过程，所以参数类型只被限定为 INPUT。
SQLColumns	返回指定表中的列名列表。	*CatalogName*、*NameLength1*、*SchemaName*、*NameLength2* 被忽略。返回的结果集的第 2、12 和 15 列总是 NULL。不支持返回码 SQL_STILL_EXECUTING。
SQLConnect	按数据源名、用户标识和密码连接特定驱动程序。	
SQLDescribeCol	描述结果集中的列。	列信息由支持的列数据类型限制。
SQLDisconnect	关闭连接。	
SQLEndTran	在与连接相关联的所有语句上为所有操作请求 COMMIT 或 ROLLBACK。	在调用 SQLEndTran() 之前，必须将连接属性 SQL_ATTR_AUTOCOMMIT 设置为 SQL_AUTOCOMMIT_OFF。

表 29. DB2 CLI 函数列表 (续)

函数名	用途	差别摘要
SQLError	返回其它错误或状态信息。	
SQLExecDirect	执行语句。	不支持返回码 SQL_STILL_EXECUTING 和 SQL_NEED_DATA。不支持异步 CLI 调用。
SQLExecute	执行已准备的语句。	在调用 SQLExecute() 之前必须绑定所有参数。不支持 SQL 调用的异步执行。
SQLFetch	返回结果行。	结果每次取装一行，而不是按行集取装。不支持语句描述符。不支持返回码 SQL_STILL_EXECUTING。
SQLFetchScroll	返回结果行集。	按行集取装结果。不支持返回码 SQL_STILL_EXECUTING。
SQLForeignKeys	返回关于指定表的外键的信息。	忽略 *PKCatalogName*、*NameLength1*、*PKSchemaName*、*NameLength2*、*FKCatalogName*、*NameLength4*、*FKSchemaName*、*NameLength5*。返回的结果集的第 1、2、5、6、12 和 13 列总是长度为零的字符串。返回的结果集的第 10、11和 14 列总是零。不支持返回码 SQL_STILL_EXECUTING。
SQLFreeConnect	释放连接句柄。	
SQLFreeEnv	释放环境句柄。	
SQLFreeHandle	释放句柄资源。	
SQLFreeStmt	结束语句处理，废弃暂挂结果，并可选择释放与语句句柄相关联的所有资源。	仅支持 SQL_DROP 和 SQL_RESET_PARAMS 选项。
SQLGetConnectAttr	返回连接属性的当前设置。	DB2 Everyplace 支持 DB2 支持的连接属性的子集。DB2 Everyplace 还支持一些 DB2 不支持的连接属性。
SQLGetCursorName	返回与语句句柄相关联的游标名。	内部生成的游标名总是以 CUR 开头。
SQLGetData	返回结果集一行中一列的部分或全部。	目标类型限制为支持的数据类型。不支持 LOB 定位器。不支持返回码 SQL_STILL_EXECUTING。

表 29. DB2 CLI 函数列表 (续)

函数名	用途	差别摘要
SQLGetDiagRec	获取诊断数据的多个字段。	仅支持与语句句柄或连接句柄相关联的诊断记录。仅支持单个诊断记录。
SQLGetInfo	返回有关特定驱动程序和数据源的信息。	DB2 Everyplace 支持 DB2 支持的信息类型的子集。
SQLGetStmtAttr	返回语句属性的当前设置。	DB2 Everyplace 支持 DB2 支持的语句属性的子集。DB2 Everyplace 还支持一些 DB2 不支持的语句属性。
SQLNumParams	返回 SQL 语句中的参数标记的数目。	不支持返回码 SQL_STILL_EXECUTING。
SQLNumResultCols	返回结果集中的列数。	
SQLPrepare	为稍后的执行准备 SQL 语句。	
SQLPrimaryKeys	返回由表的主键组成的列名的列表。	*CatalogName*、*NameLength1*、*SchemaName*、*NameLength2* 被忽略。返回的结果集的第 1、2 和 6 列总是长度为零的字符串。不支持返回码 SQL_STILL_EXECUTING。
SQLRowCount	返回受插入、更新或删除请求影响的行数。	
SQLSetConnectAttr	设置与连接相关的选项。	DB2 Everyplace 支持 DB2 支持的连接属性的子集。DB2 Everyplace 还支持一些 DB2 不支持的连接属性。
SQLSetStmtAttr	设置与语句相关的选项。	DB2 Everyplace 支持 DB2 支持的语句属性的子集。DB2 Everyplace 还支持一些 DB2 不支持的语句属性。
SQLTables	返回存储在特定数据源中的表名的列表。	将忽略 *CatalogName*、*NameLength1*、*SchemaName*、*NameLength2*、*TableType* 和 *NameLength4*。DB2 Everyplace 仅支持类型 “TABLE”。不支持返回码 SQL_STILL_EXECUTING。

相关参考:

- 第 252 页的『由 DB2 CLI 函数进行的数据转换』
- 『DB2 CLI 函数描述的关键』

DB2 CLI 函数描述的关键

每个函数描述包含下列各节:

用途 本节给出函数功能的简要概述。还指示在调用所描述的函数之前或之后是否应调用任何函数。

每个函数还有一个表，该表指示函数符合的规范或标准。

此表指示函数的支持。一些函数使用的选项集不适用于所有规范或标准。函数的限制部分标识了任何明显差异。

语法 本节包含类属 'C' 原型。该类属原型用于所有环境，包括 Windows。

所有作为指针的函数自变量都是使用宏 FAR 定义的，对于除 Windows 外的所有平台，此宏被定义为不存在（设置为空白）。在 Windows 上，FAR 用来将指针自变量定义为 far 指针。

自变量 本节列示每个函数自变量及其数据类型和描述，并指出它是输入自变量还是输出自变量。

一些函数包含输入或输出自变量（也称为*延迟*或*绑定*自变量）。

这些自变量是指向由应用程序分配的缓冲区的指针，且与 SQL 语句中的参数或结果集中的列相关联（或与这些参数或列绑定）。稍后，DB2 CLI 存取函数指定的数据区。在 DB2 CLI 存取这些延迟数据区时，它们必须仍然有效。

用法 本节提供有关如何使用函数的信息以及任何特殊注意事项。可能的错误状态不在此处讨论，而是列示在『诊断』一节中。

返回码 本节列示所有可能的函数返回码。当返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 时，可通过调用 SQLError() 或 SQLGetDiagRec() 来获取错误信息。

诊断 本节包含列示由 DB2 CLI 显式返回的 SQLSTATE（也可能返回由 DBMS 生成的 SQLSTATE）并指示错误原因的表。在函数返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO 后，可通过调用 SQLError() 或 SQLGetDiagRec() 来获取这些值。

限制 本节指示 DB2 Everyplace CLI 与 ODBC 之间可能影响应用程序的任何差异或限制。

请参阅 *IBM DB2 Universal Database Call Level Interface Guide and Reference* 以获取有关 DB2 CLI 的更多信息，包括有关返回码、诊断、示例、设置 CLI 环境和访问样本应用程序的信息。

相关参考:

- 第 252 页的『由 DB2 CLI 函数进行的数据转换』
- 第 175 页的『DB2 CLI 函数摘要』

SQLAllocConnect - 分配连接句柄

在 ODBC 版本 3 中，SQLAllocConnect() 已被废弃，而被替换为 SQLAllocHandle()；请参阅第 179 页的『SQLAllocHandle - 分配句柄』以获取更多信息。

建议: 虽然此版本的 DB2 CLI 仍然支持 SQLAllocConnect()，但请在 DB2 CLI 程序中使用 SQLAllocHandle()，以便符合最新标准。

迁移至新函数

例如，使用新的函数时，语句:

```
SQLAllocConnect(henv, hdbc);
```

将重写为:

```
SQLAllocHandle(SQL_HANDLE_DBC, henv, hdbc);
```

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLAllocEnv - 分配环境句柄

在 ODBC 版本 3 中，SQLAllocEnv() 已被废弃，且替换为 SQLAllocHandle()；请参阅『SQLAllocHandle - 分配句柄』以获取更多信息。

建议：虽然此版本的 DB2 CLI 继续支持 SQLAllocEnv()，但请在 DB2 CLI 程序中使用 SQLAllocHandle()，以便符合最新标准。

迁移至新函数

例如，使用新的函数时，语句:

```
SQLAllocEnv(henv);
```

将重写为:

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, henv);
```

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLAllocHandle - 分配句柄

用途:

规范:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLAllocHandle() 分配环境、连接或语句句柄。

此函数是分配句柄的类属函数，它替换废弃的版本 2 函数 SQLAllocConnect()、SQLAllocEnv() 和 SQLAllocStmt()。

语法:

```
SQLRETURN   SQLAllocHandle   (SQLSMALLINT       HandleType,
                              SQLHANDLE         InputHandle,
                              SQLHANDLE         *OutputHandlePtr);
```

函数自变量:

表 30. SQLAllocHandle 自变量

数据类型	自变量	使用	描述
SQLSMALLINT	*HandleType*	输入	将由 SQLAllocHandle() 分配的句柄类型。必须是下列其中一值: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT
SQLHANDLE	*InputHandle*	输入	将用作所分配新句柄的上下文的现有句柄。如果 *HandleType* 是 SQL_HANDLE_ENV，则此项为 SQL_NULL_HANDLE。如果 *HandleType* 是 SQL_HANDLE_DBC，则此项必须为环境句柄；而如果 *HandleType* 是 SQL_HANDLE_STMT，则此项必须为连接句柄。
SQLHANDLE	*OutputHandlePtr*	输出	指向缓冲区的指针，在该缓冲区中，将句柄返回至新分配的数据结构。

用法:

SQLAllocHandle() 用来分配环境、连接和语句句柄，如下所述。

应用程序一次可分配多个语句句柄。

如果应用程序调用 SQLAllocHandle()，且 **OutputHandlePtr* 设置为已存在的环境、连接、语句或描述符句柄，则 DB2 CLI 将覆盖与该句柄相关联的信息。DB2 CLI 并不检查在 **OutputHandlePtr* 中输入的句柄是否已在使用中，也不在覆盖句柄的先前内容之前检查它们。

对于 DB2 Everyplace，除语句句柄外的所有句柄都是哑句柄，并不包含可用的信息。

语句句柄提供对语句信息（如错误消息和 SQL 语句处理的状态信息）的访问。要请求语句句柄，应用程序连接数据源，然后在提交 SQL 语句之前调用 SQLAllocHandle()。在此调用中，应将 *HandleType* 设置为 SQL_HANDLE_STMT，并将 *InputHandle* 设置为调用分配该句柄的 SQLAllocHandle() 返回的连接句柄。DB2 CLI 分配语句句柄，将语句句柄与指定的连接相关联，并将相关联句柄的值传送回到 **OutputHandlePtr* 中。在需要语句句柄的所有后续调用中，应用程序都传送 **OutputHandlePtr* 值。

当应用程序退出时，将释放为该应用程序分配的所有 DB2 Everyplace 资源，因此该应用程序使用的句柄不再有效。

对于 DB2 Everyplace，没有任何描述符与具有应用程序可以更改的属性的语句句柄相关联。

在将 DB2 Everyplace 与 Visual Basic 和 DB2 Everyplace CLI/ODBC 接口配合使用时，必须显式调用在 sqlcli.h 中定义的映射 / 底层 CLI 函数。例如，调用 SQLAllocHandle() 会失败。调用 SQLAllocHandleVer(SQL_HANDLE_STMT, hdbc, hstmt, DB2eVersion) 会成功。

返回码:

- SQL_SUCCESS

- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

当分配的句柄不是环境句柄时，如果 SQLAllocHandle() 返回 SQL_ERROR，则它根据 *HandleType* 的值将 *OutputHandlePtr* 设置为 SQL_NULL_HENV、SQL_NULL_HDBC 或 SQL_NULL_HSTMT（除非输出自变量为空指针）。然后，应用程序可从与 *InputHandle* 自变量中的句柄相关联的诊断数据结构中获取附加信息。

诊断:

表 31. SQLAllocHandle SQLSTATE

SQLSTATE	描述	解释
01000	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
08003	连接已关闭。	*HandleType* 自变量是 SQL_HANDLE_STMT，但由 *InputHandle* 自变量指定的连接未打开。连接过程必须成功完成（且连接必须是打开的），DB2 CLI 才能分配语句句柄。
HY000	一般错误。	出错，该错误没有特定的 SQLSTATE。**MessageText* 缓冲区中由 SQLGetDiagRec() 返回的错误消息描述该错误及其原因。
HY001	内存分配出错。	DB2 CLI 无法为指定的句柄分配内存。
HY013	发生意外的内存处理错误。	*HandleType* 自变量是 SQL_HANDLE_DBC 或 SQL_HANDLE_STMT；因为未能访问底层内存对象（可能是因为内存不足），所以未能处理函数调用。
HY014	不再有句柄。	已达到可对 *HandleType* 自变量所指示的句柄类型分配的句柄数限制。
HY092	选项类型超出范围。	*HandleType* 自变量不是: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 215 页的『SQLFreeHandle - 释放句柄资源』

SQLAllocStmt - 分配语句句柄

在 ODBC 版本 3 中，SQLAllocStmt() 已被废弃，且替换为 SQLAllocHandle()；请参阅第 179 页的『SQLAllocHandle - 分配句柄』以获取更多信息。

建议：虽然此版本的 DB2 CLI 继续支持 SQLAllocStmt()，但请在 DB2 CLI 程序中使用 SQLAllocHandle()，以便符合最新标准。

迁移至新函数

例如，使用新的函数时，语句:

```
SQLAllocStmt(hdbc, hstmt);
```

将重写为:

```
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, hstmt);
```

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLBindCol - 将列绑定至应用程序变量

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

对于所有 C 数据类型，SQLBindCol() 用来使结果集中的列与应用程序变量相关联（绑定）。当调用 SQLFetch() 时，数据从 DBMS 被传送至应用程序。传送数据时，可能会发生数据转换。

将对结果集中应用程序需要检索的每一个列调用一次 SQLBindCol()。

通常，在调用此函数之前调用 SQLPrepare() 或 SQLExecDirect()，而在调用此函数之后调用 SQLFetch()。在调用 SQLBindCol() 之前，还可能需要列属性，可以使用 SQLDescribeCol() 来获取列属性。

语法:

```
SQLRETURN   SQLBindCol       (SQLHSTMT          StatementHandle,   /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLSMALLINT       TargetType,        /* fCType */
                SQLPOINTER        TargetValuePtr,    /* rgbValue */
                SQLINTEGER        BufferLength,      /* cbValueMax */
                SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

函数自变量:

表 32. SQLBindCol 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄
SQLUSMALLINT	*ColumnNumber*	输入	标识列的数字。列按顺序从左到右编号。列号从 1 开始。

表 32. SQLBindCol 自变量 (续)

数据类型	自变量	使用	描述
SQLSMALLINT	*TargetType*	输入	结果集中列号 *ColumnNumber* 的 C 数据类型。下列类型受支持: SQL_C_BINARY SQL_C_BIT SQL_C_CHAR SQL_C_DOUBLE SQL_C_FLOAT SQL_C_LONG SQL_C_SHORT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP SQL_C_TINYINT 指定 SQL_C_DEFAULT 会导致将数据转换为其缺省 C 数据类型。
SQLPOINTER	*TargetValuePtr*	输入 / 输出（延迟）	指向发生取装时 DB2 CLI 将存储列数据的缓冲区的指针。 如果 *TargetValuePtr* 为空，则取消绑定列。
SQLINTEGER	*BufferLength*	输入	可用来存储列数据的 *TargetValuePtr* 缓冲区的大小，以字节计。 如果 *TargetType* 指示二进制或字符串，或是 SQL_C_DEFAULT，则 *BufferLength* 必须大于 0，不然会返回错误。否则，此自变量被忽略。
SQLINTEGER *	*StrLen_or_IndPtr*	输入 / 输出（延迟）	指向一个值的指针，该值指示 DB2 CLI 可以在 *TargetValuePtr* 缓冲区中返回的字节数。 如果列的数据值为空，则 SQLFetch() 在此自变量中返回 SQL_NULL_DATA。 也可能返回 SQL_NO_LENGTH。有关信息，参阅『用法』一节。

对于此函数，*TargetValuePtr* 和 *StrLen_or_Ind* 都是延迟输出，这表示在取装结果集行之前，不会更新这些指针所指向的存储位置。因此，在调用 SQLFetch() 之前，这些指针所引用的位置必须仍然有效。例如，如果在本地函数中调用 SQLBindCol()，则必须从该函数的相同作用域中调用 SQLFetch()，或者必须将 *TargetValuePtr* 缓冲区分配或声明为静态缓冲区或全局缓冲区。

用法:

应用程序将对结果集中要检索其数据的每一列调用一次 SQLBindCol()。结果集是通过调用 SQLExecute() 或 SQLExecDirect() 生成的。当调用 SQLFetch() 时，这些*绑定*列的每一列中的数据将放置在指定位置（由指针 *TargetValuePtr* 和 *StrLen_or_Ind* 给定）。

各个列均由一个号码标识，此号码是从左到右按顺序指定的。列号从 1 开始。

结果集中的列数可以通过调用 SQLNumResultCols() 确定。

应用程序可以通过首先调用 SQLDescribeCol() 来查询列的属性（如数据类型和长度）。然后，可使用此信息来分配正确数据类型和长度的存储位置，以指示至另一数据类型的数据转换。

应用程序可以选择不绑定每个列，甚至不绑定任何列。在对当前行取装绑定的列之后，还可使用 SQLGetData() 来检索任何列中的数据。

在进行后续取装时，应用程序可以通过调用 SQLBindCol() 来更改这些列的绑定或绑定先前取消了绑定的列。新的绑定不适用于已取装的数据，它将在下次取装时使用。要取消绑定单个列，调用 SQLBindCol()，并将 *TargetValuePtr* 指针设置为 NULL。要取消绑定所有列，应用程序应调用 SQLFreeStmt()。

应用程序必须确保为要检索的数据分配足够的存储器。如果缓冲区要用来包含变长数据，则应用程序分配的存储器必须与绑定列所需的最大长度一样多，否则，数据可能会被截断。如果缓冲区要用来包含定长数据，则 DB2 CLI 假定缓冲区的大小就是 C 数据类型的长度。如果指定了数据转换，则所需的大小可能会受到影响。

如果发生字符串截断，则返回 SQL_SUCCESS_WITH_INFO，并把 *StrLen_or_IndPtr* 设置成可返回给应用程序的 *TargetValuePtr* 的实际大小。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 33. SQLBindCol SQLSTATE

SQLSTATE	描述	解释
07009	描述符索引无效。	对 *ColumnNumber* 自变量指定的值超出结果集中的最大列数。
40003 **08**S01	通信链路错误。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY002	列号无效。	对 *ColumnNumber* 自变量指定的值小于 0。 对 *ColumnNumber* 自变量指定的值超过数据源支持的最大列数。
HY003	程序类型超出范围。	*TargetType* 不是有效的数据类型或 SQL_C_DEFAULT。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY090	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 1，而 *TargetType* 自变量是 SQL_C_CHAR、SQL_C_BINARY 或 SQL_C_DEFAULT。
HYC00	驱动程序不起作用。	DB2 CLI 识别但不支持 *TargetType* 自变量中指定的数据类型。

取装时可能会报告与绑定列相关的其它诊断消息。

限制:

输出缓冲区需要字对齐（平行）。许多处理器（如 Motorola 68000）有字对齐的规则，对于非字符数据类型，应用程序应正确地将缓冲区对齐。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLBindParameter - 将参数标记绑定至缓冲区

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 2.0	

对于所有 C 数据类型，SQLBindParameter() 用来使 SQL 语句中的参数标记与应用程序变量相关联（绑定）。在这种情况下，调用 SQLExecute() 或 SQLExecDirect() 时，数据是从应用程序传送至 DBMS 的。传送数据时，可能会发生数据转换。

语法:

```
SQLRETURN  SQL_API SQLBindParameter(
                    SQLHSTMT          StatementHandle,  /* hstmt */
                  SQLUSMALLINT      ParameterNumber,  /* ipar */
                  SQLSMALLINT       InputOutputType,  /* fParamType */
                  SQLSMALLINT       ValueType,        /* fCType */
                  SQLSMALLINT       ParameterType,    /* fSqlType */
                  SQLUINTEGER       ColumnSize,       /* cbColDef */
                  SQLSMALLINT       DecimalDigits,    /* ibScale */
                  SQLPOINTER        ParameterValuePtr,/* rgbValue */
                  SQLINTEGER        BufferLength,      /* cbValueMax */
                  SQLINTEGER *FAR   StrLen_or_IndPtr);/* pcbValue */
```

函数自变量:

表 34. SQLBindParameter 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLUSMALLINT	*ParameterNumber*	输入	参数标记号码从左到右，从 1 开始依次编号。
SQLSMALLINT	*InputOutputType*	输入	参数类型。支持的类型是: • SQL_PARAM_INPUT: 执行此语句时，参数的实际数据值被发送至服务器; *ParameterValuePtr* 缓冲区必须包含有效的输入数据值，而 *StrLen_or_IndPtr* 缓冲区必须包含相应的长度值，或者 SQL_NTS 或 SQL_NULL_DATA。 DB2 Everyplace 不支持 SQLPutData()，所以不应将参数值存储在 *ParameterValuePtr* 缓冲区中。 • SQL_PARAM_INPUT_OUTPUT: 参数标记与所调用的存储过程的输入／输出参数相关联。执行语句时，参数的实际数据值被送至服务器。*ParameterValuePtr* 缓冲区必须包含有效的输入数据值; *StrLen_or_IndPtr* 缓冲区必须包含相应的长度值或 SQL_NTS、SQL_NULL_DATA。 • SQL_PARAM_OUTPUT: 参数标记与所调用的存储过程的输出参数或者该存储过程的返回值相关联。 执行该语句之后，输出参数的数据被返回到由 *ParameterValuePtr* 和 *StrLen_or_IndPtr* 指定的应用程序缓冲区中，除非这两个指针都是空指针，在这种情况下，输出数据将被废弃。如果输出参数没有返回值，则 *StrLen_or_IndPtr* 被设置为 SQL_NULL_DATA。

表 34. SQLBindParameter 自变量 (续)

数据类型	自变量	使用	描述
SQLSMALLINT	*ValueType*	输入	参数的 C 数据类型。下列类型受支持: • SQL_C_BINARY • SQL_C_BIT • SQL_C_CHAR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TINYINT 指定 SQL_C_DEFAULT 会导致将数据从其缺省 C 数据类型转换为 *ParameterType* 中指示的类型。
SQLSMALLINT	*ParameterType*	输入	参数的 SQL 数据类型。支持的类型是: • SQL_BLOB • SQL_CHAR • SQL_DECIMAL • SQL_INTEGER • SQL_SMALLINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARCHAR
SQLUINTEGER	*ColumnSize*	输入	对应的参数标记的精度。 • 如果 *ParameterType* 指示二进制或单字节字符串（如 SQL_CHAR、SQL_BLOB），则这是此参数标记的最大长度（以字节计）。 • 否则，此自变量被忽略。
SQLSMALLINT	*DecimalDigits*	输入	（相应参数的小数位数）如果 *ParameterType* 是 SQL_DECIMAL。
SQLPOINTER	*ParameterValuePtr*	输入（延迟）和/或输出（延迟）	• 在输入时（*InputOutputType* 设置为 SQL_PARAM_INPUT 或 SQL_PARAM_INPUT_OUTPUT）: 在执行时，如果 *StrLen_or_IndPtr* 未包含 SQL_NULL_DATA，则 *ParameterValuePtr* 指向包含参数实际数据的缓冲区。 • 在输出时（InputOutputType 设置为 SQL_PARAM_OUTPUT 或 SQL_PARAM_INPUT_OUTPUT）: *ParameterValuePtr* 指向存储存储过程的输出参数值的缓冲区。 • 空 *ParameterValuePtr* 指示取消绑定参数。
SQLINTEGER	*BufferLength*	输入	对于字符和二进制数据，*BufferLength* 指定 *ParameterValuePtr* 缓冲区的长度。对于非字符和非二进制数据，此自变量被忽略，且 *ParameterValuePtr* 缓冲区的长度被假定为与 C 数据类型相关联的长度。对于输出参数，*BufferLength* 被用来确定是否截断数据。

表 34. SQLBindParameter 自变量 (续)

数据类型	自变量	使用	描述
SQLINTEGER *	*StrLen_or_IndPtr*	输入（延迟）和/或输出（延迟）	• 如果这是输入或输入/输出参数：这是指向（在执行语句时）包含在 *ParameterValuePtr* 处存储的参数标记值长度的位置的指针。 要对参数标记符指定空值，此存储位置必须包含 SQL_NULL_DATA。 如果 *ValueType* 是 SQL_C_CHAR，则此存储位置必须包含在 *ParameterValuePtr* 处存储的数据的精确长度，或包含 SQL_NTS（如果 *ParameterValuePtr* 处的内容以 null 结束的话）。如果它包含确切的长度，则存储在 ParameterValuePtr 处的数据中不允许空字符。 如果 *ValueType* 指示字符数据（显式指定或使用 SQL_C_DEFAULT 隐式指定），且此指针设置为 NULL，则应用程序必须在 *ParameterValuePtr* 中提供以 null 结束的字符串。这也表示此参数标记永远不会具有空值。 • 如果这是输出参数（*InputOutputType* 设置为 SQL_PARAM_OUTPUT）：在执行存储过程之后，这必须是存储过程 CALL 的输出参数或返回值，并且必须指向下列其中一项: – 可在 *ParameterValuePtr* 中返回的字节数（null 终止字符除外）。 – SQL_NULL_DATA

用法:

在 SQL 语句中，参数标记由 ? 字符表示，用来指示语句中的一个位置，在该处，应用程序提供的值将在执行该语句时被替换。此值可以从应用程序变量获取。SQLBindParameter() 用来将应用程序存储区与参数标记绑定。

在执行 SQL 语句之前，应用程序必须将一个变量与 SQL 语句中的每个参数标记绑定。对于此函数，*ParameterValuePtr* 和 *StrLen_or_IndPtr* 是延迟自变量。执行该语句时，存储位置必须有效且包含输入数据值。这表示 SQLExecDirect() 或 SQLExecute() 调用都必须保存在与 SQLBindParameter() 调用相同的过程作用域中，或者这些存储位置必须是动态分配的，或是以静态方式或全局方式声明的。

参数标记按编号（*ColumnNumber*）引用，并且是从 1 开始从左到右按顺序编号的。

在调用下列其中一个函数之前，此函数绑定的所有参数仍然有效:

- 调用 SQLFreeStmt()，并指定 SQL_RESET_PARAMS 选项。
- 调用 SQLFreeHandle()，并将 *HandleType* 设置为 SQL_HANDLE_STMT。
- 再次对同一参数 *ParameterNumber* 编号调用 SQLBindParameter()。

在执行 SQL 语句并处理结果之后，应用程序可能想重新使用该语句句柄来执行另一 SQL 语句。如果参数标记规范不同（参数数目、长度或类型不同），则必须使用 SQL_RESET_PARAMS 调用 SQLFreeStmt()，以复位或清除参数绑定。

ValueType 给出的 C 缓冲区数据类型必须与 *ParameterType* 指示的 SQL 数据类型相兼容，否则会出错。

因为在执行语句之前，不会验证 *ParameterValuePtr* 和 *StrLen_or_IndPtr* 所引用的变量中的数据，所以在调用 SQLExecute() 或 SQLExecDirect() 之前，不会检测或报告数据内容错误或格式错误。

对于此函数，*ParameterValuePtr* 和 *StrLen_or_IndPtr* 是延迟自变量。在 *InputOutputType* 设置为 SQL_PARAM_INPUT 的情况下，当执行语句时，存储位置必须有效且包含输入

数据值。这表示 SQLExecDirect() 或 SQLExecute() 调用都必须保存在与 SQLBindParameter() 调用相同的过程作用域中，或者这些存储位置必须是动态分配的，或是以静态方式或全局方式声明的。

DB2 Everyplace 支持 SQL_PARAM_INPUT、SQL_PARAM_INPUT_OUTPUT 和 SQL_PARAM_OUTPUT。DB2 Everyplace 不支持 SQLPutData()，所以不应将参数值存储在 *ParameterValuePtr* 缓冲区中。

对于字符和二进制 C 数据，*BufferLength* 自变量指定 *ParameterValuePtr* 缓冲区的长度。对于所有其它类型的 C 数据，*BufferLength* 自变量被忽略。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 35. SQLBindParameter SQLSTATE

SQLSTATE	描述	解释
07006	转换无效。	从 *ValueType* 自变量标识的数据值到 *ParameterType* 自变量标识的数据类型的转换不是有意义的转换。（例如，从 SQL_C_DATE 转换至 SQL_DOUBLE。）
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY003	程序类型超出范围。	*ParameterNumber* 自变量指定的值不是有效的数据类型或 SQL_C_DEFAULT。
HY004	SQL 数据类型超出范围。	对 *ParameterType* 自变量指定的值不是有效的 SQL 数据类型。
HY009	自变量值无效。	*ParameterValuePtr* 自变量是空指针，且 *StrLen_or_IndPtr* 自变量是空指针，而 *InputOutputType* 不是 SQL_PARAM_OUTPUT。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY090	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 0。
HY093	参数号无效。	对 *ValueType* 自变量指定的值小于 1 或大于服务器支持的最大参数数目。
HY094	小数位值无效。	对 *ParameterType* 指定的值是 SQL_DECIMAL 或 SQL_NUMERIC，对 *DecimalDigits* 指定的值小于 0 或大于 *ParamDef* 自变量（精度）的值。

表 35. SQLBindParameter SQLSTATE (续)

SQLSTATE	描述	解释
HY104	精度值无效。	对 *ParameterType* 指定的值是 SQL_DECIMAL 或 SQL_NUMERIC，对 *ParamDef* 指定的值小于 1。
HY105	参数类型无效。	*InputOutputType* 不是 SQL_PARAM_INPUT。
HYC00	驱动程序不起作用。	DB2 CLI 或数据源不支持由对 *ValueType* 自变量指定的值和对 *ParameterType* 祖变量指定的值的组合所指定的转换。 对 *ParameterType* 自变量指定的值不受 DB2 CLI 或数据源支持。

相关参考:

- 第 160 页的『赋值和比较的数据类型兼容性』
- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 202 页的『SQLExecute - 执行语句』

SQLConnect - 连接至数据源

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	**ISO CLI**

SQLConnect() 建立与目标数据库的连接。

在调用此函数之前，必须使用 SQLAllocHandle() 分配连接句柄。

在使用 SQLAllocHandle() 分配语句句柄之前，必须调用此函数。

语法:

```
SQLRETURN   SQLConnect      (
                SQLHDBC          ConnectionHandle,  /* hdbc */
                SQLCHAR      *FAR ServerName,       /* szDSN */
                SQLSMALLINT      NameLength1,       /* cbDSN */
                SQLCHAR      *FAR UserName,         /* szUID */
                SQLSMALLINT      NameLength2,       /* cbUID */
                SQLCHAR      *FAR Authentication,   /* szAuthStr */
                SQLSMALLINT      NameLength3);      /* cbAuthStr */
```

函数自变量:

表 36. SQLConnect 自变量

数据类型	自变量	使用	描述
SQLHDBC	*ConnectionHandle*	输入	连接句柄。
SQLCHAR *	*ServerName*	输入	数据库的位置和名称。名称是可选的。DB2 Everyplace 忽略此名称。
SQLSMALLINT	*NameLength1*	输入	*ServerName* 自变量的内容的长度。
SQLCHAR *	*UserName*	输入	授权名（用户标识）。此字符串用于加密，否则 DB2 Everyplace 忽略它。

表 36. SQLConnect 自变量 (续)

数据类型	自变量	使用	描述
SQLSMALLINT	*NameLength2*	输入	*UserName* 自变量的内容的长度。
SQLCHAR *	*Authentication*	输入	认证字符串（密码）。此字符串用于加密，否则 DB2 Everyplace 忽略它。
SQLSMALLINT	*NameLength3*	输入	*Authentication* 自变量的内容的长度。

注意事项:

*未注册的*用户（在 DB2eSYSUSERS 表中不存在的用户）当在 SQLGetDiagRec() CLI 函数调用期间试图连接到加密的 DB2 Everyplace 数据库时，将接收到警告消息 42704（未定义对象）。*已注册*的用户将不会接收到此警告。与此相反，在 SQLConnect() 函数调用期间，未注册和已注册的用户都能够连接至数据库，而不会接收到警告消息。

用法:

`SQLConnect()` 可用来连接至不同位置的数据源。

要访问本地设备上的数据源，应将 *ServerName* 自变量设置为数据源名称。DB2 Everyplace 会忽略数据源名称，而访问本地数据源。

对于使用辅助存储设备的应用程序，*ServerName* 自变量接受指向在本地存在的 *DataSource* 位置的字符串，或接受在支持的辅助存储设备（如 IBM Microdrive、Sony Memory Stick、Compact Flash、SD Memory Card 或 MultiMediaCard）上的存储位置的字符串。*ServerName* 字符串的格式是:

ServerName=Device:/Path/DataSource

Device 这是在其上存储 *DataSource* 的设备的名称。保留字符 `#` 用来访问任何 Compact Flash (CF) Type II 存储设备（在具有 CF 支持的 Palm OS 设备上）。辅助存储器是使用保留字符 `#` 进行寻址的。`#0` 和 `#1` 指定要访问哪个辅助存储器槽。`#` 相当于 `#0`。例如:

```
ServerName=#:/storage/
```

DB2 Everyplace 连接至第一个 CF 槽中的 IBM Microdrive 的 `storage` 目录中的 *DataSource*。

Path 这是 *Device* 上 *DataSource* 的路径。当指定 *Path* 而未指定 *Device:/* 时，使用相对于应用程序位置的本地文件系统路径。不应将文件写入卷的根目录中。某些介质类型不支持在根目录中放入文件。例如:

```
ServerName=dir1/dir2/DATA1
```

注: DB2 Everyplace 中没有路径长度限制。
如果应用程序位于本地文件系统上的 `/myapp` 中，则 DB2 Everyplace 连接至 `/myapp/dir1/dir2/` 中的 *DataSource*。*DataSource* 名 DATA1 被忽略。

DataSource
可选: 要连接的数据源的名称。DB2 Everyplace 忽略此名称。

要使用“远程查询和存储过程”适配器访问远程存储过程，使用 *ServerName* 自变量来标识数据库的位置和名称。对于使用“远程查询”和“存储过程”适配器来存取远程数据库的应用程序，*ServerName* 自变量接受以下 URL 格式:

```
http://IPAddress:portNumber/path?DB=DataSource
```

IPAddress 和 *Authentication* 是必需的。

如果使用 Windows CE 对象存储器而不是常规文件系统，则:

- 将 CLI 函数 SQLConnect 的路径参数设置为 "@:\"

或

- 在 CLP 中，执行“connect to @:\”

Windows CE 对象存储器中没有“目录”概念。当使用对象存储器时，用户不能定义在其中创建表的目录或路径。在相同的名称空间中创建对象存储器中的所有表。由于此限制，不能同时建立多个与对象存储器的连接。在文件系统的根路径中创建连接串行化目的的锁文件。

当使用对象存储器时，与使用常规文件系统时不同，不能手工删除 DB2 Everyplace 文件。

示例:

连接至以本地方式存在于 c:\dir1\dir2\ 中的数据源。数据源名称 DS1 被忽略:

```
ServerName=c:/dir1/dir2/DS1
```

连接至使用 UNIX 文件系统表示法且以本地方式存在于 /dir1/dir2/ 中的数据源:

```
ServerName=/dir1/dir2/
```

连接至以本地方式存在于相对于应用程序路径的 dir1\ 目录中的数据源。如果应用程序位于 c:\myapp\ 中，则访问 c:\myapp\dir1\ 数据源:

```
ServerName=dir1\
```

连接至辅助存储器槽 1 中的存储内存上的 /dir1/ 目录中的数据源:

```
ServerName=#1:/dir1/
```

使用远程查询和存储过程适配器连接至端口 8080 上的 DB2 Everyplace Sync Server 192.168.0.1 以及 mysample 数据库。

```
ServerName=
 http://192.168.0.1:8080/db2e/servlet/com.ibm.mobileservices.adapter
                .agent.AgentServlet?DB=mysample
```

使用 Windows CE 存储器连接到数据源。

```
ServerName=@:\
```

连接串行化:

有关连接串行化的信息，请参阅第 63 页的『连接串行化』。

连接认证:

数据库加密需要基本用户认证。DB2 Everyplace 使用“用户名”和“认证”来在连接时认证用户。

认证按如下方式工作: 如果 DB2eSYSUSERS 目录表在 SQLConnect 连接至的数据库中不存在，则忽略“用户名”和“认证”信息。DB2 Everyplace 将*已注册*和*未注册*用户区

别开来。已注册用户是通过 GRANT SQL 语句添加的且在 DB2eSYSUSERS 表中列示的用户。连接时，如果有 DB2eSYSUSERS 表且“用户名”属于已注册用户，则尝试认证。如果“认证”参数中给定的密码不正确，则返回错误（42505）。如果“用户名”是未注册的，则 SQLConnect 函数将成功。然而，对 SQLGetDiagRec 的后续调用将返回警告 42704（未定义对象）。这允许应用程序区分成功连接的已注册用户和成功连接的未注册用户。有关更多信息，请参阅第 75 页的『本地数据加密的概述』、314 和第 143 页的『GRANT』。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 37. SQLConnect SQLSTATE

SQLSTATE	描述	解释
08001	无法连接数据源。	DB2 CLI 无法建立与数据源（服务器）的连接。
08002	连接正在使用中。	已使用指定的 *ConnectionHandle* 来建立与数据源的连接，该连接仍处于打开状态。
08004	应用程序服务器拒绝建立连接。	数据源（服务器）拒绝建立连接。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY501	*DataSource* 名无效。	指定的 *DataSource* 名无效。
HYT00	连接超时到期。	应用程序能够连接到数据源之前超时时间到期。可以使用 *SQLSetConnectAttr()* 的 SQL_ATTR_LOGIN_TIMEOUT 属性设置超时时间。当另一个应用程序正在使用数据库时返回此错误。

限制:

在可以执行任何 SQL 语句之前，必须调用 `SQLConnect()`。

相关概念:

- 第 63 页的『连接串行化』
- 第 179 页的『SQLAllocHandle - 分配句柄』
- 第 198 页的『SQLDisconnect - 与数据源断开连接』

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 179 页的『SQLAllocHandle - 分配句柄』
- 第 198 页的『SQLDisconnect - 与数据源断开连接』

SQLColumns – 获取表的列信息

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLColumns() 返回指定表中的列的列表。此信息在 SQL 结果集中返回，可以使用用来取装查询生成的结果集的函数来检索此结果集。

语法:

```
SQLRETURN   SQLColumns      (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR      FAR  *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR      FAR  *SchemaName,       /* szSchemaName */
               SQLSMALLINT       NameLength2,       /* cbSchemaName */
               SQLCHAR      FAR  *TableName,        /* szTableName */
               SQLSMALLINT       NameLength3,       /* cbTableName */
               SQLCHAR      FAR  *ColumnName,       /* szColumnName */
               SQLSMALLINT       NameLength4);      /* cbColumnName */
```

函数自变量:

表 38. SQLColumns 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLCHAR	*CatalogName*	输入	可能包含用来限定结果集的 *pattern-value* 的缓冲区。*Catalog* 是由 3 部分组成的表名的第 1 部分。 此自变量将被 DB2 Everyplace 忽略。
SQLSMALLINT	*NameLength1*	输入	*CatalogName* 的长度。 此自变量将被 DB2 Everyplace 忽略。
SQLCHAR	*SchemaName*	输入	可能包含用来按模式名限定结果集的 *pattern-value* 的缓冲区。 此自变量将被 DB2 Everyplace 忽略。
SQLSMALLINT	*NameLength2*	输入	*SchemaName* 的长度。 此自变量将被 DB2 Everyplace 忽略。
SQLCHAR	*TableName*	输入	可能包含用来按表名限定结果集的 *pattern-value* 的缓冲区。
SQLSMALLINT	*NameLength3*	输入	*TableName* 的长度。
SQLCHAR	*ColumnName*	输入	可能包含用来按列名限定结果集的 *pattern-value* 的缓冲区。
SQLSMALLINT	*NameLength4*	输入	*ColumnName* 的长度。

用法:

将调用此函数以检索有关一个表或一组表的列的信息。典型应用程序可能希望在调用 SQLTables() 后调用此函数以确定表的各列。应用程序应使用 SQLTables() 结果集的 TABLE_NAME 中返回的字符串作为此函数的输入。

SQLColumns() 返回标准结果集，按 TABLE_NAME 和 ORDINAL_POSITION 排序。194 列示结果集中的各列。

TableName 和 *ColumnName* 自变量将接受搜索模式。

此函数不返回有关结果集中的各列的信息。应使用 SQLDescribeCol() 或 SQLColAttribute() 代替。

最好不要经常调用 SQLColumns()，原因是在许多情况下它们都映射至对系统目录的复杂且高成本的查询。应保存结果而不是重复调用。

已使用最大长度属性 128 声明目录函数结果集的 VARCHAR 列，以便与 SQL92 限制一致。因为 DB2 名称少于 128 个字符，所以应用程序可选择总是为输出缓冲区保留 128 个字符（加上 null 终止符），或者使用 SQL_MAX_TABLE_NAME_LEN 调用 SQLGetInfo() 以分别确定连接的 DBMS 支持的 TABLE_NAME 和 COLUMN_NAME 列的实际长度。

虽然可以添加新列，将来的发行版中也可以更改现有列的名称，但当前列的位置不会改变。

SQLColumns 返回的列:

第 1 列 TABLE_CAT (VARCHAR(128))
这始终是 NULL。

第 2 列 TABLE_SCHEM (VARCHAR(128))
这始终是 NULL。

第 3 列 TABLE_NAME (VARCHAR(128) not NULL)
表名。

第 4 列 COLUMN_NAME (VARCHAR(128) not NULL)
列标识。指定表、视图、别名或同义词的列的名称。

第 5 列 DATA_TYPE (SMALLINT not NULL)
由 COLUMN_NAME 标识的列的 SQL 数据类型。这是第 162 页的『SQL 符号和缺省数据类型』中的“符号 SQL 数据类型”列的值之一。

第 6 列 TYPE_NAME (VARCHAR(128) not NULL)
表示与 DATA_TYPE 对应的数据类型的名称的字符串。

第 7 列 COLUMN_SIZE (INTEGER)

如果 DATA_TYPE 列值表示字符或二进制字符串，则此列包含该列的最大长度（以字符计）。

对于 DATE、TIME 或 TIMESTAMP 数据类型，这是在转换为字符时显示该值所需的总字符数。

对于数字数据类型，这是该列中允许的总位数。

另请参阅第 162 页的『数据类型属性』。

第 8 列 BUFFER_LENGTH (INTEGER)
如果在 SQLBindCol()、SQLGetData() 和 SQLBindParameter() 调用上指定了 SQL_C_DEFAULT，则为要存储此列中的数据的相关 C 缓冲区的最大字节数。此长度不包括任何 null 终止符。对于确切数字数据类型，在长度上需要考虑小数点和符号。另请参阅第 162 页的『数据类型属性』

第 9 列 DECIMAL_DIGITS (SMALLINT)
列的小数位。对于小数位不适用的数据类型，将返回 NULL。另请参阅第 162 页的『数据类型属性』

第 10 列 NUM_PREC_RADIX (SMALLINT)

10 或 NULL。

如果 DATA_TYPE 是确切的数字数据类型，则此列包含值 10，而 COLUMN_SIZE 包含该列允许的小数位数。

对于数字数据类型，DBMS 返回 NUM_PREC_RADIX 为 10。

对于基不适用的数据类型，将返回 NULL。

第 11 列 NULLABLE (SMALLINT not NULL)

如果该列不接受 NULL 值，则为 SQL_NO_NULLS。

如果该列接受 NULL 值，则为 SQL_NULLABLE。

第 12 列 REMARKS (VARCHAR(254))

这始终是 NULL。

第 13 列 COLUMN_DEF (VARCHAR(254))

列的缺省值。如果缺省值为数字文字，则此列将包含数字文字的字符表示，没有用单引号括起来。如果缺省值为字符串，则此列即用单引号括起来的该字符串。如果缺省值为伪文字（例如，DATE、TIME 和 TIMESTAMP 列），则此列将包含伪文字的关键字（例如，CURRENT DATE），没有用单引号括起来。

如果将 NULL 指定为缺省值，则此列将返回字 NULL，未括在括号中。如果未指定缺省值，则此列为 NULL。

第 14 列 SQL_DATA_TYPE (SMALLINT not NULL)

此列与 DATA_TYPE 列相同。

第 15 列 SQL_DATETIME_SUB (SMALLINT)

此列始终是 NULL。

第 16 列 CHAR_OCTET_LENGTH (INTEGER)

包含字符数据类型列的最大长度（以八位字节计）。对于“单字节”字符集，它与 COLUMN_SIZE 相同。对于所有其它数据类型，它为 NULL。

第 17 列 ORDINAL_POSITION (INTEGER not NULL)

列在表中的顺序位置。表中的第一列编号为 1。

第 18 列 IS_NULLABLE (VARCHAR(254))

如果该列已知是不可空的，则包含字符串“NO”；否则包含字符串“YES”。

此结果集与 X/Open CLI *Columns()* 结果集规范完全相同，后者是在 ODBC V2 中指定的 *SQLColumns()* 结果集的扩展版本。ODBC *SQLColumns()* 结果集在相同位置包括每一列。

注意: 此结果集与 X/Open CLI `Columns()` 结果集规范完全相同，后者是在 ODBC V2 中指定的 `SQLColumns()` 结果集的扩展版本。ODBC `SQLColumns()` 结果集在相同位置包括每一列。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 39. SQLColumns SQLSTATE

SQLSTATE	描述	解释
24000	游标状态无效。	已在语句句柄上打开游标。
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效。	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。

限制:

无。

相关参考:

• 第 250 页的『SQLTables – 获取表信息』

SQLDescribeCol – 返回列的一组属性

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLDescribeCol() 返回查询所生成的结果集中指示列的一组常用描述符信息（列名、类型、精度、小数位和可空性）。

在调用此函数之前，必须调用 SQLPrepare() 或 SQLExecDirect()。

通常在调用绑定列函数（SQLBindCol()）之前调用此函数，以便在将一个列与应用程序变量绑定之前确定它的属性。

语法:

```
SQLRETURN   SQLDescribeCol   (
                  SQLHSTMT          StatementHandle,  /* hstmt */
                SQLUSMALLINT      ColumnNumber,      /* icol */
                SQLCHAR       *FAR ColumnName,        /* szColName */
                SQLSMALLINT       BufferLength,      /* cbColNameMax */
                SQLSMALLINT   *FAR NameLengthPtr,     /* pcbColName */
                SQLSMALLINT   *FAR DataTypePtr,       /* pfSqlType */
                SQLUINTEGER   *FAR ColumnSizePtr,     /* pcbColDef */
                SQLSMALLINT   *FAR DecimalDigitsPtr,  /* pibScale */
                SQLSMALLINT   *FAR NullablePtr);      /* pfNullable */
```

函数自变量:

表 40. SQLDescribeCol 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLUSMALLINT	*ColumnNumber*	输入	要描述的列号。列从 1 开始按顺序从左到右编号。
SQLCHAR *	*ColumnName*	输出	指向列名缓冲区的指针。如果不能确定列名，则此自变量设置为 NULL。
SQLSMALLINT	*BufferLength*	输入	*ColumnName* 缓冲区的大小。

表 40. SQLDescribeCol 自变量 (续)

数据类型	自变量	使用	描述
SQLSMALLINT *	*NameLengthPtr*	输出	可为 *ColumnName* 自变量返回的字节数。如果 *NameLengthPtr* 大于或等于 *BufferLength*，则将列名 (*ColumnName*) 截断为 *BufferLength - 1* 个字节。
SQLSMALLINT *	*DataTypePtr*	输出	列的基本 SQL 数据类型。
SQLUINTEGER *	*ColumnSizePtr*	输出	在数据库中所定义的列精度。
SQLSMALLINT *	*DecimalDigitsPtr*	输出	在数据库中所定义的列的小数位（仅适用于 SQL_DECIMAL）。
SQLSMALLINT *	*NullablePtr*	输出	指示此列是否允许 NULL。下列两项中的任意一项: SQL_NO_NULLS SQL_NULLABLE

用法:

各个列由一个号码标识，并从左到右按顺序编号，且可以任何顺序进行描述。列号从 1 开始。

如果对任何指针自变量指定空指针，则 DB2 CLI 假定应用程序不需要该信息，且不返回任何信息。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

如果 `SQLDescribeCol()` 返回 SQL_ERROR 或 SQL_SUCCESS_WITH_INFO，则通过调用 `SQLError()` 函数可能会获得下列其中一个 SQLSTATE。

表 41. SQLDescribeCol SQLSTATE

SQLSTATE	描述	解释
01004	数据被截断。	*ColumnName* 自变量中返回的列名比 *BufferLength* 自变量中指定的值要长。*NameLengthPtr* 自变量包含完整列名的长度。（函数返回 SQL_SUCCESS_WITH_INFO。）
07005	语句未返回结果集。	与 *StatementHandle* 相关联的语句未返回结果集。没有要描述的列。（首先调用 `SQLNumResultCols()` 以确定结果集中是否有任何行。）
07009	描述符索引无效	对 *ColumnNumber* 指定的值等于或小于 0。对 *ColumnNumber* 自变量指定的值大于结果集中的列数。
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY002	列号无效。	对自变量 *ColumnNumber* 指定的值小于 1，或者对自变量 *ColumnNumber* 指定的值大于结果集中的列数。
HY090	字符串或缓冲区长度无效。	自变量 *BufferLength* 中指定的长度小于 1。
HY010	函数顺序错误。	在对 *StatementHandle* 调用 `SQLPrepare()` 或 `SQLExecDirect()` 之前调用了该函数。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HYC00	驱动程序不起作用。	DB2 CLI 不识别 *ColumnNumber* 列的 SQL 数据类型。

限制:

DB2 Everyplace 只支持 ODBC 定义的下列数据类型:

- SQL_BLOB
- SQL_CHAR

- SQL_DECIMAL
- SQL_INTEGER
- SQL_SMALLINT
- SQL_TYPE_DATE
- SQL_TYPE_TIME
- SQL_TYPE_TIMESTAMP
- SQL_VARCHAR

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 235 页的『SQLNumResultCols - 获取结果列数』

SQLDisconnect - 与数据源断开连接

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLDisconnect() 关闭与数据库连接句柄相关联的连接。

在调用此函数之后，调用 SQLConnect() 来连接另一个数据库，或调用 SQLFreeHandle()。

语法:

```
SQLRETURN   SQLDisconnect    (SQLHDBC          ConnectionHandle;) /* hdbc */
```

函数自变量:

表 42. SQLDisconnect 自变量

数据类型	自变量	使用	描述
SQLHDBC	*ConnectionHandle*	输入	连接句柄。

用法:

如果应用程序在释放与连接相关联的所有语句句柄之前调用 SQLDisconnect()，则 DB2 CLI 在成功与数据库断开连接之后释放这些句柄。

如果返回了 SQL_SUCCESS_WITH_INFO，则表示成功地与数据库断开连接，但会出现其它错误信息或特定于实现的信息。例如，在处理断开连接的后续事项时遇到问题，或者因为发生与应用程序无关的事件（如通信故障）而使得当前没有任何连接。

在调用 SQLDisconnect() 成功之后，应用程序可以重新使用 *ConnectionHandle* 来发出另一个 SQLConnect() 或 SQLDriverConnect() 请求。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO

- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 43. SQLDisconnect SQLSTATE

SQLSTATE	描述	解释
01002	断开连接错误。	断开连接时出错。但是，断开连接成功。（函数返回 SQL_SUCCESS_WITH_INFO。）
08003	连接已关闭。	*ConnectionHandle* 自变量中指定的连接未打开。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 179 页的『SQLAllocHandle - 分配句柄』
- 第 189 页的『SQLConnect - 连接至数据源』
- 第 215 页的『SQLFreeHandle - 释放句柄资源』

SQLEndTran - 请求 COMMIT 或 ROLLBACK

用途:

规范:	DB2 CLI	ODBC	ISO CLI

SQLEndTran() 向与连接相关联的所有语句上的所有活动操作请求 COMMIT 或 ROLLBACK 操作。

语法:

```
SQLRETURN   SQLEndTran      (SQLSMALLINT       HandleType,
                             SQLHANDLE         Handle,
                             SQLSMALLINT       Completion Type);
```

函数自变量:

表 44. SQLEndTran 自变量

数据类型	自变量	使用	描述
SQLSMALLINT	*HandleType*	输入	句柄类型。
SQLHANDLE	*Handle*	输入	连接句柄。
SQLSMALLINT	*CompletionType*	输入	如何完成与连接相关联的活动操作。

用法:

HandleType
句柄类型标识。只允许 SQL_HANDLE_DBC（连接句柄）。

Handle 由 *HandleType* 所指示的类型的句柄。

CompletionType

下列两个值之一:

- SQL_COMMIT
- SQL_ROLLBACK

在手工落实方式下，必须在调用 SQLDisconnect() 之前调用 SQLEndTran()。如果不在 SQLDisconnect() 之前调用 SQLEndTran()，则会回滚更新数据库的操作（自从启动上一事务后）。

当执行 ROLLBACK 时，将清除所有语句句柄。

如果应用程序在以手工方式进行时过早地崩溃或终止，则自上一次 COMMIT 以来所作的更新就会丢失。必须在调用断开连接之前调用 SQLEndTran()。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 45. SQLEndTran SQLSTATE

SQLSTATE	描述	解释
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 241 页的『SQLSetConnectAttr - 设置与连接相关的选项』

SQLError - 检索错误信息

在 ODBC 版本 3 中，SQLError() 已被废弃，且替换为 SQLGetDiagRec() 和 SQLGetDiagField()；有关更多信息，请参阅第 226 页的『SQLGetDiagRec - 获取诊断记录的多个字段设置』。

建议: 虽然此版本的 DB2 CLI 继续支持 SQLError()，但请在 DB2 CLI 程序中使用 SQLGetDiagRec()，以便符合最新标准。

迁移至新函数

例如，要获取与特定语句句柄相关联的诊断信息，在使用新函数时，语句:

```
SQLError(henv, hdbc, hstmt, szSqlState, pfNativeError, szErrorMsg,
       cbErrorMsgMax, pcbErrorMsg);
```

将重写为:

```
SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, 1, szSqlState, pfNativeError,
       szErrorMsg, cbErrorMsgMax, pcbErrorMsg);
```

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLExecDirect - 直接执行语句

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLExecDirect() 直接执行指定的 SQL 语句。该语句只能执行一次。

语法:

```
SQLRETURN   SQLExecDirect    (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLCHAR       *FAR StatementText,    /* szSqlStr */
                              SQLINTEGER         TextLength);      /* cbSqlStr */
```

函数自变量:

表 46. SQLExecDirect 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLCHAR *	*StatementText*	输入	SQL 语句字符串。
SQLINTEGER	*TextLength*	输入	*StatementText* 自变量的内容的长度。此长度必须设置为语句的精确长度，或者，如果语句以 null 结束，则设置为 SQL_NTS。

用法:

SQL 语句字符串不能包含参数标记。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果 SQL 语句是“搜索型 UPDATE”或“搜索型 DELETE”，且没有任何行满足搜索条件，则返回 SQL_NO_DATA_FOUND。

诊断:

表 47. SQLExecDirect SQLSTATE

SQLSTATE	描述	解释
22003	数值超出范围。	对数字类型列指定的数值导致在赋值期间或计算中间结果期间截断该数字的整个部分。
42xxx	语法错误或访问规则违例。	**42**xxx SQLSTATE 指示与语句相关的各种语法或访问问题。xxx 指的是任何具有该类代码的 SQLSTATE。示例: **42**xxx 指的是 **42** 类中的任何 SQLSTATE。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY009	自变量值无效。	*StatementText* 是空指针。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效。	*TextLength* 自变量小于 1 但不等于 SQL_NTS。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 182 页的『SQLBindCol - 将列绑定至应用程序变量』

SQLExecute - 执行语句

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

`SQLExecute()` 将使用 `SQLPrepare()` 成功准备的语句执行一次或多次。使用 `SQLBindParameter()` 绑定至参数标记的任何应用程序变量的当前值执行该语句。

语法:

```
SQLRETURN   SQLExecute      (SQLHSTMT         StatementHandle);  /* hstmt */
```

函数自变量:

表 48. SQLExecute 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。

用法:

SQL 语句字符串可以包含参数标记。参数标记由 ? 字符表示，用来指示语句中的一个位置，调用 SQLExecute() 时，将会替换应用程序在该位置中提供的值。此值可以从应用程序变量获取。SQLBindParameter() 用来将应用程序存储区与参数标记绑定。

在调用 SQLExecute() 之前，必须绑定所有参数。

在应用程序处理完 SQLExecute() 调用的结果之后，它可以再次执行该语句，并指定新的（或相同的）参数值。

不能通过调用 SQLExecute() 重新执行由 SQLExecDirect() 执行的语句；必须首先调用 SQLPrepare()。

如果生成了结果集，则 SQLFetch() 把下一行数据检索到绑定的变量中。也可以通过对任何未绑定的列调用 SQLGetData() 来检索数据。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果 SQL 语句是“搜索型 UPDATE”或“搜索型 DELETE”，且没有任何行满足搜索条件，则返回 SQL_NO_DATA_FOUND。

诊断:

SQLExecute() 的 SQLSTATE 包括 SQLExecDirect() 的所有 SQLSTATE（参阅第 202 页的表 47）（**HY**009 和 **HY**090 除外），同时还包括表 49 中的 SQLSTATE。

表 49. SQLExecute SQLSTATE

SQLSTATE	描述	解释
HY010	函数顺序错误。	指定的 *StatementHandle* 未处于已准备状态。调用 SQLExecute() 之前未首先调用 SQLPrepare()。
08004	应用程序服务器拒绝了连接。	用来连接至数据源的用户名或密码不正确。
08S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
39001	用户定义的函数返回了无效的 SQLSTATE。	用户定义的函数返回了无效的 SQLSTATE。
59101	未定义用户。	在“移动式设备管理中心”控制数据库中未定义用户。
59102	密码不正确。	用户密码与“移动式设备管理中心”中定义的密码不匹配。
59103	未定义组。	在“移动式设备管理中心”中未定义组。
59104	未定义应用程序。	在“移动式设备管理中心”中未定义应用程序。
59105	未定义预订。	在“移动式设备管理中心”中未定义具有 ″AgentAdapter″ 的预订。
59106	预订不完整。	预订没有调用远程存储过程所需的所有信息。

表 49. SQLExecute SQLSTATE (续)

SQLSTATE	描述	解释
59120	XML 转换错误。	AgentAdapter 在将用户输入数据转换为 XML 文档时失败。
59121	一般 AgentAdapter 错误。	一般 AgentAdapter 错误。
59122	装入库失败。	系统上找不到某些必需的库。
HY501	数据源名称无效。	指定的数据源名称无效。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 185 页的『SQLBindParameter - 将参数标记绑定至缓冲区』
- 第 182 页的『SQLBindCol - 将列绑定至应用程序变量』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 236 页的『SQLPrepare - 准备语句』
- 『SQLFetch - 取装下一行』

SQLFetch - 取装下一行

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	

SQLFetch() 使游标进至结果集的下一行，并检索任何绑定的列。

列可以与应用程序存储器绑定。

当调用 SQLFetch() 时，会执行适当的数据传输和任何数据转换（如果绑定列时指示了转换的话）。在取装之后，还可以通过调用 SQLGetData() 分别地接收到列。

仅当通过执行查询生成了结果集之后（使用同一语句句柄），才能调用 SQLFetch()。

语法:

```
SQLRETURN   SQLFetch         (SQLHSTMT          StatementHandle);  /* hstmt */
```

函数自变量:

表 50. SQLFetch 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。

用法:

仅当对同一语句句柄生成了结果集之后，才能调用 SQLFetch()。在首次调用 SQLFetch() 之前，游标定位在结果集开头的前面。

与 SQLBindCol() 绑定的应用程序变量的数目不得超过结果集中的列数，否则 SQLFetch() 会失败。

如果未调用 SQLBindCol() 来绑定任何列，则 SQLFetch() 不将任何数据返回至应用程序，而是仅仅使游标前进。在这种情况下，可以调用 SQLGetData() 来分别获取所有的列。当 SQLFetch() 时游标进至下一行时，未绑定列中的数据被废弃。

列可以与应用程序存储器绑定。SQLBindCol() 用来将应用程序存储器绑定至列。在取装时，会将数据从数据库传送至应用程序。还会设置要返回的可用数据的长度。

如果任何绑定的存储器缓冲区不够大，无法存放 SQLFetch() 返回的数据，则会截断该数据将。如果字符数据被截断，则返回 SQL_SUCCESS_WITH_INFO，且生成指示此截断的 SQLSTATE。SQLBindCol() 延迟输出自变量 *pcbValue* 包含从服务器检索的列数据的实际长度。应用程序应将实际输出长度与输入缓冲区长度（来自 SQLBindCol() 的 *pcbValue* 和 *cbValueMax* 自变量）作比较，以确定截断了哪些字符列。

截断数字数据类型时，如果截断小数点右面的位，则会将此截断报告为警告。如果截断在小数点左边发生，则会返回错误（参阅『诊断』一节）。

在从结果集中检索了所有行，或者不需要剩余的行时，调用 SQLFreeStmt() 来关闭游标并废弃剩余的数据和相关联的资源。

DB2 Everyplace 每次至多取装一行，而不是使用行集。DB2 Everyplace 不支持语句描述符。

SQLFetch() 确定应用程序是否指定了单独的长度和指示符缓冲区。在这种情况下，当数据不是 NULL 时，SQLFetch() 将指示符缓冲区设置为 0，并在长度缓冲区中返回长度。当数据为 NULL 时，SQLFetch() 将指示符缓冲区设置为 SQL_NULL_DATA，而不修改长度缓冲区。

定位游标:

当创建结果集时，游标定位在结果集开头的前面。SQLFetch() 取装下一行。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果结果集中没有任何行，或先前的 SQLFetch() 调用已从结果集中取装了所有的行，则返回 SQL_NO_DATA_FOUND。

如果已取装所有的行，则游标定位在结果集末尾之后。

诊断:

表 51. SQLFetch SQLSTATE

SQLSTATE	描述	解释
01004	数据被截断。	对一个或多个列返回的数据被截断。字符串值或数值在右端被截断。（如果不出错，则返回 SQL_SUCCESS_WITH_INFO。）
07006	转换无效。	未能将数据值以有意义的方式转换为 SQLBindCol() 中的 *fCType* 所指定的数据类型。
22002	指定的输出或指示符缓冲区无效。	对 SQLBindCol() 中 *pcbValue* 自变量指定的指针值是空指针，且相应列的值为空。报告 SQL_NULL_DATA 是没有意义的。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	在对 *StatementHandle* 调用 SQLPrepare() 或 SQLExecDirect() 之前调用了该函数。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 182 页的『SQLBindCol - 将列绑定至应用程序变量』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 223 页的『SQLGetData - 从列中获取数据』

SQLFetchScroll - 取装行集并返回所有已绑定列的数据

用途:

规范:	**DB2 CLI** 5.0	**ODBC** 3.0	

SQLFetchScroll() 从结果集取装指定的数据行集，并返回所有已绑定列的数据。可以使用绝对位置或相对位置指定行集。

语法:

```
SQLRETURN   SQLFetchScroll  (
                SQLHSTMT            StatementHandle,
                  SQLSMALLINT       FetchOrientation,
                  SQLINTEGER        FetchOffset);
```

函数自变量:

表 52. SQLFetchScroll 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄
SQLSMALLINT	*FetchOrientation*	输入	取装类型: • SQL_FETCH_NEXT • SQL_FETCH_PRIOR • SQL_FETCH_FIRST • SQL_FETCH_LAST • SQL_FETCH_ABSOLUTE • SQL_FETCH_RELATIVE
SQLINTEGER	*FetchOffset*	输入	要取装的行数。此自变量的解释取决于 *FetchOrientation* 自变量的值。

用法:

SQLFetchScroll() 从结果集返回指定的行集。可用绝对位置或相对位置指定行集。仅当结果集存在时，才能调用 SQLFetchScroll() – 即，在创建结果集的调用之后，而在该结果集上的游标关闭之前，才能调用。如果绑定了任何列，则返回这些列中的数据。如果应用程序指定了指向行状态数组或要在其中返回所取装行数的缓冲区的指针，则 SQLFetchScroll() 还返回此信息。对 SQLFetchScroll() 调用可以与对 SQLFetch() 调用混合在一起。

定位游标:

当创建结果集时，游标定位在结果集开头的前面。SQLFetchScroll() 根据 *FetchOrientation* 和 *FetchOffset* 自变量的值定位块游标，如以下列表所示。确定新行集开头的精确规则在下一节中说明。

FetchOrientation	含义
SQL_FETCH_NEXT	返回下一行集。这与调用 SQLFetch() 等效。SQLFetchScroll() 忽略 *FetchOffset* 的值。
SQL_FETCH_PRIOR	返回前一行集。SQLFetchScroll() 忽略 *FetchOffset* 的值。
SQL_FETCH_RELATIVE	返回自当前行集起始位置开始的行集 *FetchOffset*。
SQL_FETCH_ABSOLUTE	返回从行 *FetchOffset* 开始的行集。
SQL_FETCH_FIRST	返回结果集中的第一个行集。SQLFetchScroll() 忽略 *FetchOffset* 的值。
SQL_FETCH_LAST	返回结果集中的最后一个完整行集。SQLFetchScroll() 忽略 *FetchOffset* 的值。

SQL_ATTR_ROW_ARRAY_SIZE 语句属性指定行集中的行数。如果 SQLFetchScroll() 取装的行集与结果集的末尾重叠，则 SQLFetchScroll() 返回部分行集。也就是说，如果 S + R - 1 大于 L，其中 S 是正在取装的行集的起始行，R 是行集大小，而 L 是结果集中的最后一行，则只有行集的前 L - S + 1 行有效。其余各行是空的，状态为 SQL_ROW_NOROW。

SQLFetchScroll() 返回后，行集游标定位在结果集的第一行。

游标定位规则:

下列各节描述 *FetchOrientation* 的每个值的精确规则。这些规则使用下列表示法:

在起始点之前

块游标定位在结果集起始点前面。如果新行集的第一行在结果集的起始点之前，则 SQLFetchScroll() 返回 SQL_NO_DATA。

在末尾之后

块游标定位在结果集末尾之后。如果新行集的第一行在结果集的末尾之后，则 SQLFetchScroll() 返回 SQL_NO_DATA。

CurrRowsetStart

这是当前行集中第一行的编号。

LastResultRow

这是结果集中最后一行的编号。

RowsetSize

这是行集大小。

FetchOffset

这是 *FetchOffset* 自变量的值。

SQL_FETCH_NEXT 规则:

表 53. SQL_FETCH_NEXT 规则:

条件	新行集的第一行
Before start	1
CurrRowsetStart + RowsetSize <= LastResultRow	CurrRowsetStart + RowsetSize
CurrRowsetStart + RowsetSize > LastResultRow	在末尾之后
After end	在末尾之后

SQL_FETCH_PRIOR 规则:

表 54. SQL_FETCH_PRIOR 规则:

条件	新行集的第一行
Before start	在起始点之前
CurrRowsetStart = 1	在起始点之前
1 < CurrRowsetStart <= RowsetSize	1[a]
CurrRowsetStart > RowsetSize	CurrRowsetStart - RowsetSize
After end AND LastResultRow < RowsetSize	1[a]
After end AND LastResultRow >= RowsetSize	LastResult - RowRowsetSize + 1

a SQLFetchScroll() 返回 SQLSTATE 01S06 （试图在结果集返回第一个行集之前进行取装）和 SQL_SUCCESS_WITH_INFO。

SQL_FETCH_RELATIVE 规则:

表 55. SQL_FETCH_RELATIVE 规则:

条件	新行集的第一行
(Before start AND *FetchOffset* > 0) OR (After end AND *FetchOffset* < 0)	--[a]
Before start AND *FetchOffset* <= 0	在起始点之前

表 55. SQL_FETCH_RELATIVE 规则: (续)

条件	新行集的第一行
CurrRowsetStart = 1 AND *FetchOffset* < 0	在起始点之前
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| > RowsetSize	在起始点之前
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| <= RowsetSize	1[b]
1 <= CurrRowsetStart + *FetchOffset* <= LastResultRow	CurrRowsetStart + *FetchOffset*
CurrRowsetStart + *FetchOffset* > LastResultRow	在末尾之后
After end AND *FetchOffset* >= 0	在末尾之后

a SQLFetchScroll() 返回的结果集与调用它时将 *FetchOrientation* 设置为 SQL_FETCH_ABSOLUTE 时返回的行集相同。有关更多信息，请参阅『SQL_FETCH_ABSOLUTE』一节。

b SQLFetchScroll() 返回 SQLSTATE 01S06 （试图在结果集返回第一个行集之前进行取装。）和 SQL_SUCCESS_WITH_INFO。

SQL_FETCH_ABSOLUTE 规则:

表 56. SQL_FETCH_ABSOLUTE 规则:

条件	新行集的第一行
FetchOffset < 0 AND |*FetchOffset*| <= LastResultRow	LastResultRow + *FetchOffset* + 1
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |*FetchOffset*| > RowsetSize	在起始点之前
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |*FetchOffset*| <= RowsetSize	1[a]
FetchOffset = 0	在起始点之前
1 <= *FetchOffset* <= LastResultRow	*FetchOffset*
FetchOffset > LastResultRow	在末尾之后

a SQLFetchScroll() 返回 SQLSTATE 01S06 （试图在结果集返回第一个行集之前进行取装。）和 SQL_SUCCESS_WITH_INFO。

SQL_FETCH_FIRST 规则:

表 57. SQL_FETCH_FIRST 规则:

条件	新行集的第一行
任何	1

SQL_FETCH_LAST 规则:

表 58. SQL_FETCH_LAST 规则:

条件	新行集的第一行
RowsetSize <= LastResultRow	LastResultRow - RowsetSize + 1
RowsetSize > LastResultRow	1

返回绑定列中的数据:

SQLFetchScroll() 返回绑定列中数据的方式与 SQLFetch() 相同。有关更多信息，请参阅第 204 页的『SQLFetch - 取装下一行』。

如果未绑定任何列，则 SQLFetchScroll() 不返回数据，但将块游标移至指定的位置。对于 SQLFetch()，在这种情况下可以使用 SQLGetData() 来检索信息。

缓冲区地址:

SQLFetchScroll() 使用 SQLFetch() 所使用的公式来确定数据的地址和长度/指示符缓冲区。有关更多信息，请参阅第 182 页的『SQLBindCol - 将列绑定至应用程序变量』。

行状态数组:

行状态数组用来返回行集中每一行的状态。此数组的地址使用 SQL_ATTR_ROW_STATUS_PTR 语句属性指定。此数组由应用程序分配，且必须具有 SQL_ATTR_ROW_ARRAY_SIZE 语句属性所指定的元素数目。它的值由 SQLFetch() 和 SQLFetchScroll() 设置。如果 SQL_ATTR_ROW_STATUS_PTR 语句属性的值是空指针，则这些函数不返回行状态。

如果 SQLFetch() 或 SQLFetchScroll() 未返回 SQL_SUCCESS 或 SQL_SUCCESS_WITH_INFO，则行状态数组缓冲区的内容未定义。

下列值在行状态数组中返回。

行状态数组值	描述
SQL_ROW_SUCCESS	成功取装了行。
SQL_ROW_SUCCESS_WITH_INFO	成功取装了行。但是返回了关于该行的警告。
SQL_ROW_ERROR	取装行时出错。
SQL_ROW_NOROW	行集与结果集的末尾重叠，未返回与行状态数组的此元素对应的行。

“取装的行”缓冲区:

“取装的行”缓冲区用来返回取装的行数，包括那些因为在取装行时出错而未返回数据的行。这是在行状态数组中其值不是 SQL_ROW_NOROW 的那些行的数目。此缓冲区的地址使用 SQL_ATTR_ROWS_FETCHED_PTR 语句属性指定。此缓冲区由应用程序分配。它由 SQLFetch() 和 SQLFetchScroll() 设置。如果 SQL_ATTR_ROWS_FETCHED_PTR 语句属性的值是空指针，则这些函数不返回取装的行数。要确定结果集中的当前行数，应用程序可以调用 SQLGetStmtAttr()，并指定 SQL_ATTR_ROW_NUMBER 属性。

如果 SQLFetch() 或 SQLFetchScroll() 未返回 SQL_SUCCESS 或 SQL_SUCCESS_WITH_INFO，则“取装的行”缓冲区的内容未定义，返回 SQL_NO_DATA 时是一个例外，在这种情况下，“取装的行”缓冲区中的值设置为 0。

返回码:

- SQL_SUCCESS

- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 59. SQLFetchScroll SQLSTATE

SQLSTATE	描述	解释
01000	警告	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
01004	数据被截断。	对一个或多个列返回的数据被截断。字符串值或数值在右端被截断。（如果不出错，则返回 SQL_SUCCESS_WITH_INFO。）
01S06	试图在结果集返回第一个行集之前进行取装。	在当前位置在第一行之外，且 FetchOrientation 是 SQL_PRIOR，或 FetchOrientation 是 SQL_RELATIVE，而 FetchOffset 是负数时（其绝对值小于或等于当前 SQL_ATTR_ROW_ARRAY_SIZE），则请求的行集与结果集的起始点重叠。（函数返回 SQL_SUCCESS_WITH_INFO。）
07006	转换无效。	未能将数据值以有意义的方式转换为 SQLBindCol() 中的 *fCType* 所指定的数据类型。
22002	指定的输出或指示符缓冲区无效。	对 SQLBindCol() 中 *pcbValue* 自变量指定的指针值是空指针，且相应列的值为空。报告 SQL_NULL_DATA 是没有意义的。
22003	数值超出范围。	对一个或多个绑定列返回数字值（作为数字或字符串）会导致数字的整个部分（并非分数部分）被截断。
24000	游标状态无效。	*StatementHandle* 处于已执行状态，但没有任何结果集与 *StatementHandle* 相关联。
HY000	一般错误。	出错，该错误没有特定的 SQLSTATE。**MessageText* 缓冲区中由 SQLGetDiagRec() 返回的错误消息描述该错误及其原因。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	在对 *StatementHandle* 调用 SQLPrepare() 或 SQLExecDirect() 之前调用了该函数。
HY106	取装类型超出范围。	对自变量 *FetchOrientation* 指定的值无效。 SQL_CURSOR_TYPE 语句属性的值是 SQL_CURSOR_FORWARD_ONLY，而 *FetchOrientation* 自变量的值不是 SQL_FETCH_NEXT。
HY107	行值超出范围。	用 SQL_ATTR_CURSOR_TYPE 语句属性指定的值是 SQL_CURSOR_KEYSET_DRIVEN，但用 SQL_ATTR_KEYSET_SIZE 语句属性指定的值大于 0 且小于用 SQL_ATTR_ROW_ARRAY_SIZE 语句属性指定的值。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 182 页的『SQLBindCol - 将列绑定至应用程序变量』
- 第 196 页的『SQLDescribeCol - 返回列的一组属性』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 204 页的『SQLFetch - 取装下一行』
- 第 202 页的『SQLExecute - 执行语句』
- 第 235 页的『SQLNumResultCols - 获取结果列数』
- 第 244 页的『SQLSetStmtAttr - 设置与语句相关的选项』

SQLForeignKeys - 获取外键列的列表

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLForeignKeys() 返回关于指定表的外键的信息。此信息在 SQL 结果集中返回，可以使用用来检索查询生成的结果集的函数来处理此结果集。忽略 *PKCatalogName*、*NameLength1*、*PKSchemaName*、*NameLength2*、*FKCatalogName*、*NameLength4*、*FKSchemaName* 和 *NameLength5*。返回的结果集的第 1、2、5、6、12 和 13 列总是长度为零的字符串。返回的结果集的第 10、11和 14 列总是零。

语法:

```
SQLRETURN   SQLForeignKeys  (
                  SQLHSTMT         StatementHandle,  /* hstmt */
                  SQLCHAR      *FAR PKCatalogName,   /* szPkCatalogName */
                  SQLSMALLINT      NameLength1,      /* cbPkCatalogName */
                  SQLCHAR      *FAR PKSchemaName,    /* szPkSchemaName */
                  SQLSMALLINT      NameLength2,      /* cbPkSchemaName */
                  SQLCHAR      *FAR PKTableName,     /* szPkTableName */
                  SQLSMALLINT      NameLength3,      /* cbPkTableName */
                  SQLCHAR      *FAR FKCatalogName,   /* szFkCatalogName */
                  SQLSMALLINT      NameLength4       /* cbFkCatalogName */
                  SQLCHAR      *FAR FKSchemaName,    /* szFkSchemaName */
                  SQLSMALLINT      NameLength5,      /* cbFkSchemaName */
                  SQLCHAR      *FAR FKTableName,     /* szFkTableName */
                  SQLSMALLINT      NameLength6);     /* cbFkTableName */
```

函数自变量:

表 60. SQLForeignKeys 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLCHAR*	*PKCatalogName*	输入	主键表的目录限定符。DB2 Everyplace 忽略此字段。
SQLSMALLINT	*NameLength1*	输入	*PKCatalogName* 的长度。DB2 Everyplace 忽略此字段。
SQLCHAR*	*PKSchemaName*	输入	主键表的模式限定符。DB2 Everyplace 忽略此字段。
SQLSMALLINT	*NameLength2*	输入	*PKSchemaName* 的长度。DB2 Everyplace 忽略此字段。
SQLCHAR*	*PKTableName*	输入	包含主键的表的名称。
SQLSMALLINT	*NameLength3*	输入	*PKTableName* 的长度。

表 60. SQLForeignKeys 自变量 (续)

数据类型	自变量	使用	描述
SQLCHAR*	*FKCatalogName*	输入	包含外键的表的目录限定符。DB2 Everyplace 忽略此字段。
SQLSMALLINT	*NameLength4*	输入	*FKCatalogName* 的长度。DB2 Everyplace 忽略此字段。
SQLCHAR*	*FKSchemaName*	输入	包含外键的表的模式限定符。DB2 Everyplace 忽略此字段。
SQLSMALLINT	*NameLength5*	输入	*FKSchemaName* 的长度。DB2 Everyplace 忽略此字段。
SQLCHAR*	*FKTableName*	输入	包含外键的表的名称。
SQLSMALLINT	*NameLength6*	输入	*FKTableName* 的长度。

用法:

如果 *PKTableName* 包含表名，且 *FKTableName* 是空字符串，则 SQLForeignKeys() 返回一个结果集，此结果集包含指定表的主键以及所有引用该表的外键（在其它表中）。

如果 *FKTableName* 包含一个表名，而 *PKTableName* 是空字符串，则 SQLForeignKeys() 返回一个结果集，此结果集包含指定表中的所有外键以及它们所引用的主键（在其它表中）。

如果 *PKTableName* 和 *FKTableName* 都包含表名，则 SQLForeignKeys() 返回在 *FKTableName* 中指定的表中的外键，这些外键引用在 *PKTableName* 中指定的表的主键。最多只应有一个键。

如果请求与主键相关联的外键，则结果集按 FKTABLE_NAME 和 ORDINAL_POSITION 排序。如果请求与外键相关联的主键，则结果集按 PKTABLE_NAME 和 ORDINAL_POSITION 排序。

用最大长度属性 128 声明目录函数结果集的 VARCHAR 列，以便与 SQL92 限制一致。

虽然可以添加新列，将来的发行版中也可以更改现有列的名称，但当前列的位置不会改变。

结果集包含以下列: :

第 1 列 PKTABLE_CAT（VARCHAR(128)）
这始终是长度为零的字符串。

第 2 列 PKTABLE_SCHEM（VARCHAR(128)）
这始终是长度为零的字符串。

第 3 列 PKTABLE_NAME（VARCHAR(128) 且不为 NULL）
包含主键的表的名称。

第 4 列 PKCOLUMN_NAME（VARCHAR(128) 且不为 NULL）
主键列名。

第 5 列 FKTABLE_CAT（VARCHAR(128)）
这始终是长度为零的字符串。

第 6 列 FKTABLE_SCHEM（VARCHAR(128)）
这始终是长度为零的字符串。

第 7 列 FKTABLE_NAME（VARCHAR(128) 且不为 NULL）
包含外键的表的名称。

第 8 列 FKCOLUMN_NAME（VARCHAR(128) 且不为 NULL）
外键列名。

第 9 列 ORDINAL_POSITION（SMALLINT 且不为 NULL）
键中列的原始位置，从 1 开始。

第 10 列 UPDATE_RULE（SMALLINT）
这始终是零。

第 11 列 DELETE_RULE（SMALLINT）
这始终是零。

第 12 列 FK_NAME（VARCHAR(128)）
这始终是长度为零的字符串。

第 13 列 PK_NAME（VARCHAR(128)）
这始终是长度为零的字符串。

第 14 列 DEFERRABILITY（SMALLINT）
这始终是零。

DB2 CLI 使用的列名遵循 X/Open CLI CAE 规范样式。列类型、内容和顺序与对 ODBC 中的 `SQLForeignKeys()` 结果集定义的列类型、内容和顺序完全相同。

返回码:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 61. SQLForeign SQLSTATE

SQLSTATE	描述	解释
24000	游标状态无效。	已在该语句句柄上打开了游标。
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY009	自变量值无效。	*PKTableName* 和 *FKTableName* 自变量都是 NULL 指针。
HY010	函数顺序错误。	执行“处理数据”的操作（`SQLPrepare()` 或 `SQLExecDirect()`）时调用了该函数。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效。	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS

限制:

无。

相关参考:
- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

- 第 238 页的『SQLPrimaryKeys - 获取表的主键列』

SQLFreeConnect - 释放连接句柄

在 ODBC 版本 3 中，SQLFreeConnect() 已被废弃，且替换为 SQLFreeHandle()；有关更多信息，请参阅『SQLFreeHandle - 释放句柄资源』。

建议：虽然此版本的 DB2 CLI 继续支持 SQLFreeConnect，但请在 DB2 CLI 程序中使用 SQLFreeHandle，以便符合最新标准。

迁移至新函数

例如，使用新的函数时，语句:

```
SQLFreeConnect(hdbc);
```

将重写为:

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLFreeEnv - 释放环境句柄

在 ODBC 版本 3 中，SQLFreettEnv 已被废弃，且替换为 SQLFreeHandle()；有关更多信息，请参阅『SQLFreeHandle - 释放句柄资源』。

建议：虽然此版本的 DB2 CLI 继续支持 SQLFreeEnv()，但请在 DB2 CLI 程序中使用 SQLFreeHandle，以便符合最新标准。

迁移至新函数

例如，使用新的函数时，语句:

```
SQLFreeEnv(henv);
```

将重写为:

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLFreeHandle - 释放句柄资源

用途:

规范:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFreeHandle() 释放与特定环境、连接或语句句柄相关联的资源。

此函数是用于释放资源的类属函数。它替换 SQLFreeConnect（用于释放连接句柄）和 SQLFreeEnv()（用于释放环境句柄）。 SQLFreeHandle() 还替换用于释放语句句柄的 SQLFreeStmt()（指定 SQL_DROP 选项）。

语法:

```
SQLRETURN   SQLFreeHandle   (SQLSMALLINT      HandleType,
                             SQLHANDLE       Handle);
```

函数自变量:

表 62. SQLFreeHandle 自变量

数据类型	自变量	使用	描述
SQLSMALLINT	*HandleType*	输入	SQLFreeHandle() 要释放的句柄的类型。必须是下列其中一值: SQL_HANDLE_ENV SQL_HANDLE_DBC SQL_HANDLE_STMT 如果 *HandleType* 不是上述值之一，则 SQLFreeHandle() 返回 SQL_INVALID_HANDLE。
SQLHANDLE	*Handle*	输入	要释放的句柄的名称。

用法:

SQLFreeHandle() 用来释放环境、连接和语句的句柄。

在释放句柄之后，应用程序不应再使用该句柄；DB2 CLI 不检查函数调用中句柄的有效性。

释放环境句柄:

在调用 SQLFreeHandle()（*HandleType* 为 SQL_HANDLE_ENV）之前，应用程序必须对环境中分配的所有连接调用 SQLFreeHandle()（*HandleType* 为 SQL_HANDLE_DBC）。否则，对 SQLFreeHandle() 的调用返回 SQL_ERROR，且环境和任何活动连接仍有效。

释放连接句柄:

在调用 SQLFreeHandle()（*HandleType* 为 SQL_HANDLE_DBC）之前，应用程序必须对连接调用 SQLDisconnect()。否则，对 SQLFreeHandle() 的调用返回 SQL_ERROR，且连接仍有效。

释放语句句柄:

对 SQLFreeHandle() 的调用（*HandleType* 为 SQL_HANDLE_STMT）释放 SQLAllocHandle() 调用（*HandleType* 为 SQL_HANDLE_STMT）分配的所有资源。当应用程序调用 SQLFreeHandle() 来释放带有暂挂结果的语句时，暂挂的结果被删除。调用 SQLFreeHandle() 时，如果有暂挂的结果，则结果集被废弃。

SQLDisconnect() 自动删除在该连接上打开的所有语句。

返回码:

- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

如果 SQLFreeHandle() 返回 SQL_ERROR，则句柄仍有效。

诊断:

表 63. SQLFreeHandle SQLSTATE

SQLSTATE	描述	解释
01000	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
08S01	通信链路故障。	*HandleType* 自变量是 SQL_HANDLE_DBC，在函数完成处理之前，DB2 CLI 与它尝试连接的数据源之间的通信链路失效。
HY000	一般错误。	出错，该错误没有特定的 SQLSTATE。**MessageText* 缓冲区中由 SQLGetDiagRec() 返回的错误消息描述该错误及其原因。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	*HandleType* 自变量为 SQL_HANDLE_ENV 且至少有一个连接处于已分配或已连接状态。在调用 *HandleType* 为 SQL_HANDLE_ENV 的 SQLFreeHandle() 之前，必须对每个连接调用 *HandleType* 为 SQL_HANDLE_DBC 的 SQLDisconnect() 和 SQLFreeHandle()。*HandleType* 自变量是 SQL_HANDLE_DBC，而在对连接调用 SQLDisconnect() 之前调用了该函数。 *HandleType* 自变量是 SQL_HANDLE_STMT；使用语句句柄调用了 SQLExecute() 或 SQLExecDirect()，且返回了 SQL_NEED_DATA。（DM）在调用 SQLFreeHandle() 之前，未释放任何附属句柄和其它资源。
HY013	发生意外的内存处理错误。	*HandleType* 自变量是 SQL_HANDLE_STMT；因为未能访问底层内存对象（可能是因为内存不足），所以未能处理函数调用。
HY017	自动分配的描述符句柄的使用无效。	*Handle* 自变量设置为自动分配的描述符或实现描述符的句柄。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 179 页的『SQLAllocHandle - 分配句柄』

SQLFreeStmt - 释放（或复位）语句句柄

用途:

规范:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLFreeStmt() 结束对语句句柄引用的语句的处理。使用此函数来:

- 取消参数与应用程序变量的关联（复位）。
- 删除语句句柄并释放与该句柄相关联的 DB2 CLI 资源。

在执行 SQL 语句和处理结果之后，调用 SQLFreeStmt()。

语法:

```
SQLRETURN   SQLFreeStmt     (SQLHSTMT          StatementHandle,   /* hstmt */
                             SQLUSMALLINT      Option);           /* fOption */
```

函数自变量:

表 64. SQLFreeStmt 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLUSMALLINT	*Option*	输入	指定释放语句句柄的方式的选项。此选项可以具有下列其中一个值: SQL_DROP 或 SQL_RESET_PARAMS。

用法:

调用 SQLFreeStmt() 时，可以指定下列选项:

SQL_DROP

释放与输入语句句柄相关联的 DB2 CLI 资源，使该句柄无效。废弃所有暂挂结果。

此选项已被替换为 SQLFreeHandle() 调用（*HandleType* 设置为 SQL_HANDLE_STMT）。

建议: 虽然此版本的 DB2 CLI 继续支持此选项，但请在 DB2 CLI 程序中使用 SQLFreeHandle，以便符合最新标准。

SQL_RESET_PARAMS

释放 SQLBindParameter() 对 *StatementHandle* 设置的所有参数缓冲区。

另外，可以删除语句句柄并分配新句柄。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

如果 *Option* 设置为 SQL_DROP，则不返回 SQL_SUCCESS_WITH_INFO，这是因为调用 SQLError() 时，将没有任何语句句柄可用。

诊断:

表 65. SQLFreeStmt SQLSTATE

SQLSTATE	描述	解释
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY092	选项类型超出范围。	对 *Option* 自变量指定的值不是 SQL_DROP 或 SQL_RESET_PARAMS。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 179 页的『SQLAllocHandle - 分配句柄』
- 第 182 页的『SQLBindCol - 将列绑定至应用程序变量』

SQLGetConnectAttr - 获取连接属性的当前设置

用途:

规范:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

SQLGetConnectAttr() 返回连接属性的当前设置。

语法:

```
SQLRETURN   SQLGetConnectAttr(SQLHDBC           ConnectionHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER         BufferLength,
                              SQLINTEGER        *StringLengthPtr);
```

函数自变量:

表 66. SQLGetConnectAttr 自变量

数据类型	自变量	使用	描述
SQLHDBC	*ConnectionHandle*	输入	连接句柄。
SQLINTEGER	*Attribute*	输入	要检索的属性。
SQLPOINTER	*ValuePtr*	输出	指向内存的指针，将在该内存中返回 *Attribute* 指定的属性的当前值。

表 66. SQLGetConnectAttr 自变量 (续)

数据类型	自变量	使用	描述
SQLINTEGER	*BufferLength*	输入	• 如果 *ValuePtr* 指向字符串，则此自变量的长度应为 **ValuePtr*。 • 如果 *ValuePtr* 是一个指针但并未指向字符串，则 *BufferLength* 应具有值 *SQL_IS_POINTER*。 • 如果 **ValuePtr* 中的值为 Unicode 字符串，则 *BufferLength* 自变量必须为偶数。
SQLINTEGER *	*StringLengthPtr*	输出	指向缓冲区的指针，将在该缓冲区中返回可在 **ValuePtr* 中返回的总字节数（排除 null 终止符）。 • 如果 *ValuePtr* 是空指针，则不返回任何长度。 • 如果属性值是字符串且可返回的字节数大于 *BufferLength* 减去 null 终止字符的长度，**ValuePtr* 中的数据将截断为 *BufferLength* 减去 null 终止字符的长度且由 DB2 CLI 以 null 结束。

用法:

调用 SQLGetConnectAttr() 将在 **ValuePtr* 中返回在 *Attribute* 中指定的连接属性的值。在 DB2 Everyplace 中，该值是 32 位值，未使用 *BufferLength* 和 *StringLengthPtr* 自变量。

下列连接属性可通过 SQLGetConnectAttr() 检索。有关属性的描述，请参阅 SQLSetConnectAttr – 设置与连接有关的选项。

- SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)
- SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)
- SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)
- SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)

根据属性，应用程序不必在调用 SQLGetConnectAttr() 之前建立连接。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 67. SQLGetConnectAttr SQLSTATE

SQLSTATE	描述	解释
01000	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）

表 67. SQLGetConnectAttr SQLSTATE (续)

SQLSTATE	描述	解释
01004	数据被截断。	**ValuePtr* 中返回的数据将截断为 *BufferLength* 减去 null 终止字符的长度。将在 **StringLengthPtr* 中返回未截断字符串值的长度。（函数返回 SQL_SUCCESS_WITH_INFO。）
08003	连接已关闭。	*Attribute* 值被指定为需要打开连接。
HY000	一般错误。	发生了错误，该错误没有特定 SQLSTATE。**MessageText* 缓冲区中的 SQLGetDiagRec() 返回的错误消息描述该错误及其原因。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。可能进程级别内存已被应用程序进程耗尽。查阅操作系统配置以获取有关进程级别内存限制的信息。
HY090	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 0。
HY092	选项类型超出范围。	对自变量 *Attribute* 指定的值无效。
HYC00	驱动程序不起作用。	对自变量 Attribute 指定的值是对 DB2 CLI 驱动程序的版本有效的连接或语句属性，但数据源不支持它。

限制:

无。

相关参考:

• 第 241 页的『SQLSetConnectAttr - 设置与连接相关的选项』

SQLGetCursorName - 获取游标名

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLGetCursorName() 返回与输入语句句柄相关联的游标名。如果先前已通过调用 SQLSetCursorName() 显式设置了游标名，则会返回此名称；否则，将返回隐式生成的名称。

语法:

```
SQLRETURN   SQLGetCursorName (
                SQLHSTMT          StatementHandle,  /* hstmt */
            SQLCHAR      *FAR CursorName,       /* szCursor */
            SQLSMALLINT       BufferLength,     /* cbCursorMax */
            SQLSMALLINT  *FAR NameLengthPtr);    /* pcbCursor */
```

函数自变量:

表 68. SQLGetCursorName 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄

表 68. SQLGetCursorName 自变量 (续)

数据类型	自变量	使用	描述
SQLCHAR *	*CursorName*	输出	游标名
SQLSMALLINT	*BufferLength*	输入	*CursorName* 缓冲区的长度
SQLSMALLINT *	*NameLengthPtr*	输出	可为 *CursorName* 返回的字节数

用法:

SQLGetCursorName() 返回用 SQLSetCursorName() 显式设置的游标名，或者，如果未设置任何名称，则它返回 DB2 CLI 内部生成的游标名。

如果使用 SQLSetCursorName() 显式设置了一个名称，则在删除语句之前，或在设置另一显式名称之前，返回此名称。

内部生成的游标名总是以 SQLCUR 或 SQL_CUR 开始。游标名总是不超过 18 个字符，且在连接中总是唯一的。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 69. SQLGetCursorName SQLSTATE

SQLSTATE	描述	解释
01004	数据被截断。	*CursorName* 中返回的游标名比 *BufferLength* 中的值要长，被截断为 *BufferLength* - 1 个字节。*NameLengthPtr* 自变量包含可返回的完整游标名的长度。此函数返回 SQL_SUCCESS_WITH_INFO。
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	执行“处理数据”的操作（SQLParamData() 或 SQLPutData()）时调用了该函数。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY090	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 0。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』

- 第 175 页的『DB2 CLI 函数摘要』
- 第 201 页的『SQLExecDirect - 直接执行语句』

SQLGetData - 从列中获取数据

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLGetData() 在结果集的当前行中检索单一列的数据。这是 SQLBindCol() 的替代函数，用来在进行每个 SQLFetch() 调用时将数据直接传送到应用程序变量中。

必须在调用 SQLGetData() 之前调用 SQLFetch()。

在对每个需要的列调用 SQLGetData() 之后，调用 SQLFetch() 来检索下一行。

语法:

```
SQLRETURN   SQLGetData       (
                SQLHSTMT          StatementHandle,  /* hstmt */
            SQLUSMALLINT      ColumnNumber,     /* icol */
            SQLSMALLINT       TargetType,       /* fCType */
            SQLPOINTER        TargetValuePtr,   /* rgbValue */
            SQLINTEGER        BufferLength,     /* cbValueMax */
            SQLINTEGER   *FAR StrLen_or_IndPtr); /* pcbValue */
```

函数自变量:

表 70. SQLGetData 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLUSMALLINT	*ColumnNumber*	输入	请求检索其数据的列号。结果集列按顺序编号。列号从 1 开始。
SQLSMALLINT	*TargetType*	输入	*ColumnNumber* 标识的列的 C 数据类型。下列类型受支持: SQL_C_BINARY SQL_C_BIT SQL_C_CHAR SQL_C_DOUBLE SQL_C_FLOAT SQL_C_LONG SQL_C_SHORT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_TYPE_TIMESTAMP SQL_C_TINYINT 指定 SQL_C_DEFAULT 导致数据被转换为其缺省 C 数据类型。

表 70. SQLGetData 自变量 (续)

数据类型	自变量	使用	描述
SQLPOINTER	*TargetValuePtr*	输出	指向存储所检索到列数据的缓冲区的指针。 输出缓冲区需要字对齐（平行）。许多处理器（如 Motorola 68000）有字对齐的规则，对于非字符数据类型，应用程序应正确地将缓冲区对齐。
SQLINTEGER	*BufferLength*	输入	*TargetValuePtr* 指向的缓冲区的最大大小。 如果 *TargetType* 指示二进制或字符串，则 BufferLength 必须大于 0，不然会返回错误。否则，此自变量被忽略。
SQLINTEGER *	*StrLen_or_IndPtr*	输出	指向一个值的指针，该值指示 DB2 CLI 可以在 *TargetValuePtr* 缓冲区中返回的字节数。如果发生数据截断的情况，它会包含检索整列所需的总字节数。 对于二进制和字符数据类型，应用程序可选择逐块检索方式来一段一段地检索庞大数据。在此方式下，*StrLen_or_IndPtr* 自变量包含列中余下的字节数。 如果列的数据值为空，则值是 SQL_NULL_DATA。如果此指针是 NULL，而 SQLFetch() 获取到包含空数据的列，则此函数会失败，原因是没有办法报告这种情况。 如果 SQLFetch() 取装到包含二进制数据的列，则指向 *StrLen_or_IndPtr* 的指针不得是 NULL，否则此函数会失败，原因是它没有其它办法通知应用程序在 *TargetValuePtr* 缓冲区中检索到的数据的长度。

用法:

如果使用 SQLFetch()，则可以对同一结果集配合使用 SQLGetData() 和 SQLBindCol()。一般的步骤是:

1. SQLFetch() 进至第一行，检索该第一行，并传送绑定列的数据。
2. SQLGetData() 传送指定列的数据。
3. SQLGetData() 对所需的每个列重复步骤 2。
4. SQLFetch() 进至下一行，检索该下一行，并传送绑定列的数据。
5. 对结果集中的每一行重复步骤 2、3 和 4，或者重复至不再需要该结果集为止。

要废弃通过检索所得的列数据部分，应用程序可以调用 SQLGetData()，并将 *ColumnNumber* 设置为感兴趣的下一个列位置。要废弃尚未对整行检索的数据，应用程序应调用 SQLFetch() 来进至下一行；或者，如果不再需要结果集中的数据，则调用 SQLFreeStmt()。

TargetType 输入自变量确定在将列数据放入由 *TargetValuePtr* 指向的存储区之前所需的数据转换（如果有的话）的类型。

除非要检索的列数据是二进制数据，否则 *TargetValuePtr* 中返回的值以空结束。

截断数字数据类型时，如果截断小数点右面的位，则会将此截断报告为警告。如果截断在小数点左边发生，则会返回错误。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

如果 SQLGetData() 检索到长度为零的字符串，则返回 SQL_SUCCESS。如果出现这种情况，则 *StrLen_or_IndPtr* 包含 0，*TargetValuePtr* 包含 null 终止符。

如果先前的 SQLFetch() 调用失败，则不要调用 SQLGetData()，原因是未定义结果。

诊断:

表 71. SQLGetData SQLSTATE

SQLSTATE	描述	解释
01004	数据被截断。	对指定列（*ColumnNumber*）返回的数据被截断。字符串或数值在右端被截断。返回 SQL_SUCCESS_WITH_INFO。
07006	转换无效。	不能将数据值转换为自变量 *TargetType* 指定的 C 数据类型。 先前对同一 *ColumnNumber* 值调用了此函数，但 *TargetType* 值不同。
22002	指定的输出或指示符缓冲区无效。	对 *StrLen_or_IndPtr* 自变量指定的指针值是空指针，列的值为空。报告 SQL_NULL_DATA 是没有意义的。
22005	赋值出错。	返回值与 *TargetType* 自变量指示的数据类型不兼容。
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY002	列号无效。	指定的列小于 0 或大于结果列的数目。
HY003	程序类型超出范围。	*TargetType* 不是有效的数据类型或 SQL_C_DEFAULT。

表 71. SQLGetData SQLSTATE (续)

SQLSTATE	描述	解释
HY010	函数顺序错误。	调用该函数之前未首先调用 SQLFetch()。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY090	字符串或缓冲区长度无效。	*BufferLength* 自变量的值小于 0，*TargetType* 自变量是 SQL_C_CHAR 或 SQL_C_BINARY，或者 *TargetType* 是 SQL_C_DEFAULT，而缺省类型是 SQL_C_CHAR、SQL_C_BINARY 或 SQL_C_DBCHAR 之一。
HYC00	驱动程序不起作用。	DB2 CLI 识别但不支持指定数据类型的 SQL 数据类型。 DB2 CLI 或数据源不能执行所请求的从 SQL 数据类型到应用程序数据 *TargetType* 的转换。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 182 页的『SQLBindCol – 将列绑定至应用程序变量』

SQLGetDiagRec – 获取诊断记录的多个字段设置

用途:

规范:	**DB2 CLI** 5.0	**ODBC** 3.0	

SQLGetDiagRec() 返回包含错误、警告和状态信息的诊断记录的 SQLSTATE 字段当前值。

在调用此函数之前，必须使用 SQLAllocHandle() 分配连接句柄。

语法:

```
SQLRETURN   SQLGetDiagRec    (SQLSMALLINT       HandleType,
                              SQLHANDLE         Handle,
                              SQLSMALLINT       RecNumber,
                              SQLCHAR           *SQLState,
                              SQLINTEGER        *NativeErrorPtr,
                              SQLCHAR           *MessageText,
                              SQLSMALLINT       BufferLength,
                              SQLSMALLINT       *TextLengthPtr);
```

函数自变量:

表 72. SQLGetDiagRec 自变量

数据类型	自变量	使用	描述
SQLSMALLINT	*HandleType*	输入	句柄类型标识，它描述期望诊断的句柄的类型。可以是 SQL_HANDLE_STMT 或 SQL_HANDLE_DBC。

表 72. SQLGetDiagRec 自变量 (续)

数据类型	自变量	使用	描述
SQLHANDLE	*Handle*	输入	具有 *HandleType* 所指示类型的诊断数据结构的句柄。
SQLSMALLINT	*RecNumber*	输入	指示应用程序从中查找信息的状态记录。状态记录必须是 1。
SQLCHAR	*SQLState*	输出	指向一个缓冲区的指针，关于诊断记录 *RecNumber* 的 5 字符 SQLSTATE 代码将在这个缓冲区中返回。前两个字符指示类；接着的 3 个字符指示子类。
SQLINTEGER	*NativeErrorPtr*	输出	指向一个缓冲区的指针，特定于数据源的本机错误码将在此缓冲区中返回。
SQLCHAR	*MessageText*	输出	指向一个缓冲区的指针，错误消息正文将在此缓冲区中返回。SQLGetDiagRec() 返回的字段包含在一个文本字符串中。
SQLINTEGER	*BufferLength*	输入	*MessageText* 缓冲区的长度（以字节计）。
SQLSMALLINT	*TextLengthPtr*	输出	指向一个缓冲区的指针，可以在 *MessageText* 中返回的总字节数（不包括 null 终止符所需的字节数）将在此缓冲区中返回。如果可供返回的字节数大于 *BufferLength*，则 *MessageText* 中的错误消息正文被截断为 *BufferLength* 减去 null 终止字符的长度。

用法:

应用程序通常在先前 DB2 CLI 函数调用返回除 SQL_SUCCESS 外的任何内容时调用 SQLGetDiagRec()。

SQLGetDiagRec() 返回包含诊断数据结构记录的多个字段的字符串。

在 DB2 Everyplace 的版本 8.1 扩展了 SQLGetDiagRec() 的功能。现在可以返回下列 SQLSTATE: 57011、HY024、HY092、HY000 和 HY012。有关这些 SQLSTATE 的更多信息，请参阅第 164 页的『SQLState 列表』。

SQLGetDiagRec() 只检索最近与 *Handle* 自变量中指定的句柄相关联的诊断信息。如果应用程序调用除 SQLGetDiagRec() 外的任何函数，则对同一句柄的先前调用所返回的任何诊断信息丢失。

HandleType 自变量

每种句柄类型都可以有与之相关联的诊断信息。*HandleType* 自变量指示 *Handle* 的句柄类型。DB2 Everyplace 支持语句句柄和连接句柄。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

SQLGetDiagRec() 不向自己公布错误值。它使用下列返回值来报告它自己的执行结果:

SQL_SUCCESS

函数成功返回了诊断信息。

SQL_SUCCESS_WITH_INFO

MessageText 缓冲区太小，无法存放请求的诊断消息。未生成诊断记录。要确定是否发生了截断，应用程序必须将 *BufferLength* 与可用的实际字节数作比较（该字节数被写至 *StringLengthPtr*）。

SQL_INVALID_HANDLE

HandleType 和 *Handle* 指示的句柄不是有效的句柄。

SQL_ERROR

发生了下列其中一种情况:

- *RecNumber* 是负数或 0。
- *BufferLength* 小于 0。

SQL_NO_DATA

RecNumber 大于对 *Handle* 中指定的句柄存在的诊断记录数。如果没有 *Handle* 的诊断记录，则此函数还对任何正的 *RecNumber* 返回 SQL_NO_DATA。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLGetInfo - 获取概要信息

用途:

规范:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetInfo() 返回关于应用程序连接至的 DBMS 的概要信息（包括受支持的数据转换）。

语法:

```
SQLRETURN   SQLGetInfo      (
                SQLHDBC          ConnectionHandle,  /* hdbc */
                SQLUSMALLINT         InfoType,          /* fInfoType */
                SQLPOINTER       InfoValuePtr,    /* rgbInfoValue */
                SQLSMALLINT      BufferLength,       /* cbInfoValueMax */
                SQLSMALLINT    *FAR StringLengthPtr,       /* pcbInfoValue */
```

函数自变量:

表 73. SQLGetInfo 自变量

数据类型	自变量	用法	描述
SQLHDBC	*ConnectionHandle*	输入	数据库连接句柄
SQLUSMALLINT	*InfoType*	输出	期望的信息的类型。自变量必须是“数据类型”和“数据转换”中的表的第一列中的其中一个值。

表 73. SQLGetInfo 自变量 (续)

数据类型	自变量	用法	描述
SQLPOINTER	*InfoValuePtr*	输出（也是输入）	指向此函数存储必要信息的缓冲区的指针。根据所检索的信息类型的不同，可以返回 5 种类型的信息: 16 位整数值 32 位整数值 32 位二进制值 32 位掩码 以 null 结束的字符串
SQLSMALLINT	*BufferLength*	输入	*InfoValuePtr* 指向的缓冲区的最大大小。
SQLSMALLINT *	*StrLen_or_IndPtr*	输出	指向一个位置的指针，此函数在该位置中返回足够返回期望信息的总字节数。在字符串输出的情况下，此大小并不包括 null 终止符。 如果 *StringLengthPtr* 所指向的位置中的值大于 *BufferLength* 中指定的 *InfoValuePtr* 缓冲区大小，则字符串输出信息将被截断为 *BufferLength* - 1 个字节，且该函数将返回 SQL_SUCCESS_WITH_INFO。

用法:

参阅 SQLGetInfo 返回的信息以获取 *InfoType* 的可能值的列表和 `SQLGetInfo()` 将对该值返回的信息的描述。

DB2 CLI 对此表中的每个 *InfoType* 返回一个值。如果 *InfoType* 不适用或不受支持，结果将取决于返回类型:

- 如果返回类型为包含 “Y” 或 “N” 的字符串，则将返回 “N”。
- 如果返回类型为包含仅 “Y” 或 “N” 以外的值的字符串，则将返回空字符串。
- 如果返回类型为 16 位整数，则返回 0（零）。
- 如果返回类型为 32 位整数，则返回 0（零）。
- 如果返回类型为 32 位掩码，则返回 0（零）。

SQLGetInfo 返回的信息

SQL_DBMS_NAME (string)

正在访问的 DBMS 产品的名称。例如: “DB2 Everyplace”。

SQL_DBMS_VER (string)

DB2 Everyplace DBMS 产品的版本。返回的信息是具有以下格式的字符串: `DB2 Everyplace Vm.v.r Build yyyy-mm-dd`，其中 `m` 是主版本，`v` 是次版本，`r` 是发行版，而 `yyyy-mm-dd` 是 ISO 格式的构建日期。

例如:

```
'DB2 Everyplace V8.1.2 Build 2003-04-01'
是 DB2 Everyplace V8.1.2，构建于 2003 年 4 月 1 日
```

注: 应用程序需要至少可以包含 39 个字符（BUFSIZE）的缓冲区。例如:

```
rc = SQLGetInfo(hdbc, SQL_DBMS_VER, buf, BUFSIZE, &len);
```

SQL_IDENTIFIER_QUOTE_CHAR (string)
指示该字符用于将定界标识括起来。

SQL_MAX_BINARY_LITERAL_LEN (32-bit unsigned integer)
32 位的不带符号的整数值，用于指定 SQL 语句中的十六进制文字的最大长度。

SQL_MAX_CHAR_LITERAL_LEN (32-bit unsigned integer)
SQL 语句中的字符文字的最大长度（以字节计）。

SQL_MAX_COLUMN_NAME_LEN (16-bit integer)
列名的最大长度（以字节计）。

SQL_MAX_COLUMNS_IN_GROUP_BY (16-bit integer)
指示 GROUP BY 子句中服务器支持的最大列数。如果没有限制，则为零。

SQL_MAX_COLUMNS_IN_INDEX (16-bit integer)
指示索引中服务器支持的最大列数。如果没有限制，则为零。

SQL_MAX_COLUMNS_IN_ORDER_BY (16-bit integer)
指示 ORDER BY 子句中服务器支持的最大列数。如果没有限制，则为零。

SQL_MAX_COLUMNS_IN_SELECT (16-bit integer)
指示选择列表中服务器支持的最大列数。如果没有限制，则为零。

SQL_MAX_CONCURRENT_ACTIVITIES (16-bit integer)
DB2 Everyplace CLI 驱动程序可支持的最大活动环境数。如果没有指定限制或限制是未知的，则该值将设置为零。

SQL_MAX_DRIVER_CONNECTIONS (16-bit integer)
每个应用程序支持的最大活动连接数。

SQL_MAX_INDEX_SIZE (32-bit unsigned integer)
指示对于索引中的组合列，服务器支持的最大大小（以字节计）。如果没有限制，则为零。

SQL_MAX_ROW_SIZE (32-bit unsigned integer)
指定基本表的单行中服务器支持的最大长度（以字节计）。如果没有限制，则为零。

SQL_MAX_STATEMENT_LEN (32-bit unsigned integer)
指示 SQL 语句字符串的最大长度（以字节计），包括语句中的空格数。

SQL_MAX_TABLE_NAME_LEN (16-bit integer)
表名的最大长度（以字节计）。

SQL_MAX_TABLES_IN_SELECT (16-bit integer)
指示查询规范中的 FROM 子句中允许的最大表名数。

SQL_MAX_USER_NAME_LEN (16-bit integer)
指示用户标识允许的最大大小（以字节计）。

SQL_SEARCH_PATTERN_ESCAPE (string)
用来指定驱动程序支持作为目录函数（例如，SQLTables() 和 SQLColumns()）的转义字符的字符。

SQL_TXN_CAPABLE (16-bit integer)

指示事务是否可包含 DDL 和 / 或 DML。

- SQL_TC_NONE = 不支持事务。
- SQL_TC_DML = 事务只能包含 DML 语句（SELECT、INSERT、UPDATE 和 DELETE 等等）。在事务中遇到的 DDL 语句（CREATE TABLE 和 DROP INDEX 等等）将导致错误。
- SQL_TC_DDL_COMMIT = 事务只能包含 DML 语句。在事务中遇到的 DDL 语句将导致落实该事务。
- SQL_TC_DDL_IGNORE = 事务只能包含 DML 语句。在事务中遇到的 DDL 语句将被忽略。
- SQL_TC_ALL = 事务可以以任何顺序包含 DDL 和 DML 语句。

SQL_USER_NAME (string)

在特定数据库中使用的用户名。这是在 SQLConnect() 调用上指定的标识。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLGetStmtAttr – 获取语句属性的当前设置

用途:

规范:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetStmtAttr() 返回语句属性的当前设置。

语法:

```
SQLRETURN   SQLGetStmtAttr       (
                SQLHSTMT             StatementHandle,
                                SQLINTEGER        Attribute,
                                SQLPOINTER        ValuePtr,
                SQLINTEGER           BufferLength,
                SQLINTEGER          *StringLengthPtr);
```

函数自变量:

表 74. SQLGetStmtAttr 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。

表 74. SQLGetStmtAttr 自变量 (续)

数据类型	自变量	使用	描述
SQLINTEGER	*Attribute*	输入	要检索的属性。
SQLPOINTER	*ValuePtr*	输出	指向缓冲区的指针，将在该缓冲区中返回在 *Attribute* 中指定的属性值。
SQLINTEGER	*BufferLength*	输入	如果 *Attribute* 是一个 ODBC 定义的属性，且 *ValuePtr* 指向字符串或二进制缓冲区，则此自变量应是 **ValuePtr* 的长度。 如果 *Attribute* 是 ODBC 定义的属性且 **ValuePtr* 是一个整数，则忽略 *BufferLength*。如果 *Attribute* 是 DB2 CLI 属性，则应用程序通过设置 *BufferLength* 自变量指示该属性的特性。*BufferLength* 可以具有以下值: • 如果 **ValuePtr* 是指向字符串的指针，则 *BufferLength* 是字符串或 SQL_NTS 的长度。 • 如果 **ValuePtr* 是指向二进制缓冲区的指针，则应用程序将 SQL_LEN_BINARY_ATTR（长度）宏的结果放置在 *BufferLength* 中。 • 如果 **ValuePtr* 是指向除字符串或二进制字符串以外的值的指针，则 *BufferLength* 的值应为 SQL_IS_POINTER。 • 如果 **ValuePtr* 包含定长数据类型，则 *BufferLength* 是 SQL_IS_INTEGER 或 SQL_IS_UINTEGER（在适当的时候）。
SQLSMALLINT	**StringLengthPtr*	输出	指向一个缓冲区的指针，在该缓冲区中返回可在 *ValuePtr* 中返回的字节总数（排除 null 终止符）。如果这是空指针，不返回长度。如果属性值是字符串，可返回的字节数大于或等于 *BufferLength*，**ValuePtr* 中的数据截断为 *BufferLength* 减去 null 终止字符的长度且由 DB2 CLI 以 null 结束。

用法:

对 `SQLGetStmtAttr()` 的调用在 **ValuePtr* 中返回在 *Attribute* 中指定的语句属性的值。在 DB2 Everyplace 中，该值是 32 位值，未使用 *BufferLength* 和 *StringLengthPtr* 自变量。

可以通过 `SQLGetStmtAttr()` 检索下列语句属性。有关属性的描述，请参阅第 244 页的『SQLSetStmtAttr – 设置与语句相关的选项』。

- SQL_ATTR_CURSOR_SCROLLABLE (DB2 CLI/ODBC)
- SQL_ATTR_CURSOR_SENSITIVITY (DB2 CLI/ODBC)

- SQL_ATTR_CURSOR_TYPE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_BIND_TYPE (DB2 CLI/ODBC)
- SQL_ATTR_ROW_NUMBER (DB2 CLI/ODBC)
- SQL_ATTR_DELETE_MODE (DB2 Everyplace)
- SQL_ATTR_DIRTYBIT_SET_MODE (DB2 Everyplace)
- SQL_ATTR_READ_MODE (DB2 Everyplace)
- SQL_ATTR_REORG_MODE (DB2 Everyplace)

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 75. SQLGetStmtAttr SQLSTATE

SQLSTATE	描述	解释
01000	警告。	参考消息。（函数返回 SQL_SUCCESS_WITH_INFO。）
01004	数据被截断。	在 **ValuePtr* 中返回的数据截断为 *BufferLength* 减去 null 终止字符的长度。在 **StringLengthPtr* 中返回未截断的字符串值的长度。（函数返回 SQL_SUCCESS_WITH_INFO。）
24000	游标状态无效。	自变量 *Attribute* 是 SQL_ATTR_ROW_NUMBER 且未打开游标，或在结果集启动之前或结果集结束之后定位游标。
HY000	一般错误。	出错，该错误没有特定的 SQLSTATE。**MessageText* 缓冲区中由 `SQLGetDiagRec()` 返回的错误消息描述该错误及其原因。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	为 *StatementHandle* 调用异步执行函数且当调用此函数时仍在执行该函数。 为 *StatementHandle* 调用 `SQLExecute()` 或 `SQLExecDirect()` 并返回 SQL_NEED_DATA。在对所有“处理数据”参数或列发送数据时调用此函数。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。
HY090	字符串或缓冲区长度无效。	对 *BufferLength* 自变量指定的值小于 0。
HY092	选项类型超出范围。	对自变量 *Attribute* 指定的值对于此版本的 DB2 CLI 无效
HY109	无效的游标位置。	*Attribute* 自变量是 SQL_ATTR_ROW_NUMBER，已删除了该行或不能取装该行。

表 75. SQLGetStmtAttr SQLSTATE (续)

SQLSTATE	描述	解释
HYC00	驱动程序不起作用。	对自变量 *Attribute* 指定的值是该版本的 DB2 CLI 的有效 DB2 CLI 属性，但不受数据源支持。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 241 页的『SQLSetConnectAttr - 设置与连接相关的选项』
- 第 244 页的『SQLSetStmtAttr - 设置与语句相关的选项』

SQLNumParams - 获取 SQL 语句中的参数数目

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLNumParams() 返回 SQL 语句中的参数标记数目。

语法:

```
SQLRETURN   SQLNumParams     (SQLHSTMT          StatementHandle,
                              SQLSMALLINT  FAR  *ParameterCountPtr);
```

函数自变量:

表 76. SQLNumParams 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLSMALLINT	*ParameterCountPtr*	输出	语句中的参数数目。

用法:

只能在准备与 *StatementHandle* 相关联的语句之后调用此函数。如果语句不包含任何参数标记，则 *ParameterCountPtr* 设置为零。

应用程序可调用此函数来确定与该语句句柄相关联的 SQL 语句需要多少个 SQLBindParameter() 调用。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 77. SQLNumParams SQLSTATE

SQLSTATE	描述	解释
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	在对指定 *StatementHandle* 调用 SQLPrepare() 之前调用了此函数。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持该函数的执行或完成所需的内存。

限制:

无。

相关参考:

- 第 185 页的『SQLBindParameter - 将参数标记绑定至缓冲区』
- 第 236 页的『SQLPrepare - 准备语句』

SQLNumResultCols - 获取结果列数

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	

SQLNumResultCols() 返回与输入语句句柄相关联的结果集中的列数。

在调用此函数之前，必须调用 SQLPrepare() 或 SQLExecDirect()。

在调用此函数之前，可以调用 SQLColAttribute() 或绑定列函数之一。

语法:

```
SQLRETURN   SQLNumResultCols (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLSMALLINT  FAR  *ColumnCountPtr);  /* pccol */
```

函数自变量:

表 78. SQLNumResultCols 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLSMALLINT *	*ColumnCountPtr*	输出	结果集中的列数。

用法:

如果对输入语句句柄执行的最后一个语句或函数未生成结果集，则此函数将输出自变量设置为 0。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

诊断:

表 79. SQLNumResultCols SQLSTATE

SQLSTATE	描述	解释
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	在对 *StatementHandle* 调用 SQLPrepare() 或 SQLExecDirect() 之前调用了该函数。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 182 页的『SQLBindCol - 将列绑定至应用程序变量』
- 第 196 页的『SQLDescribeCol - 返回列的一组属性』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 223 页的『SQLGetData - 从列中获取数据』

SQLPrepare - 准备语句

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLPrepare() 将一个 SQL 语句与输入语句句柄相关联，并将该语句发送至要准备的 DBMS。应用程序可以通过将语句句柄传送至其它函数来引用这个准备好的语句。

如果该语句句柄先前是配合查询语句（或任何返回结果集的函数）使用的，则在调用 SQLPrepare() 之前，必须调用 SQLFreeStmt()。

语法:

```
SQLRETURN   SQLPrepare      (SQLHSTMT          StatementHandle,   /* hstmt */
                             SQLCHAR      FAR  *StatementText,    /* szSqlStr */
                             SQLINTEGER        TextLength);       /* cbSqlStr */
```

函数自变量:

表 80. SQLPrepare 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。

表 80. SQLPrepare 自变量 (续)

数据类型	自变量	使用	描述
SQLCHAR	*StatementText*	输入	SQL 语句字符串
SQLINTEGER	*TextLength*	输入	*StatementText* 自变量的内容的长度。 这必须设置为 *szSqlstr* 中 SQL 语句的精确长度，或者，如果语句文本以 null 结束，则必须设置为 SQL_NTS。

用法:

在使用 SQLPrepare() 准备语句之后，应用程序可以通过调用下列函数之一来请求关于结果集格式的信息（如果语句是查询的话）：

- SQLNumResultCols()
- SQLDescribeCol()

SQL 语句字符串可以包含参数标记。参数标记由 ? 字符表示，用来指示语句中的一个位置，调用 SQLExecute() 时，将会在该位置中换上应用程序提供的值。绑定参数函数 SQLBindParameter() 使应用程序值与每个参数标记绑定在一起（产生关联），指示在传送数据时是否应执行任何数据转换。

在调用 SQLExecute() 之前，必须绑定所有参数。有关更多信息，参阅第 202 页的『SQLExecute - 执行语句』。

参阅 *DB2 Universal Database SQL Reference* 中有关 PREPARE 语句的章节以获取与参数标记相关的规则的信息。

在应用程序处理完 SQLExecute() 调用的结果之后，它可以再次执行该语句，并指定新的（或相同的）参数值。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 81. SQLPrepare SQLSTATE

SQLSTATE	描述	解释
42nnn	语法错误。	**42**nnn SQLSTATE 指示与语句相关的各种语法或访问问题。字符 nnn 指的是任何具有该类代码的 SQLSTATE。示例: **42**nnn 指的是 **42** 类中的任何 SQLSTATE。
58004	意外系统故障。	不可恢复的系统错误。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY009	自变量值无效。	*StatementText* 是空指针。
HY013	发生意外的内存处理错误。	DB2 CLI 无法访问支持函数的执行或完成所需的内存。

表 81. SQLPrepare SQLSTATE (续)

SQLSTATE	描述	解释
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效。	*TextLength* 自变量小于 1，但不等于 SQL_NTS。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 185 页的『SQLBindParameter - 将参数标记绑定至缓冲区』
- 第 196 页的『SQLDescribeCol - 返回列的一组属性』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 202 页的『SQLExecute - 执行语句』
- 第 235 页的『SQLNumResultCols - 获取结果列数』

SQLPrimaryKeys - 获取表的主键列

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLPrimaryKeys() 返回由表的主键组成的列名的列表。此信息在 SQL 结果集中返回，可以使用用来处理查询生成的结果集的函数来检索此结果集。*CatalogName*、*NameLength1*、*SchemaName* 和 *NameLength2* 被忽略。返回的结果集的第 1、2 和 6 列总是长度为零的字符串。

语法:

```
SQLRETURN   SQLPrimaryKeys  (
                  SQLHSTMT          StatementHandle,  /* hstmt */
                  SQLCHAR      FAR  *CatalogName,     /* szCatalogName */
                  SQLSMALLINT       NameLength1,      /* cbCatalogName */
                  SQLCHAR      FAR  *SchemaName,      /* szSchemaName */
                  SQLSMALLINT       NameLength2,      /* cbSchemaName */
                  SQLCHAR      FAR  *TableName,       /* szTableName */
                  SQLSMALLINT       NameLength3);     /* cbTableName */
```

函数自变量:

表 82. SQLPrimaryKeys 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLCHAR*	*CatalogName*	输入	表名（该表名由 3 个部分组成）的目录限定符。 DB2 Everyplace 忽略此字段。
SQLSMALLINT	*NameLength1*	输入	*CatalogName* 的长度。DB2 Everyplace 忽略此字段。
SQLCHAR*	*SchemaName*	输入	表名的模式限定符。DB2 Everyplace 忽略此字段。
SQLSMALLINT	*NameLength2*	输入	*SchemaName* 的长度。DB2 Everyplace 忽略此字段。
SQLCHAR*	*TableName*	输入	表名。
SQLSMALLINT	*NameLength3*	输入	*TableName* 的长度。

用法:

SQLPrimaryKeys() 返回单个表中的主键列。不能使用搜索模式来指定表名。

如果指定的表不包含主键，则返回空的结果集。

对 SQLPrimaryKeys() 的调用在许多情况下都会映射至对系统目录的复杂且高成本的查询。

虽然可以添加新列，在将来的发行版中也可以更改现有列的名称，但当前列的位置不会改变。

结果集包含以下列，按 TABLE_NAME 和 ORDINAL_POSITION 排序: :

第 1 列 TABLE_CAT (VARCHAR(128))
这始终是长度为零的字符串。

第 2 列 TABLE_SCHEM (VARCHAR(128))
这始终是长度为零的字符串。

第 3 列 TABLE_NAME (VARCHAR(128) not NULL)
指定的表的名称。

第 4 列 COLUMN_NAME (VARCHAR(128) not NULL)
主键列名。

第 5 列 ORDINAL_POSITION (SMALLINT not NULL)
主键中的列序号，从 1 开始。

第 6 列 PK_NAME (VARCHAR(128))
这始终是长度为零的字符串。

DB2 CLI/ODBC 使用的列名遵循 X/Open CLI CAE 规范样式。

返回码:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 83. SQLPrimaryKey SQLSTATE

SQLSTATE	描述	解释
24000	游标状态无效。	已在该语句句柄上打开了游标。
40003 **08**S01	通信链路故障。	函数尚未完成，应用程序与数据源间的通信链路就失效了。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	执行“处理数据”的操作（SQLPrepare() 或 SQLExecDirect()）时调用了该函数。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。

表 83. SQLPrimaryKey SQLSTATE (续)

SQLSTATE	描述	解释
HY090	字符串或缓冲区长度无效。	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。

限制:

不要经常调用 SQLPrimaryKeys()，保存结果而不是重复的调用。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 212 页的『SQLForeignKeys - 获取外键列的列表』

SQLRowCount - 获取行计数

用途:

规范:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLRowCount() 返回表中某些行的数目，这些行受到对表执行的带有可滚动游标的 UPDATE、INSERT、DELETE 或 SELECT 语句的影响。

在调用此函数之前，必须调用 SQLExecute() 或 SQLExecDirect()。

语法:

```
SQLRETURN   SQLRowCount      (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLINTEGER   FAR  *RowCountPtr);     /* pcrow */
```

函数自变量:

表 84. SQLRowCount 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLINTEGER	*RowCountPtr*	输出	指向存储受影响行数的位置的指针。

用法:

如果输入语句句柄引用的上次执行的语句不是 UPDATE、INSERT 或 DELETE 语句，或者，如果该语句未成功执行，则此函数将 *RowCountPtr* 的内容设置为 -1。

其它表中任何可能受该语句影响的行都不包括在此计数中。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 85. SQLRowCount SQLSTATE

SQLSTATE	描述	解释
40003 **08**S01	通信链路发生故障。	在功能完成之前应用程序与数据源之间的通信链接失效。
58004	发生意外系统故障。	不可恢复的系统错误。
HY001	发生内存分配故障。	DB2 CLI 未能分配支持功能执行或完成所需的内存。应用程序进程有可能已用完应用程序级内存。有关进程级内存限制的信息，参见操作系统配置。
HY010	发生函数序列错误。	在对 *StatementHandle* 调用 SQLExecute() 或 SQLExecDirect() 之前调用了该函数。
HY013	发生意外的内存处理错误。	DB2 CLI 未能访问支持功能执行或完成所需的内存。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 201 页的『SQLExecDirect - 直接执行语句』
- 第 202 页的『SQLExecute - 执行语句』
- 第 235 页的『SQLNumResultCols - 获取结果列数』

SQLSetConnectAttr - 设置与连接相关的选项

用途:

规范:	DB2 CLI	ODBC 1.0	ISO CLI

SQLSetConnectAttr() 设置与连接相关的选项。

语法:

```
SQLRETURN   SQLSetConnectAttr   (SQLHDBC        ConnectionHandle,
                                 SQLINTEGER     Attribute,
                                 SQLPOINTER     ValuePtr,
                                 SQLINTEGER     StringLength);
```

函数自变量:

表 86. SQLSetConnectAttr 自变量

数据类型	自变量	使用	描述
SQLHDBC	*ConnectionHandle*	输入	连接句柄。
SQLINTEGER	*Attribute*	输入	要设置的选项。

表 86. SQLSetConnectAttr 自变量 (续)

数据类型	自变量	使用	描述
SQLPOINTER	*ValuePtr*	输入	如果 *Attribute* 是一个 ODBC 定义的属性，且 *ValuePtr* 指向字符串或二进制缓冲区，则此自变量应是 *ValuePtr* 的长度。如果 *Attribute* 是一个 ODBC 定义的属性，且 ValuePtr 是整数，则忽略 *StringLength*。 如果 *Attribute* 是一个 DB2 CLI 属性，则应用程序通过设置 *StringLength* 自变量来指示该属性的性质。*StringLength* 可以具有下列值: • 如果 *ValuePtr* 是指向字符串的指针，则 *StringLength* 是字符串或 SQL_NTS 的长度。 • 如果 *ValuePtr* 是指向二进制缓冲区的指针，则应用程序将 SQL_LEN_BINARY_ATTR(length) 宏的结果放在 *StringLength* 中。这将在 *StringLength* 中放置一个负数值。 • 如果 *ValuePtr* 是指向除字符串或二进制字符串外的值的指针，则 *StringLength* 的值应为 SQL_IS_POINTER。 • 如果 *ValuePtr* 包含定长值，则 *StringLength* 是 SQL_IS_INTEGER 或 SQL_IS_UINTEGER（在适当的时候）。
SQLINTEGER	*StringLength*	输入	如果 *ValuePtr* 指向字符串或二进制缓冲区，则此自变量应是 *ValuePtr* 的长度。如果 *ValuePtr* 是一个指针，但并非指向字符串或二进制缓冲区，则 *StringLength* 的值应当是 SQL_IS_POINTER。如果 *ValuePtr* 不是指针，则 *StringLength* 的值应当是 SQL_IS_NOT_POINTER。

用法:

连接的连接属性会一直有效，直到它们被对 `SQLSetConnectAttr()` 的另一调用更改，或是通过调用 `SQLDisconnect()` 断开了该连接。

`SQLSetConnectAttr()` 接受两种不同格式的属性信息: 以 null 结束的字符串或 32 位整数值。每一种的格式都在该属性的描述中有记载。`SQLSetConnectAttr()` 的 *ValuePtr* 自变量所指向的字符串的长度为 *StringLength*。

连接属性:

当前定义的属性显示如下。

SQL_ATTR_AUTOCOMMIT (DB2 CLI/ODBC)

一个 32 位整数值，它指定方式类型。支持的值是:

• SQL_AUTOCOMMIT_ON = 自动提交每个语句。这是缺省值。

在自动落实方式下，一个语句所执行的所有更新在该语句执行完成后都自动具有持久状态。自动落实方式是缺省行为。在缺省情况下，未启用事务支持，而且，并不保证语句级别基数。例如，下列 UPDATE 语句可能会在处理期间失败且可能只更新行的子集:

```
UPDATE T SET A = A + 1
```

更新／删除／插入操作失败有多种原因。例如，在更新期间可能违反了检查约束。因此，可以正确更新表的一部分，而表的其余部分不能更新，并且不能回滚所作的更改。

- SQL_AUTOCOMMIT_OFF = 应用程序必须手工显式落实或回滚事务。落实或回滚事务是通过调用 SQLEndTran() 来完成的。有关使用 SQLEndTran() 的更多信息，请参阅第 199 页的『SQLEndTran - 请求 COMMIT 或 ROLLBACK』。

  在手工落实方式下，事务是在第一次访问数据库时通过使用 SQLPrepare() 和 SQLExecDirect() 隐式启动的，在此情况下，即使调用失败，事务也已经开始了。当您使用 SQLEndTran() 来对事务进行 ROLLBACK 或 COMMIT 操作时，事务就结束了。

  在手工落实方式下，事务可发出所有 SQL 语句，包括 DDL 和 DML（例如，CREATE TABLE 或 UPDATE 语句）。

SQL_ATTR_CONNECTION_DEAD (DB2 CLI/ODBC)

READ ONLY 32 位整数值，用来指示连接是否仍然是活动的。DB2 CLI 将返回下列其中一个值:

- SQL_CD_FALSE - 连接仍然是活动的。
- SQL_CD_TRUE - 连接已死。

SQL_ATTR_LOGIN_TIMEOUT (DB2 CLI/ODBC)

一个 32 位整数值，对应于在将控制返回给应用程序之前等待登录请求完成的秒数。

SQL_ATTR_FILENAME_FORMAT (DB2 Everyplace)

一个 32 位整数指定 DB2e 数据库引擎是以长格式还是以 8.3 格式创建文件名。当调用 SQLSetConnectAttr 时，仅当连接的路径中没有目录文件，才允许应用程序更改文件名格式。如果由于预先存在的目录文件而拒绝更改文件名格式，将返回具有 SQLState HY000 的 SQL_ERROR。例如，如果应用程序连接至已存在 DB2 Everyplace 目录文件的路径，任何更改文件格式的尝试都将失败。如果应用程序连接至不存在 DB2 Everyplace 目录文件的路径且在试图在第一个 CREATE TABLE 语句之后更改文件名格式，SQLSetConnectAttr 也将返回 SQL_ERROR。这是因为在首个 CREATE TABLE 语句期间会创建目录文件，在创建目录文件之后不允许更改文件名格式。缺省文件名格式取决于平台。SQL_FILENAME_FORMAT_LONG 当前是受支持的所有平台的缺省值。

属性值:

SQL_FILENAME_FORMAT_LONG - 将以长文件名格式创建文件。

SQL_FILENAME_FORMAT_83 - 将以 8.3 文件名格式创建文件。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

诊断:

表 87. SQLSetConnectAttr SQLSTATE

SQLSTATE	描述	解释
HY000	一般错误。	不能更改文件名格式。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效。	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。

限制:

- 一个事务中可以更新的表的数目是有限的。DB2 Everyplace 允许事务中最多有 256 个打开文件（假定操作系统也允许这么多的打开文件）。这通常意味着可更新大约 100 个表。表的数目取决于索引使用情况以及语句句柄数。活动语句句柄越多，可以潜在地更新的表越少。即使在一个事务中多次访问和／或更新某个表，对这个表也只计一次。
- 对 DB2 Everyplace 添加了事务，以允许在许多个表中一致地更新和插入若干个相关记录。在应用程序落实事务之后，便将更改写至数据表。
- 如果应用程序过早终止，而没有落实当前事务，则会自动回滚该事务内的更新。
- 在 SQLEndTran 返回之后，落实或回滚事务。
- 当应用程序连接至（在活动事务期间）过早终止的数据库时，会对事务进行恢复。数据库使用下列逻辑恢复事务:
  - 如果事务未完成，则不更新数据库。
  - 如果事务完成，则会使用来自该事务的信息更新数据库。
  - 如果恢复被中断，则在下一次连接时执行适当的操作。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』
- 第 199 页的『SQLEndTran - 请求 COMMIT 或 ROLLBACK』

SQLSetStmtAttr - 设置与语句相关的选项

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	**ISO CLI**

`SQLSetStmtAttr()` 设置与语句相关的选项。

语法:

```
SQLRETURN   SQLSetStmtAttr   (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        StringLength);
```

函数自变量:

表 88. SQLSetStmtAttr 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLINTEGER	*Attribute*	输入	要设置的选项。
SQLPOINTER	*ValuePtr*	输入	如果 *Attribute* 是一个 ODBC 定义的属性，且 *ValuePtr* 指向字符串或二进制缓冲区，则此自变量应是 **ValuePtr* 的长度。如果 *Attribute* 是一个 ODBC 定义的属性，且 ValuePtr 是整数，则忽略 *StringLength*。 如果 *Attribute* 是一个 DB2 CLI 属性，则应用程序通过设置 *StringLength* 自变量来指示该属性的性质。*StringLength* 可以具有下列值: • 如果 *ValuePtr* 是指向字符串的指针，则 *StringLength* 是字符串或 SQL_NTS 的长度。 • 如果 *ValuePtr* 是指向二进制缓冲区的指针，则应用程序将 SQL_LEN_BINARY_ATTR(length) 宏的结果放在 *StringLength* 中。这将在 *StringLength* 中放置一个负数值。 • 如果 *ValuePtr* 是指向除字符串或二进制字符串外的值的指针，则 *StringLength* 的值应为 SQL_IS_POINTER。 • 如果 *ValuePtr* 包含定长值，则 *StringLength* 是 SQL_IS_INTEGER 或 SQL_IS_UINTEGER。
SQLINTEGER	*StringLength*	输入	如果 *ValuePtr* 指向字符串或二进制缓冲区，则此自变量应是 *ValuePtr* 的长度。如果 *ValuePtr* 是一个指针，但并不指向字符串或二进制缓冲区，则 *StringLength* 的值应当是 SQL_IS_POINTER。如果 *ValuePtr* 不是指针，则 *StringLength* 的值应当是 SQL_IS_NOT_POINTER。

用法:

语句的语句属性在被另一 `SQLSetStmtAttr()` 调用更改之前，或通过调用 `SQLFreeHandle()` 删除该语句之前，语句属性保持有效。调用 `SQLFreeStmt()` 时，如果指定了 SQL_CLOSE、SQL_UNBIND 或 SQL_RESET_PARAMS 选项，则不会将语句属性复位。

如果数据源不支持 *ValuePtr* 中指定的值，则某些语句属性支持类似值的替换。在这样的情况下，DB2 CLI 返回 SQL_SUCCESS_WITH_INFO 和 SQLSTATE 01S02（选项值已更改）。例如，如果 Attribute 是 SQL_ATTR_CONCURRENCY，*ValuePtr* 是 SQL_CONCUR_ROWVER，而数据源不支持这一点，则 DB2 CLI 替换

SQL_CONCUR_VALUES 并返回 SQL_SUCCESS_WITH_INFO。要确定替换的值，应用程序调用 SQLGetStmtAttr()。用 *ValuePtr* 设置的信息格式取决于指定的 Attribute。

SQLSetStmtAttr() 接受两种不同格式之一的属性信息: null 终止的字符串或 32 位整数值。每一种的格式都在该属性的描述中有记载。此格式适用于对 SQLGetStmtAttr() 中每个属性返回的信息。SQLSetStmtAttr() 的 *ValuePtr* 自变量所指向的字符串的长度为 *StringLength*。

脏位:

DB2 Everyplace 使用脏位跟踪对记录的更改。SQL_ATTR_DELETE_MODE、SQL_ATTR_READ_MODE 和 SQL_ATTR_DIRTYBIT_SET_MODE 语句属性影响脏位的性能。下表显示在记录上执行某些数据库操作后脏位的状态。该表假定 SQL_ATTR_DIRTYBIT_SET_MODE 参数被设置为带有系统维护的脏位的 SQL_DIRTYBIT_SET_BY_SYSTEM。

表 89. DB2 Everyplace 脏位状态

记录上的操作	脏位状态
清除状态（0）然后 INSERT	INSERT
清除状态（0）然后 DELETE	DELETE
清除状态（0）然后 UPDATE	UPDATE
DELETE 然后 INSERT	UPDATE
DELETE 然后 DELETE	不可应用
DELETE 然后 UPDATE	不可应用
INSERT 然后 INSERT	不可应用
INSERT 然后 DELETE	物理除去记录
INSERT 然后 UPDATE	INSERT
UPDATE 然后 INSERT	不可应用
UPDATE 然后 DELETE	DELETE
UPDATE 然后 UPDATE	UPDATE

通过查询表的 $dirty 列能够获得脏位的值。例如，以下语句返回脏位和电话簿表的 NAME 列:

```
SELECT $dirty, NAME from PHONEBOOK
```

脏位可以有以下值。

表 90. DB2 Everyplace 脏位值

描述	脏位值
未更改记录 (CLEAN)	0
已删除记录 (DELETE)	1
已插入记录 (INSERT)	2
已更新记录 (UPDATE)	3

语句属性:

当前定义的属性显示如下。

SQL_ATTR_CURSOR_SCROLLABLE（DB2 CLI）

指定该应用程序需要的支持级别的 32 位整数。设置此属性会影响对 SQLExecDirect() 和 SQLExecute() 的后续调用。支持的值是:

- SQL_NONSCROLLABLE

  语句句柄上不需要可滚动游标。如果应用程序在此句柄上调用 SQLFetchScroll()，则 FetchOrientation() 的唯一有效值是 SQL_FETCH_NEXT。这是缺省值。

- SQL_SCROLLABLE

  该语句句柄上需要可滚动游标。当调用 SQLFetchScroll() 时，应用程序可以指定 FetchOrientation 的任何有效值，以便可以以除顺序方式以外的方式定位游标。

SQL_ATTR_CURSOR_SENSITIVITY（DB2 CLI）

一个 32 位整数值，指定游标是否对另一个游标的写活动敏感。支持的值是:

- SQL_UNSPECIFIED

  其它游标的写活动对当前游标具有未定义的影响。这是缺省值。

- SQL_INSENSITIVE

  其它游标的写活动对当前游标没有影响。

  注: 因为此属性值会影响性能，所以应尽量少使用。

SQL_ATTR_CURSOR_TYPE (DB2 CLI)

一个 32 位整数值，它指定游标类型。支持的值是:

- SQL_CURSOR_FORWARD_ONLY = 游标只正向滚动。这是缺省值。
- SQL_CURSOR_STATIC = 结果集中的数据是静态的。

不能对打开的游标指定此选项。

SQL_ATTR_ROW_ARRAY_SIZE (DB2 CLI)

一个 32 位整数值，它指定行集中的行数。这是每次调用 SQLFetch() 或 SQLFetchScroll() 时返回的行数。缺省值是 1。如果指定的行集大小超过数据源支持的最大行集大小，则 DB2 CLI 换上该值并返回 SQLSTATE 01S02（选项值已更改）。可对打开的游标指定此选项。

SQL_ATTR_ROW_BIND_TYPE (DB2 CLI)

一个 32 位的整数值，它设置对相关联的语句调用 SQLFetch() 或 SQLFetchScroll() 时要使用的绑定方向。通过在 *ValuePtr* 中提供已定义的常量 SQL_BIND_BY_COLUMN 选择列智能绑定。*ValuePtr* 中指定的长度必须包括用于所有绑定列的空间以及任何结构或缓冲区的填充，以确保当绑定列的地址以指定的长度增加时，结果会指向下一行中的同一列的起始位置。当将 **sizeof** 运算符与 ANSI C 中的结构或连接配合使用时，此行为能得到保证。列智能绑定是 SQLFetchScroll() 的缺省绑定方向。

SQL_ATTR_ROW_NUMBER（DB2 CLI）

一个 32 位整数值，它是整个结果集中当前行的编号。如果不能确定当前行的编号或没有当前行，则 DB2 CLI 返回 0。可以通过调用 SQLGetStmtAttr() 检索此属性，但不能调用 SQLSetStmtAttr() 进行设置。

SQL_ATTR_ROW_STATUS_PTR (DB2 CLI)

一个 16 位无符号整数值，它指向 UWORD 值的一个数组，该数组包含调用 SQLFetch() 或 SQLFetchScroll() 后的行状态值。该数组具有的元素数目与行

集中的行数相同。可以将此语句属性设置为空指针，在这种情况下，DB2 CLI 不返回行状态值。此属性可以随时设置，但新值只能在下次调用 SQLFetch() 或 SQLFetchScroll() 之后才可使用。

SQL_ATTR_ROWS_FETCHED_PTR (DB2 CLI)

一个 32 位无符号整数值，它指向一个缓冲区，调用 SQLFetch() 或 SQLFetchScroll() 后取装的行数在该缓冲区中返回。

SQL_ATTR_DELETE_MODE (DB2 Everyplace)

支持的值是:

- SQL_DELETE_MARK_ONLY

  这是系统缺省值。在执行删除 SQL 语句时，记录仅被标记为“删除”。如果设置了 SQL_READ_INCLUDE_MARKED_DELETE，则该记录的内容仍然可读。

- SQL_DELETE_PHYSICAL_REMOVE

  删除 SQL 语句会上物理上除去满足 WHERE 子句条件的记录，而不管其脏位如何。

例如，使用以下语法物理除去某些记录，而忽略其脏位的状态:

```
SQLSetStmtAttr (stmt, SQL_ATTR_DELETE_MODE, SQL_DELETE_PHYSICAL_REMOVE, 0)
```

然后执行以下 SQL 语句以从表 T 中删除 X 不等于 0 的所有记录:

```
DELETE T WHERE X<>0
```

SQL_ATTR_DIRTYBIT_SET_MODE (DB2 Everyplace)

一个 32 位整数值，它指定游标类型。支持的值是:

- SQL_DIRTYBIT_SET_BY_SYSTEM

  这是系统缺省值。插入、更新或删除的记录的脏位分别设置为 INSERT、UPDATE 或 DELETE。当设置了 SQL_DIRTYBIT_SET_BY_SYSTEM 时，不允许对 $dirty 列执行 UPDATE。

- SQL_DIRTYBIT_SET_BY_APPLICATION

  应用程序负责在插入、更新或删除记录时设置脏位。每个操作的语义如下:

  UPDATE

  系统完全按照应用程序所指定的那样来设置脏位。例如，如果应用程序执行以下语句，则会将表中的所有记录复位为 0（CLEAN）:

  ```
  UPDATE T SET $dirty=0 WHERE $dirty>0
  ```

  INSERT

  新插入的记录的脏位设置为 CLEAN。

  DELETE

  如果设置了 SQL_DELETE_PHYSICAL_REMOVE，DELETE 在物理上从数据库除去记录。否则，$dirty 列的值设置为 DELETE 且记录仍保留在数据库中。

例如，要清除记录的脏位，使用以下语句:

```
SQLSetStmtAttr (stmt, SQL_ATTR_DIRTYBIT_SET_MODE,
SQL_DIRTYBIT_SET_BY_APPLICATION, 0)
```

然后执行以下 SQL 语句:

```
UPDATE T SET $DIRTY=0 WHERE $DIRTY>0
```

一般说来，如果在跟踪最终用户所作的数据库更新时不需要脏位，应用程序可设置 SQL_DIRTYBIT_SET_BY_APPLICATION。

SQL_ATTR_READ_MODE (DB2 Everyplace)

一个 32 位整数值，它指定游标类型。支持的值是:

- SQL_READ_EXCLUDE_MARKED_DELETE

  这是系统缺省值。其脏位被设置为“删除”的所有记录将不为 SQL 所知。

- SQL_READ_INCLUDE_MARKED_DELETE

  一旦设置了此项，其脏位被设置为 DELETE 的记录对 SQL SELECT 语句是可视的。应用程序可通过检查记录的脏位来将这些已删除记录与其它记录区别开。

例如，使用以下语句来读取设置了脏位的所有记录，包括那些脏位被标记为 DELETE 的记录:

```
SQLSetStmtAttr (stmt, SQL_ATTR_READ_MODE, SQL_READ_INCLUDE_MARKED_DELETE, 0)
```

然后执行以下 SQL 语句以检索所有记录:

```
SELECT * FROM T WHERE $dirty<>0
```

SQL_ATTR_REORG_MODE（DB2 Everyplace）

一个 32 位的整数值，指定是否对用户创建的表自动执行数据库重组以及是否允许显式的 REORG SQL 语句。支持的值是:

- SQL_REORG_ENABLED – 这是系统缺省值。DB2 Everyplace 可以执行数据库重组，用户也可以使用 REORG SQL 语句以显式方式执行重组。
- SQL_REORG_DISABLED – REORG SQL 语句受到限制，禁止对用户创建的表自动执行数据库重组。

不能对打开的游标指定此选项。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 91. SQLSetStmtAttr SQLSTATE

SQLSTATE	描述	解释
24000	游标状态无效。	已在该语句句柄上打开了游标。
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY010	函数顺序错误。	执行“处理数据”的操作（SQLPrepare() 或 SQLExecDirect()）时调用了该函数。 在 BEGIN COMPOUND 和 END COMPOUND SQL 操作时调用了该函数。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。
HY090	字符串或缓冲区长度无效。	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。

限制:

无。

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

SQLTables – 获取表信息

用途:

规范:	**DB2 CLI** 2.1	**ODBC** 1.0	

SQLTables() 返回表名列表和存储在连接的数据源的系统目录中的相关信息。表名列表将作为结果集返回，可使用用于处理查询生成的结果集的函数来检索它。

语法:

```
SQLRETURN   SQLTables        (
                SQLHSTMT           StatementHandle,   /* hstmt */
                SQLCHAR      FAR   *CatalogName,      /* szCatalogName */
                SQLSMALLINT        NameLength1,       /* cbCatalogName */
                SQLCHAR      FAR   *SchemaName,       /* szSchemaName */
                SQLSMALLINT        NameLength2,       /* cbSchemaName */
                SQLCHAR      FAR   *TableName,        /* szTableName */
                SQLSMALLINT        NameLength3,       /* cbTableName */
                SQLCHAR      FAR   *TableType,        /* szTableType */
                SQLSMALLINT        NameLength4);      /* cbTableType */
```

函数自变量:

表 92. SQLTables 自变量

数据类型	自变量	使用	描述
SQLHSTMT	*StatementHandle*	输入	语句句柄。
SQLCHAR	*CatalogName*	输入	可能包含用来限定结果集的 *pattern-value* 的缓冲区。*Catalog* 是由 3 部分组成的表名的第 1 部分。 此字段将被 DB2 Everyplace 忽略。
SQLSMALLINT	*NameLength1*	输入	*CatalogName* 的长度。 此字段将被 DB2 Everyplace 忽略。
SQLCHAR	*SchemaName*	输入	可能包含用来按模式名限定结果集的 *pattern-value* 的缓冲区。 此字段将被 DB2 Everyplace 忽略。
SQLSMALLINT	*NameLength2*	输入	*SchemaName* 的长度。 此字段将被 DB2 Everyplace 忽略。
SQLCHAR	*TableName*	输入	可能包含用来按表名限定结果集的 *pattern-value* 的缓冲区。
SQLSMALLINT	*NameLength3*	输入	*TableName* 的长度。

表 92. SQLTables 自变量 (续)

数据类型	自变量	使用	描述
SQLCHAR	*TableType*	输入	DB2 Everyplace 仅支持类型 TABLE。此字段将被 DB2 Everyplace 忽略。
SQLSMALLINT	*NameLength4*	输入	此字段将被 DB2 Everyplace 忽略。

注意，*TableName* 自变量将接受搜索模式。

用法:

将在结果集中返回表信息，在该结果集中每个表由结果集的一行表示。

有时应用程序会使用空指针 *TableName* 自变量调用 SQLTables()，这样就不会尝试限制返回的结果集。对于包含大量表的一些数据源，此方案使得结果集极其大且检索时间会很长。

SQLTables() 返回的结果集包含的列以给定顺序列示在表 93 中。这些行是按 TABLE_NAME 排序的。

最好不要经常调用 SQLTables()，原因是在许多情况下它们都映射至对系统目录的复杂且高成本的查询。应保存结果而不是重复调用。

已使用最大长度属性 128 声明目录函数结果集的 VARCHAR 列，以便与 SQL92 限制一致。因为 DB2 名称少于 128 个字符，所以应用程序可选择总是为输出缓冲区保留 128 个字符（加上 null 终止符），或者使用 SQL_MAX_TABLE_NAME_LEN 调用 SQLGetInfo() 以确定连接的 DBMS 支持的 TABLE_NAME 列的实际长度。

表 93. SQLTables 返回的列

列名	数据类型	描述
TABLE_CAT	VARCHAR (128)	这始终是长度为零的字符串。
TABLE_SCHEM	VARCHAR (128)	这始终是长度为零的字符串。
TABLE_NAME	VARCHAR (128)	表名。
TABLE_TYPE	VARCHAR (128)	标识 TABLE_NAME 列中的名称给定的类型。它始终具有字符串值 “TABLE”。
REMARKS	VARCHAR(254)	包含有关该表的描述性信息。

返回码:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

诊断:

表 94. SQLTables SQLSTATE

SQLSTATE	描述	解释
HY001	内存分配失败。	DB2 CLI 无法分配支持函数的执行或完成所需的内存。
HY014	不再有句柄。	因为内部资源问题，DB2 CLI 无法分配句柄。

表 94. SQLTables SQLSTATE (续)

SQLSTATE	描述	解释
HY090	字符串或缓冲区长度无效。	其中一个名称长度自变量的值小于 0，但不等于 SQL_NTS。 超出该数据源支持的最大值的名称长度自变量之一的有效性。最大受支持值可通过调用 SQLGetInfo() 函数获取。

限制:

无。

相关参考:

• 第 228 页的『SQLGetInfo - 获取概要信息』

由 DB2 CLI 函数进行的数据转换

DB2 CLI 管理应用程序与 DB2 Everyplace 之间数据的传送和任何必需的转换。在数据传输实际发生之前，在调用 SQLBindParameter()、SQLBindCol() 或 SQLGetData() 时指示源和/或目标的数据类型。这些函数使用符号名（如 SQL_CHAR 和 SQL_C_CHAR）来标识涉及的数据类型。

例如，要将与 SQL_VARCHAR 的 SQL 数据类型相应的参数标记绑定至应用程序的长整数 C 缓冲区类型，适当的 SQLBindParameter() 调用应为:

```
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
                  SQL_VARCHAR, 0, 0, long_ptr, 0, NULL);
```

表 95 显示了 C 与 SQL 数据类型之间支持的数据转换。表 95 中的第一列包含 SQL 数据类型。剩余各列表示 C 数据类型。如果 C 数据类型列包含:

D 该转换受支持，且是 SQL 数据类型的缺省转换。

X DB2 Everyplace 支持该转换。

空白 DB2 Everyplace 不支持该转换。

对精度和小数位的限制以及用于类型转换的截断和舍入规则，遵循 SQL 语法规则。

表 95. 支持的数据转换

SQL 数据类型	缺省转换	其它受支持转换
BLOB	SQL C BINARY	SQL C CHAR
CHAR	SQL C CHAR	SQL C LONG SQL C SHORT SQL C TINYINT SQL C TYPE DATE SQL C TYPE TIME SQL C BINARY SQL C BIT SQL C TYPE TIMESTAMP
DATE	SQL C TYPE DATE	SQL C CHAR

表 95. 支持的数据转换 (续)

SQL 数据类型	缺省转换	其它受支持转换
DECIMAL	SQL_C_CHAR	SQL_C_LONG SQL_C_SHORT SQL_C_TINYINT SQL_C_BIT
INTEGER	SQL_C_LONG	SQL_C_CHAR SQL_C_SHORT SQL_C_TINYINT SQL_C_FLOAT SQL_C_DOUBLE SQL_C_BIT
SMALLINT	SQL_C_SHORT	SQL_C_CHAR SQL_C_LONG SQL_C_TINYINT SQL_C_FLOAT SQL_C_DOUBLE SQL_C_BIT
TIME	SQL_C_TYPE_TIME	SQL_C_CHAR
TIMESTAMP	SQL_C_TYPE_TIMESTAMP	SQL_C_CHAR
VARCHAR	SQL_C_CHAR	SQL_C_LONG SQL_C_SHORT SQL_C_TINYINT SQL_C_TYPE_DATE SQL_C_TYPE_TIME SQL_C_BINARY SQL_C_BIT SQL_C_TYPE_TIMESTAMP

相关参考:

- 第 177 页的『DB2 CLI 函数描述的关键』
- 第 175 页的『DB2 CLI 函数摘要』

支持的 JDBC 方法

本章包含有关 DB2 Everyplace 支持的 JDBC 方法的信息。本章包含下列各节:

- 『DB2 Everyplace JDBC 支持的概述』
- 第 254 页的『java.sql 包中的接口』
- 第 269 页的『javax.sql 包中的接口』

DB2 Everyplace JDBC 支持的概述

DB2 Everyplace 支持 Sun Java Developer's Kit 中提供的 "Java 数据库连接"（JDBC）API 规范中定义的方法的一个子集。有关 DB2 Everyplace 支持的 JDBC 方法的信息在 Sun 的 Java Development Kit V1.4.1 文档中作了修改。DB2 Everyplace 还支持扩展 "连接" 和 "语句" 接口。

有关更多信息，请参阅第 268 页的『DB2eStatement 类』和第 257 页的『DB2eConnection 类』。

DB2 Everyplace JDBC 驱动程序与 JSR 169 指定的 JDBC Optional Package for CDC/Foundation Profile 兼容。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 『Blob 接口』
- 第 256 页的『连接接口』
- 第 257 页的『DB2eConnection 类』
- 第 258 页的『DatabaseMetaData 接口』
- 第 260 页的『驱动程序接口』
- 第 261 页的『PreparedStatement 接口』
- 第 262 页的『ResultSet 接口』
- 第 266 页的『ResultSetMetaData 接口』
- 第 267 页的『语句接口』
- 第 268 页的『DB2eStatement 类』
- 第 174 页的『JDBC 报告的 SQLState 消息』

java.sql 包中的接口

本章提供有关 java.sql 包中的 JDBC 方法的信息。涉及的主题包括:

- 『Blob 接口』
- 第 255 页的『CallableStatement 接口』
- 第 256 页的『连接接口』
- 第 257 页的『DB2eConnection 类』
- 第 258 页的『DatabaseMetaData 接口』
- 第 260 页的『驱动程序接口』
- 第 261 页的『PreparedStatement 接口』
- 第 262 页的『ResultSet 接口』
- 第 266 页的『ResultSetMetaData 接口』
- 第 267 页的『语句接口』
- 第 268 页的『DB2eStatement 类』

Blob 接口

Blob 接口以 Java™ 编程语言表示（映射）SQL BLOB。SQL BLOB 是将二进制大对象存储库为数据库表行中的列值的内置类型。BLOB 对象在创建它的事务的持续时间内有效。

接口 ResultSet 和 PreparedStatement 中的方法（如 getBlob 和 setBlob）允许程序员访问 SQL BLOB。Blob 接口提供了一些方法来获取 SQL BLOB（二进制大对象）值的长度和在客户机上实现 BLOB 值。

java.sql 包

公共接口 **Blob**

表 96 列示 Blob 接口中 DB2 Everyplace 支持的方法。

表 96. Blob 接口方法

方法返回值类型	方法
InputStream	**getBinaryStream**() 检索由此 BLOB 实例指定为流的 BLOB。
byte[]	**getBytes**(long pos, int length) 返回作为字节部分的数组或此 BLOB 对象指定的所有 BLOB 值。
long	**length**() 返回由此 BLOB 对象指定的 BLOB 值中的字节数。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

CallableStatement 接口

该接口用于执行远程 SQL 存储过程。结果参数必须注册为 OUT 参数。其它参数可用于输入和 / 或输出。可按编号顺序引用这些参数。第一个参数的编号为 1。

有关更多详细信息，请参阅《*DB2 Everyplace Sync Server 管理指南*》中称为『远程查询和存储过程适配器』的一节。

```
call <procedure-name> (?,?, ...)
```

IN 参数值是使用继承自 PreparedStatement 的 set 方法设置的。必须在执行存储过程之前注册该类型的所有 OUT 参数；将在执行后通过此处提供的 get 方法检索它们的值。输出参数的大小限制为 4K 字节。

CallableStatement 可返回一个 ResultSet。

java.sql 包

公共接口 **CallableStatement**

扩展 PreparedStatement

表 97 列示 CallableStatement 接口中 DB2 Everyplace 支持的方法。

表 97. CallableStatement 接口方法

方法返回值类型	方法
Blob	**getBlob**(int i) JDBC 2.0 在 Java 编程语言中以 Blob 对象的形式获取 JDBC BLOB 参数的值。
byte[]	**getBytes**(int parameterIndex) 在 Java 编程语言中以一组 byte 值的形式获取 JDBC BINARY 或 VARBINARY 参数的值。

表 97. CallableStatement 接口方法 (续)

方法返回值类型	方法
Date	**getDate**(int parameterIndex) 以 java.sql.Date 对象的形式获取 JDBC DATE 参数的值。
int	**getInt**(int parameterIndex) 在 Java 编程语言中以 int 的形式获取 JDBC INTEGER 参数的值。
Object	**getObject**(int parameterIndex) 在 Java 编程语言中以 object 的形式获取参数的值。
short	**getShort**(int parameterIndex) 在 Java 编程语言中以 short 的形式获取 JDBC SMALLINT 参数的值。
String	**getString**(int parameterIndex) 在 Java 编程语言中以 String 的形式检索 JDBC CHAR、VARCHAR 或 LONGVARCHAR 参数的值。
Time	**getTime**(int parameterIndex) 以 java.sql.Time 对象的形式获取 JDBC TIME 参数的值。
Timestamp	**getTimestamp**(int parameterIndex) 以 java.sql.Timestamp 对象的形式获取 JDBC TIMESTAMP 参数的值。
void	**registerOutParameter**(int parameterIndex, int sqlType) 以顺序位置 parameterIndex 将 OUT 参数注册为 JDBC 类型 sqlType。
boolean	**wasNull**() 指示读取的上一个 OUT 参数是否具有值 SQL NULL。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

连接接口

“连接”接口建立与特定数据库的连接（会话）。在“连接”的上下文中，会执行 SQL 语句并返回结果。

“连接”的数据库能够提供描述下列各项的信息: 它的表、支持的 SQL 语法、它的存储过程、此连接的能力，等等。此信息是通过 getMetaData 方法获得的。

java.sql 包

公共接口**连接**

表 98 列示“连接”接口中 DB2 Everyplace 支持的方法。

表 98. “连接”接口方法

方法返回值类型	方法
void	**clearWarnings**() 清除对此“连接”对象报告的所有警告。
void	**close**() 立即释放“连接”的数据库和 JDBC 资源，而不是等待自动释放它们。
void	**commit**() 使自从上次落实或回滚之后所作的所有更改具有持久状态，并释放当前由 Connection 挂起的任何数据库锁。
Statement	**createStatement**() 创建一个语句对象以将 SQL 语句发送至数据库。

表 98. “连接”接口方法 (续)

方法返回值类型	方法
Statement	**createStatement**(int resultSetType, int resultSetConcurrency) JDBC 2.0。创建将生成具有给定的类型和并行性的 ResultSet 对象的“语句”对象。
boolean	**isClosed**() 测试“连接”是否关闭。
DatabaseMetaData	**getMetaData**() 获取与此“连接”的数据库有关的元数据。
SQLWarning	**getWarnings**() 返回此“连接”上的调用所报告的首次警告。
CallableStatement	**prepareCall**(String sql) 创建 CallableStatement 对象以调用数据库存储过程。
PreparedStatement	**prepareStatement** (String sql) 创建一个 PreparedStatement 对象，以将参数化的 SQL 语句发送至数据库。
PreparedStatement	**prepareStatement**(String sql, int resultSetType, int resultSetConcurrency) JDBC 2.0。创建将生成具有给定的类型和并行性的 ResultSet 对象的 PreparedStatement 对象。
void	**rollback**() 删除从前一次落实或回滚之后所作的所有更改，并释放当前由此 Connection 挂起的任何数据库锁。
void	**setAutoCommit** (boolean autoCommit) 设置此连接的自动落实方式。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

DB2eConnection 类

DB2eConnection 类获取和设置特定“连接”属性。要对“连接”对象使用 DB2eConnection 类方法，“连接”对象首先必须强制转型为 DB2eConnection 对象。这些方法是通过使用适当自变量调用 CLI/ODBC 函数 SQLGetConnectAttr 和 SQLSetConnectAttr 实现的。

有关更多信息，请参阅第 10 章: 『受支持的 DB2 CLI/ODBC 函数』。

com.ibm.db2e.jdbc 包

公用类 **DB2eConnection**

实现 Connection

表 99 列示 DB2eConnection 类中 DB2 Everyplace 支持的方法。

表 99. DB2eConnection 类方法

方法返回类型	方法
void	**enableFilenameFormat83** (boolean enable) 如果启用为 true 的话，则启用数据库引擎来以 8.3 格式创建文件名，否则将以长格式启用文件名。仅当此连接的路径中不存在任何目录文件时，才能更改文件名格式。

表 99. DB2eConnection 类方法 (续)

方法返回类型	方法
boolean	**isEnabledFilenameFormat83 ()** 数据库引擎会以 8.3 格式创建文件名吗？或者它会以长格式创建文件名吗？

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』
- 第 219 页的『SQLGetConnectAttr – 获取连接属性的当前设置』
- 第 241 页的『SQLSetConnectAttr – 设置与连接相关的选项』

DatabaseMetaData 接口

DatabaseMetaData 接口作为整体提供有关数据库的综合信息。

其中某些方法采用“字符串”自变量作为目录和模式名称。DB2 Everyplace 忽略这些自变量。

此处的某些方法以 ResultSet 对象的格式返回信息列表。可以使用正常 ResultSet 方法（如 getString 和 getInt）来从这些 ResultSet 中检索数据。

如果元数据的给定形式不可用，则这些方法抛出 SQLException。

java.sql 包

公共接口 **DatabaseMetaData**

表 100 列示 DatabaseMetaData 接口中 DB2 Everyplace 支持的字段。

表 100. DatabaseMetaData 字段

字段类型	字段
static int	**columnNoNulls** 指示该列可能不允许 NULL 值。
static int	**columnNullable** 指示该列明确允许 NULL 值。
static int	**columnNullableUnknown** 指示列的可空性未知。

表 101 列示 DatabaseMetaData 接口中 DB2 Everyplace 支持的方法。

表 101. DatabaseMetaData 接口方法

方法返回值类型	方法
ResultSet	**getColumns**(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) 获取指定目录中可用的表列的描述。
Connection	**getConnection**() JDBC 2.0 检索生成此元数据对象的连接。

表 101. DatabaseMetaData 接口方法 (续)

方法返回值类型	方法
ResultSet	**getCrossReference**(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable) 获取外键表中引用主键表的主键列的外键列的描述（描述一个表如何导入另一个键。）正常情况下应返回单个外键／主键对（大多数表仅从表导入某个外键一次。）按 FKTABLE_NAME 和 KEY_SEQ 排序它们。
String	**getDatabaseProductName**() 此数据库产品的名称是什么？
String	**getDatabaseProductVersion**() 此数据库产品的版本是什么？
int	**getDriverMajorVersion**() 此 JDBC 驱动程序的主版本号是什么？
int	**getDriverMinorVersion**()此 JDBC 驱动程序的次版本号是什么？
String	**getDriverName**() 此 JDBC 驱动程序的名称是什么？
String	**getDriverVersion**() 此 JDBC 驱动程序的版本是什么？
ResultSet	**getExportedKeys**(String catalog, String schema, String table) 获取引用表的主键列的外键列的描述（由表导出外键）。
String	**getIdentifierQuoteString**() 什么是用来将 SQL 标识引起的字符串？如果不支持将标识加引号，则返回空格 " "。
ResultSet	**getImportedKeys**(String catalog, String schema, String table) 获取由表的外键列引用的主键列的描述（由表导入的主键）。
int	**getMaxBinaryLiteralLength**() 直接插入的二进制文字内可以具有多少个十六进制字符？
int	**getMaxCharLiteralLength**() 字符文字的最大长度是多少？
int	**getMaxColumnNameLength**() 列名长度的限制是多少？
int	**getMaxColumnsInGroupBy**() GROUP BY 子句中的最大列数是多少？
int	**getMaxColumnsInIndex**() 索引内允许的最大列数是多少？
int	**getMaxColumnsInOrderBy**() ORDER BY 子句中的最大列数是多少？
int	**getMaxColumnsInSelect**() SELECT 语句中的最大列数是多少？
int	**getMaxConnections**() 每次可以有多少个与此数据库的活动连接？
int	**getMaxIndexLength**() 索引的最大长度是多少（以字节计）？
int	**getMaxRowSize**() 单行的最大长度是多少？
int	**getMaxStatementLength**() SQL 语句的最大长度是多少？
int	**getMaxStatements**() 每次可以对此数据库打开多少个活动的语句？
int	**getMaxTableNameLength**() 表名的最大长度是多少？
int	**getMaxTablesInSelect**() SELECT 语句中最大表数是多少？
int	**getMaxUserNameLength**() 用户名的最大长度是多少？
ResultSet	**getPrimaryKeys**(String catalog, String schema, String table) 获取表的主键列的描述。
String	**getSearchStringEscape**() 获取可用于将通配符转义的字符串。
ResultSet	**getTables**(String catalog, String schemaPattern, String tableNamePattern, String[] types) 获取目录中可用的表的描述。

表 101. DatabaseMetaData 接口方法 (续)

方法返回值类型	方法
ResultSet	**getUDTs**(String catalog, String schemaPattern, String typeNamePattern, int[] types) JDBC 2.0 获取在特定模式中定义的用户定义类型的描述。DB2 Everyplace 总是返回空的结果集，原因是它不支持 UDT。
String	**getURL**() 此数据库的 URL 是什么？
String	**getUserName**() 数据库所知的用户名是什么？
boolean	**supportsColumnAliasing**() 支持列别名判别吗？
boolean	**supportsFullOuterJoins**() 支持全嵌套外连接吗？
boolean	**supportsMixedCaseIdentifiers**() 数据库要将混合大小写且未加引号的 SQL 标识视作区分大小写并从而以混合大小写方式存储它们吗？
boolean	**supportsMixedCaseQuotedIdentifiers**() 数据库要将混合大小写且加了引号的 SQL 标识视作区分大小写并从而以混合大小写方式存储它们吗？
boolean	**supportsNonNullableColumns**() 列可以定义为不可空吗？
boolean	**supportsOrderByUnrelated**() “ORDER BY”子句可以使用不在 SELECT 语句中的列吗？
boolean	**supportsOuterJoins**() 支持某种形式的外连接吗？
boolean	**supportsPositionedDelete**() 支持定位 DELETE 吗？
boolean	**supportsPositionedUpdate**() 支持定位 UPDATE 吗？
boolean	**supportsResultSetType**(int type) JDBC 2.0 数据库支持给定结果集类型吗？
boolean	**supportsSchemasInTableDefinitions**() 可以在表定义语句中使用模式名吗？
boolean	**supportsTransactions**() 事务受支持吗？如果不受支持，则隔离级别是 TRANSACTION_NONE。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

驱动程序接口

“驱动程序”接口是允许有多个数据库驱动程序的 Java SQL 框架。

在装入“驱动程序”类时，该类应创建它自身的实例，并向 DriverManager 注册。这意味着用户可通过调用以下程序来装入和注册 DB2 Everyplace JDBC 驱动程序:

```
Class.forName("com.ibm.db2e.jdbc.DB2eDriver")
```

java.sql 包

公共接口**驱动程序**

表 102 列示“驱动程序”接口中 DB2 Everyplace 支持的方法。

表 102. 驱动程序接口方法

方法返回值类型	方法
boolean	**acceptsURL**(String url) 如果驱动程序认为它能打开与给定 URL 的连接，则返回 true。
Connection	**connect**(String url, Properties info) 尝试建立与给定 URL 的数据库连接。java.util.Properties 自变量可以用于传送任意字符串标记／值对作为连接自变量。DB2 Everyplace 支持下列特定于驱动程序的键和值对: • 键: LOGIN_TIMEOUT 值: 秒数 • 键: DB2e_ENCODING 值: 字符编码
int	**getMajorVersion**() 获取驱动程序的主版本号。
int	**getMinorVersion**() 获取驱动程序的次版本号。
boolean	**jdbcCompliant**() 报告此驱动程序是否为真正的 JDBC COMPLIANT™ 驱动程序。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

PreparedStatement 接口

PreparedStatement 接口创建表示预编译的 SQL 语句的对象。

SQL 语句经过预编译，并存储在 PreparedStatement 对象中。然后，此对象可用来有效地多次执行此语句。

java.sql 包

公共接口 **PreparedStatement**

扩展“语句”

表 103 列示 PreparedStatement 接口中 DB2 Everyplace 支持的方法。

表 103. PreparedStatement 接口方法

方法返回值类型	方法
void	**clearParameters**() 立即清除当前参数值。
boolean	**execute**() 执行任何种类的 SQL 语句。
ResultSet	**executeQuery**() 在此 PreparedStatement 对象中执行 SQL 查询，并返回由该查询生成的结果集。
int	**executeUpdate**() 在此 PreparedStatement 对象中执行 SQL INSERT、UPDATE 或 DELETE 语句。
void	**setBigDecimal** (int parameterIndex, BigDecimal x) 将指定的参数设置为 java.lang.BigDecimal 值。此方法在 Palm OS 的 DB2 Everyplace JDBC 驱动程序中不可用。

表 103. PreparedStatement 接口方法 (续)

方法返回值类型	方法
void	**setBoolean** (int parameterIndex, boolean x) 将指定的参数设置为 Java boolean 值。当它发送至数据库时，DB2 Everyplace JDBC 驱动程序将它转换为 SQL SMALLINT 值。
void	**setBlob**(int i, Blob x) JDBC 2.0 设置 BLOB 参数。
void	**setBytes**(int parameterIndex, byte[ ]x) 将指定的参数设置为字节的 Java 数组。
void	**setDate**(int parameterIndex, Date x) 将指定参数设置为 java.sql.Date 值。
void	**setDouble**(int parameterIndex, double x) 将指定的参数设置为 Java 双精度值。当它发送至数据库时，DB2 Everyplace JDBC 驱动程序将它转换为 SQL DECIMAL 值。
void	**setFloat**(int parameterIndex, float x) 将指定的参数设置为 Java 浮点值。 当 BigDecimal 转换为浮点型时，如果 BigDecimal 太大而无法表示为浮点型，它将根据需要转换为 FLOAT.NEGATIVE_INFINITY 或 FLOAT.POSITIVE_INFINITY。
void	**setInt** (int parameterIndex, int x) 将指定的参数设置为 Java int 值。
void	**setLong**(int parameterIndex, long x) 将指定的参数指定为 Java 长整型值。
void	**setNull** (int parameterIndex, int sqlType) 将指定的参数设置为 SQL NULL。
void	**setObject**(int parameterIndex, Object x, int targetSqlType) 使用给定对象来设置指定参数的值。 **DB2 Everyplace 限制**: • targetSqlType 必须与 DB2 Everyplace 支持的数据类型之一相对应。 • 支持基本转换和字符串转换。例如，如果 targetSqlType 为 Types.INTEGER，则 x 应为 Integer 或 String 对象。 • 如果 targetSqlType 为 Types.DECIMAL，则 x 还可以是 Double、Float 或 Long 对象。 • 如果 targetSqlType 为 Types.SMALLINT，则 x 还可以是 Boolean 对象。 • 在 Palm OS 上，如果 targetSqlType 是 Types.DECIMAL，则 x 应为 String 对象。
void	**setShort** (int parameterIndex, short x) 将指定的参数设置为 Java short 值。
void	**setString** (int parameterIndex, String x) 将指定参数设置为 Java String值。
void	**setTime** (int parameterIndex, Time x) 将指定参数设置为 java.sql.Time 值。
void	**setTimestamp** (int parameterIndex, Timestamp x) 将指定参数设置为 java.sql.Timestamp 值。

相关任务:

• 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

• 第 253 页的『DB2 Everyplace JDBC 支持的概述』
• 第 174 页的『JDBC 报告的 SQLState 消息』

ResultSet 接口

ResultSet 接口提供对数据表的访问。ResultSet 对象通常是通过执行“语句”来生成的。

ResultSet 始终有一个游标指向其当前数据行。最初，游标定位在第一行的前面。next() 方法将游标移至下一行。

getXXX 方法会检索当前行的列值。可使用列的索引号或列的名称来检索这些值。通常，使用列索引将更为有效。列是从 1 开始编号的。

java.sql 包

公共接口 **ResultSet**

表 104 列示 ResultSet 接口中 DB2 Everyplace 支持的字段。

表 104. ResultSet 接口字段

字段类型	字段
static int	**CONCUR_READ_ONLY** 该常量指示不能更新的 ResultSet 对象的并行性方式。**注意**: DB2 Everyplace 不支持 CONCUR_UPDATABLE。如果在创建“语句”对象时对 ResultSet 对象的并行性方式指定 CONCUR_UPDATABLE，则 DB2 Everyplace JDBC 驱动程序将对产生“语句”对象的“连接”对象发出 SQLWarning 并使用 CONCUR_READ_ONLY 代替。
static int	**TYPE_FORWARD_ONLY** 该常量指示其游标只能向前移动的 ResultSet 对象的类型。
static int	**TYPE_SCROLL_INSENSITIVE** 该常量指示可滚动但通常对他人所作的更改不敏感的 ResultSet 对象的类型。**注意**: 不要经常使用此类型的 ResultSet 对象，原因是它可能会影响性能。此类型使用 SQL_INSENSITIVE 作为 CLI 语句属性 SQL_ATTR_CURSOR_SENSITIVITY 的值。有关详细信息，参阅 CLI 函数 SQLSetStmtAttr 的文档。
static int	**TYPE_SCROLL_SENSITIVE** 该常量指示可滚动且通常对他人所作的更改敏感的 ResultSet 对象的类型。**注意**: 此类型使用 SQL_UNSPECIFIED 作为 CLI 语句属性 SQL_ATTR_CURSOR_SENSITIVITY 的值。有关详细信息，参阅 CLI 函数 SQLSetStmtAttr 的文档。

表 105 列示 ResultSet 接口中 DB2 Everyplace 支持的方法。

表 105. ResultSet 接口方法

方法返回值类型	方法
boolean	**absolute**(int row) JDBC 2.0。将游标移至结果集中的给定行号。
void	**afterLast**() JDBC 2.0。将游标移至结果集的末尾，正好在最后一行的后面。
void	**beforeFirst**() JDBC 2.0。将游标移至结果集的前方，正好在第一行的前面。
void	**clearWarnings**() 清除此 ResultSet 对象上报告的所有警告。
void	**close**() 立即释放此 ResultSet 对象的数据库和 JDBC 资源，而不是等待对象自动关闭时才释放它们。
int	**findColumn**(String columnName) 将给定 ResultSet 列名映射至其 ResultSet 列索引。
boolean	**first**() JDBC 2.0。将游标移至结果集中的第一行。
BigDecimal	**getBigDecimal**(int columnIndex) JDBC 2.0。以具有全部精度的 java.math.BigDecimal 对象形式获取当前行中某个列的值。Palm OS 的 DB2 Everyplace JDBC 驱动程序不支持此方法。

表 105. ResultSet 接口方法 (续)

方法返回值类型	方法
BigDecimal	**getBigDecimal**(int columnIndex, int scale) 以 Java 编程语言中的 java.math.BigDecimal 对象形式获取此 ResultSet 对象当前行中指定列的值。Palm OS 的 DB2 Everyplace JDBC 驱动程序不支持此方法。**不受支持。**
BigDecimal	**getBigDecimal**(String columnName) JDBC 2.0。以具有全部精度的 java.math.BigDecimal 对象形式获取当前行中某个列的值。Palm OS 的 DB2 Everyplace JDBC 驱动程序不支持此方法。
BigDecimal	**getBigDecimal**(String columnName, int scale) 以 Java 编程语言中的 java.math.BigDecimal 对象形式获取此 ResultSet 对象当前行中指定列的值。Palm OS 的 DB2 Everyplace JDBC 驱动程序不支持此方法。**不受支持。**
Blob	**getBlob**(int columnIndex) JDBC 2.0。获取此 ResultSet 对象的当前行中的 BLOB 值。
Blob	**getBlob**(String columnName) JDBC 2.0。获取此 ResultSet 对象的当前行中的 BLOB 值。
boolean	**getBoolean**(int columnIndex) 以 Java 布尔值形式获取当前行中某列的值。
boolean	**getBoolean**(String columnName) 以 Java 布尔值形式获取当前行中某列的值。
byte	**getByte**(int columnIndex) 以 Java 编程语言中的字节形式获取此 ResultSet 对象当前行中指定列的值。
byte	**getByte**(String columnName) 以 Java 编程语言中的字节形式获取此 ResultSet 对象当前行中指定列的值。
byte[]	**getBytes**(int columnIndex) 以 Java 编程语言中的字节数组形式获取此 ResultSet 对象当前行中指定列的值。
byte[]	**getBytes**(String columnName) 以 Java 编程语言中的字节数组形式获取此 ResultSet 对象当前行中指定列的值。
int	**getConcurrency**() JDBC 2.0。返回结果集的并行性方式。
Date	**getDate**(int columnIndex) 以 Java 编程语言中的 java.sql.Date 对象形式获取此 ResultSet 对象当前行中指定列的值。
Date	**getDate**(int columnIndex, Calendar cal) 以 Java 编程语言中的 java.sql.Date 对象形式返回此 ResultSet 对象的当前行中指定列的值。
Date	**getDate**(String columnName) 以 Java 编程语言中的 java.sql.Date 对象形式获取此 ResultSet 对象的当前行中指定列的值。
double	**getDouble**(int columnIndex) 以 Java 双精度形式获取当前行中某列的值。
double	**getDouble**(String columnName) 以 Java 双精度形式获取当前行中某列的值。
float	**getFloat**(int columnIndex) 以 Java 浮点形式获取当前行中某列的值。
float	**getFloat**(String columnName) 以 Java 浮点形式获取当前行中某列的值。
int	**getInt**(int columnIndex) 以 Java 编程语言中的整数形式获取此 ResultSet 对象当前行中指定列的值。

表 105. ResultSet 接口方法 (续)

方法返回值类型	方法
int	**getInt**(String columnName) 以 Java 编程语言中的整数形式获取此 ResultSet 对象的当前行中指定列的值。
long	**getLong**(int columnIndex) 以 Java 长整型形式获取当前行中某列的值。
long	**getLong**(String columnName) 以 Java 长整型形式获取当前行中某列的值。
ResultSetMetaData	**getMetaData**() 检索此 ResultSet 对象的列的数目、类型和属性。
Object	**getObject**(int columnIndex) 以 Java 对象形式获取当前行中某列的值。
Object	**getObject**(String columnName) 以 Java 对象形式获取当前行中某列的值。
int	**getRow**() JDBC 2.0。检索当前行号。
short	**getShort**(int columnIndex) 以 Java 编程语言中的 short 形式获取此 ResultSet 对象当前行中指定列的值。
short	**getShort**(String columnName) 以 Java 编程语言中的 short 形式获取此 ResultSet 对象当前行中指定列的值。
Statement	**getStatement**() JDBC 2.0。返回产生此 ResultSet 对象的 “语句”。
String	**getString**(int columnIndex) 以 Java 编程语言中的 String 形式获取此 ResultSet 对象当前行中指定列的值。
String	**getString**(String columnName) 以 Java 编程语言中的 String 形式获取此 ResultSet 对象当前行中指定列的值。
Time	**getTime**(int columnIndex) 以 Java 编程语言中的 java.sql.Time 对象形式获取此 ResultSet 对象的当前行中指定列的值。
Time	**getTime**(String columnName) 以 Java 编程语言中的 java.sql.Date 对象形式获取此 ResultSet 对象的当前行中指定列的值。
Timestamp	**getTimestamp**(String columnName) 以 Java 编程语言中的 java.sql.Timestamp 对象形式获取此 ResultSet 对象的当前行中指定列的值。
Timestamp	**getTimestamp**(int columnIndex) 以 Java 编程语言中的 java.sql.Timestamp 对象形式获取此 ResultSet 对象的当前行中指定列的值。
int	**getType**() JDBC 2.0。返回此结果集的类型。
SQLWarning	**getWarnings**() 返回此 ResultSet 上的调用报告的首次警告。
boolean	**isAfterLast**() JDBC 2.0。指示游标是否在结果集中的最后一行后面。
boolean	**isBeforeFirst**() JDBC 2.0。指示游标是否在结果集中的第一行前面。
boolean	**isFirst**() JDBC 2.0。指示游标是否在结果集中的第一行上。
boolean	**isLast**() JDBC 2.0。指示游标是否在结果集中的最后一行上。对于具有类型 TYPE_FORWARD_ONLY 的结果集，不支持此方法。
boolean	**last**() JDBC 2.0。将游标移至结果集中的最后一行。
boolean	**next**() 将游标从当前位置向下移动一行。
boolean	**previous**() JDBC 2.0。将游标移至结果集中的前一行。
boolean	**relative**(int rows) JDBC 2.0。将游标移动相对行数，正数或负数。
boolean	**wasNull**() 报告读取的最后一列是否具有值 SQL NULL。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』
- 第 244 页的『SQLSetStmtAttr – 设置与语句相关的选项』

ResultSetMetaData 接口

ResultSetMetaData 接口创建一个对象，可使用该对象找出 ResultSet 中的各列的类型和属性。

java.sql 包

公共接口 **ResultSetMetaData**

表 106 列示 ResultSetMetaData 接口中 DB2 Everyplace 支持的字段。

表 106. ResultSetMetaData 接口字段

字段类型	字段
`static int`	**columnNoNulls** 一个常量，指示某列不允许 NULL 值。
`static int`	**columnNullable** 一个常量，指示某列允许 NULL 值。
`static int`	**columnNullableUnknown** 一个常量，指示某列的值的可空性未知。

表 107 列示 ResultSetMetaData 接口中 DB2 Everyplace 支持的方法。

表 107. ResultSetMetaData 接口方法

方法返回值类型	方法
`String`	**getCatalogName**(int column) 获取列的表的目录名。DB2 Everyplace 总是返回 ""（不适用）。
`int`	**getColumnCount**() 返回此 ResultSet 对象中的列的数目。
`int`	**getColumnDisplaySize** (int column) 指示指定列的正常最大宽度（以字符计）。
`String`	**getColumnLabel**(int column) 获取在打印输出和显示中使用的建议列标题。
`String`	**getColumnName** (int column) 获取指定列的名称。
`int`	**getColumnType** (int column) 获取指定列的 SQL 类型。
`String`	**getColumnTypeName**(int column) 检索列的特定于数据库的类型名。
`int`	**getPrecision** (int column) 获取指定列的小数位数。
`int`	**getScale** (int column) 获取指定列的小数点右边的位数。
`String`	**getSchemaName**(int column) 获取列的表的模式名。DB2 Everyplace 总是返回 ""（不适用）。
`int`	**isNullable** (int column) 指示指定列中的值的可空性。
`boolean`	**isWritable**(int column) 指示对列的写操作能否成功。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

语句接口

“语句”接口创建用于执行静态 SQL 语句的对象并获取通过该对象产生的结果。

java.sql 包

公共接口**语句**

表 108 列示“语句”接口中 DB2 Everyplace 支持的方法。

表 108. 语句接口方法

方法返回值类型	方法
void	**addBatch**(String sql) JDBC 2.0 将 SQL 命令添加至该语句的当前一批命令。
void	**clearBatch**() JDBC 2.0 清空当前批中的一组命令。
void	**close**() 立即释放此语句对象的数据库和 JDBC 资源，而不是等待该对象自动关闭时才释放它们。
boolean	**execute**(String sql) 执行可能返回多个结果的 SQL 语句。
int[]	**executeBatch**() JDBC 2.0 将一批命令提交至数据库以便执行。
ResultSet	**executeQuery**(String sql) 执行返回单个 ResultSet 对象的 SQL 语句。
int	**executeUpdate**(String sql) 执行 SQL INSERT、UPDATE 或 DELETE 语句。
Connection	**getConnection** () JDBC 2.0。返回产生此“语句”对象的“连接”对象。
boolean	**getMoreResults**() 移至语句的下一个结果。DB2 Everyplace 始终返回 false（没有其它结果）。
ResultSet	**getResultSet**() 以 ResultSet 对象形式返回当前结果。
int	**getResultSetConcurrency**() JDBC 2.0。检索结果集并行性。
int	**getResultSetType**() JDBC 2.0。确定结果集类型。
int	**getUpdateCount**() 返回当前结果作为更新计数；如果结果为 ResultSet 或没有其它结果，则返回 -1。

表 109 列示“语句”接口中 DB2 Everyplace 支持的字段。

表 109. 语句接口字段

字段类型	字段
static int	**SUCCESS_NO_INFO** 该常量指示批处理语句执行成功；但未提供它影响的行的计数。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

DB2eStatement 类

DB2eStatement 类获取和设置特定“语句”属性。要对“语句”对象使用 DB2eStatement 类方法，“语句”对象首先必须强制转型为 DB2eStatement 对象。这些方法是通过使用适当自变量调用 CLI/ODBC 函数 SQLGetStmtAttr 和 SQLSetStmtAttr 实现的。

有关更多信息，请参阅第 175 页的『DB2 CLI 函数摘要』。

com.ibm.db2e.jdbc 包

公用类 **DB2eStatement**

实现“语句”

表 110 列示 DB2eStatement 类中 DB2 Everyplace 支持的方法。

表 110. DB2eStatement 类方法

方法返回类型	方法
`void`	**enableDeletePhysicalRemove**(boolean `enable`) 在 DELETE SQL 语句中，启用或禁用在物理上除去记录，而不考虑它们的脏位值。
`void`	**enableDirtyBitSetByApplication**(boolean `enable`) 如果 enable 为 true，则启用应用方式。否则，启用系统方式。
`void`	**enableReadIncludeMarkedDelete** (boolean `enable`) 使逻辑上删除的记录可视或不可视。
`void`	**enableReorg** (boolean `enable`) 由 DB2 Everyplace 启用或禁用数据库重组，或由用户使用 REORG SQL 语句显式启用或禁用数据库重组。
`boolean`	**isEnabledDeletePhysicalRemove**() 删除 SQL 语句将在物理上除去记录而不考虑它们的脏位值吗？或者仅将记录标记为“删除”？
`boolean`	**isEnabledDirtyBitSetByApplication**() 数据库系统处于应用方式吗？或者它处于系统方式？
`boolean`	**isEnabledReadIncludeMarkedDelete**() 可以从 SQL 语句看到逻辑上删除的记录吗？或者这些记录对于 SQL 是不可视的？
`boolean`	**isEnabledReorg**() 可以由 DB2 Everyplace 执行数据库重组或由用户使用 REORG SQL 语句显式执行数据库重组吗？或者 REORG SQL 语句受到限制，禁止对用户创建的表自动执行数据库重组？

在这些示例中，`st` 表示“语句”对象，`rs` 表示 ResultSet 对象。

要从表 T 物理上除去某些记录，而忽略其脏位的状态:

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableDeletePhysicalRemove(true);
st.executeUpdate("DELETE FROM T WHERE X<>0");
```

要读取表 T 中设置了脏位的所有记录，包括那些脏位被标记为 DELETE 的记录:

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableReadIncludeMarkedDelete(true);
rs = st.executeQuery("SELECT * FROM T WHERE $dirty<>0");
```

要清除表 T 中记录的脏位:

```
DB2eStatement db2e_st = (DB2eStatement) st;
db2e_st.enableDirtyBitSetByApplication(true);
st.executeUpdate("UPDATE T SET $dirty=0 WHERE $dirty>0");
```

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』
- 第 231 页的『SQLGetStmtAttr – 获取语句属性的当前设置』
- 第 244 页的『SQLSetStmtAttr – 设置与语句相关的选项』

javax.sql 包中的接口

本章提供有关 javax.sql 包中的 JDBC 方法的信息。涉及的主题包括:

- 『DataSource 接口』

DataSource 接口

与此 DataSource 对象表示的物理数据源的连接的工厂。DataSource 对象替换 DriverManager 设施成为获取连接的首选手段。

可在独立程序中使用 DataSource 对象的实例以创建“连接”对象。在以下示例中, DB2eDataSource 的实例用来使用 URL “jdbc:db2e:myDataSource” 创建与 DB2 Everyplace 数据库的“连接”:

```
com.ibm.db2e.jdbc.DB2eDataSource ds = new com.ibm.db2e.jdbc.DB2eDataSource();
    ds.setUrl("jdbc:db2e:myDataSource");
    Connection con = ds.getConnection();
```

javax.sql 包

公共接口 **DataSource**

表 111 和第 270 页的表 112 列示 DataSource 接口中 DB2 Everyplace 支持的属性。可使用 “getter” 和 “setter” 方法访问这些属性。(DataSource 属性遵循在 “JavaBeans 1.01 规范” 中对 JavaBeans™ 组件的属性指定的约定。

表 111. DB2 Everyplace 支持的标准 DataSource 属性

属性名	类型	描述
description	String	此数据源的描述
password	String	数据库密码
user	String	用户的帐户名

第 270 页的表 112 列示 DataSource 接口中特定于 DB2 Everyplace 的受支持属性。

表 112. DataSource 接口的特定于 DB2 Everyplace 的属性

属性名	类型	描述
encoding	String	字符编码
URL	String	数据源

表 113 列示 DataSource 接口中 DB2 Everyplace 支持的方法。

表 113. DataSource 接口方法

方法返回值类型	方法
Connection	**getConnection**() 尝试建立与此 DataSource 对象表示的数据源的连接。
Connection	**getConnection** (java.lang.String username, java.lang.String password) 尝试建立与此 DataSource 对象表示的数据源的连接。
int	**getLoginTimeout**() 获取此数据源在尝试连接至数据库时可能等待的最长时间（以秒计）。
java.io.PrintWriter	**getLogWriter**() 检索此 DataSource 对象的日志写程序。
void	**setLoginTimeout**(int seconds) 设置此数据源在尝试连接至数据库时将等待的最长时间（以秒计）。
void	**setLogWriter**(java.io.PrintWriter out) 将此 DataSource 对象的日志写程序设置为给定 java.io.PrintWriter 对象。

相关任务:

- 第 17 页的『开发 DB2 Everyplace Java 应用程序』

相关参考:

- 第 253 页的『DB2 Everyplace JDBC 支持的概述』
- 第 174 页的『JDBC 报告的 SQLState 消息』

支持的 .NET 类

本章包含有关 DB2 Everyplace 支持的 .NET 类的信息。本章包含下列各节:

- 『DB2eCommandBuilder 成员』
- 第 271 页的『DB2eCommand 成员』
- 第 272 页的『DB2eConnection 成员』
- 第 272 页的『DB2eDataAdapter 成员』
- 第 273 页的『DB2eDataReader 成员』
- 第 274 页的『DB2eError 成员』
- 第 275 页的『DB2eException 成员』
- 第 275 页的『DB2eParameter 成员』
- 第 276 页的『DB2eTransaction 成员』
- 第 276 页的『DB2eType 枚举』

DB2eCommandBuilder 成员

表 114. 公用静态（共享）方法

方法	描述
DeriveParameters	从在 *DB2eCommand* 中指定的存储过程检索参数信息并填充指定 *DB2eCommand* 对象的“参数”集合。

表 115. 公用实例构造函数

构造函数	描述
DB2eCommandBuilder()	已重载。初始化 *DB2eCommandBuilder* 类的新实例。
DB2eCommandBuilder(DB2eDataAdapter)	已重载。使用相关联的 *DB2eDataAdapter* 对象初始化 *DB2eCommandBuilder* 类的新实例。

表 116. 公用实例属性

属性	描述
DataAdapter	获取或设置此 *DB2eCommandBuilder* 对象将对其生成 SQL 语句的 *DB2eDataAdapter* 对象。

表 117. 公用实例方法

方法	描述
GetDeleteCommand	获取在数据库上执行删除所需且自动生成的 *DB2eCommand* 对象。
GetInsertCommand	获取在数据库上执行插入所需且自动生成的 *DB2eCommand* 对象。
GetUpdateCommand	获取在数据库上执行更新所需且自动生成的 *DB2eCommand* 对象。
RefreshSchema	刷新用于生成 INSERT、UPDATE 或 DELETE 语句的数据库模式信息。

表 118. 受保护的实例方法

方法	描述
Dispose	已重载。

DB2eCommand 成员

表 119. 公用实例构造函数

构造函数	描述
DB2eCommand()	已重载。初始化 *DB2eCommand* 类的新实例。
DB2eCommand(string)	已重载。使用查询的文本初始化 *DB2eCommand* 类的新实例。
DB2eCommand(string, DB2eConnection)	已重载。使用查询的文本和 *DB2eConnection* 对象初始化 *DB2eCommand* 类的新实例。
DB2eCommand(string, DB2eConnection, DB2eTransaction)	已重载。使用查询的文本、*DB2eConnection* 对象和 *DB2eTransaction* 对象初始化 *DB2eCommand* 类的新实例。

表 120. 公用实例属性

属性	描述
CommandText	获取或设置要对数据库执行的 SQL 语句或存储过程。
CommandType	获取或设置指示如何解释 *CommandText* 属性的值。
Connection	获取或设置由 *DB2eCommand* 的此实例使用的 *DB2eConnection*。
DesignTimeVisible	获取或设置指示命令对象是否应在定制接口控件中可视的值。
参数	获取 *DB2eParameterCollection*。
Transaction	获取或设置 *DB2eCommand* 在其中执行的 *DB2eTransaction*。
UpdatedRowSource	获取或设置指定 *Update* 方法应如何将命令结果应用于 *DataRow* 的值。

表 121. 公用实例方法

方法	描述
CreateParameter	创建 *DB2eParameter* 对象的新实例。
Dispose	已重载。清除。
EnableDeletePhysicalRemove	启用或禁用在物理上除去记录。
EnableDirtyBitSetByApplication	如果启用为 true，则启用应用方式。否则，启用系统方式。
EnableReadIncludeMarkedDelete	使逻辑上已删除的记录可视或不可视。
EnableReorg	启用或禁用由 DB2 Everyplace 执行或由用户使用 REORG SQL 语句显式执行的数据库重组。
ExecuteNonQuery	对*连接*执行 SQL 语句并返回受影响的行数。

表 121. 公用实例方法 (续)

方法	描述
ExecuteReader	已重载。将 *CommandText* 发送至“连接”并构建 *DB2eDataReader*。
ExecuteScalar	执行该查询，然后返回查询返回的结果集中的第一行的第一列。额外列或行将被忽略。
IsEnabledDeletePhysicalRemove	检查是否启用了物理除去。如果已启用，则返回 true；否则返回 false。
IsEnabledDirtyBitSetByApplication	检查数据库系统是否处于应用方式或系统方式。如果已启用，则返回 true；否则返回 false。
IsEnabledReadIncludeMarkedDelete	检查逻辑上已删除的记录对应用程序是否可视。如果逻辑上已删除的记录对应用程序可视，则返回 true；否则返回 false。
IsEnabledReorg	检查是否启用了数据库重组。如果已启用，则返回 true，否则返回 false。
Prepare	数据库上创建该命令的已准备（或已编译）版本。

DB2eConnection 成员

表 122. 公用静态（共享）方法

方法	描述
ReleaseObjectPool	指示可在释放上一个基础连接时释放 DB2e 环境句柄。

表 123. 公用实例构造函数

构造函数	描述
DB2eConnection()	已重载。初始化 *DB2eConnection* 类的新实例。
DB2eConnection(string)	已重载。使用指定连接字符串初始化 *DB2eConnection* 类的新实例。

表 124. 公用实例属性

属性	描述
ConnectionString	获取或设置用于打开数据库的字符串。
ConnectionTimeout	获取或设置在终止尝试并生成错误之前等待尝试建立连接的时间。
Database	获取当前数据库或要在打开连接后使用的数据库的名称。
ServerVersion	获取包含客户机连接至的服务器的版本的字符串。
State	获取连接的当前状态。

表 125. 公用实例方法

方法	描述
BeginTransaction	已重载。在数据库上开始一个事务。
ChangeDatabase	更改与打开的 *DB2eConnection* 相关联的当前数据库。
Close	关闭与数据库的连接。这是关闭任意打开连接的首选方法。
CreateCommand	创建并返回与 *DB2eConnection* 相关联的 *DB2eCommand* 对象。
Open	使用 *ConnectionString* 指定的属性设置打开与数据源的连接。

表 126. 公用实例事件

事件	描述
InfoMessage	当 DB2 Everyplace 发送警告或参考消息时发生。
StateChange	当连接的状态更改时发生。

DB2eDataAdapter 成员

表 127. 公用实例构造函数

构造函数	描述
DB2eDataAdapter()	已重载。初始化 *DB2eDataAdapter* 类的新实例。
DB2eDataAdapter(DB2eCommand)	已重载。使用指定的 SQL SELECT 语句初始化 *DB2eDataAdapter* 类的新实例。

表 127. 公用实例构造函数 (续)

构造函数	描述
DB2eDataAdapter(string, DB2eConnection)	已重载。使用指定的 SQL SELECT 语句和 *DB2eConnection* 对象初始化 *DB2eDataAdapter* 类的新实例。
DB2eDataAdapter(string, string)	已重载。使用指定的 SQL SELECT 语句和连接字符串初始化 *DB2eDataAdapter* 类的新实例。

表 128. 公用实例属性

属性	描述
AcceptChangesDuringFill（继承自 **DataAdapter**）	获取或设置指示在将 *DataRow* 添加至 *DataTable* 后是否在 *DataRow* 上调用 *AcceptChanges* 的值。
ContinueUpdateOnError（继承自 **DataAdapter**）	获取或设置指定在行更新期间遇到错误时是生成异常还是出错行的值。
DeleteCommand	获取或设置用来在数据库中删除记录的 SQL 语句或存储过程。
InsertCommand	获取或设置用来将新记录插入到数据源中的 SQL 语句或存储过程。
MissingMappingAction（继承自 **DataAdapter**）	确定入局数据没有匹配表或列时要采取的操作。
MissingSchemaAction（继承自 **DataAdapter**）	确定现有 *DataSet* 模式没有匹配入局数据时要采取的操作。
SelectCommand	获取或设置用来在数据库中选择记录的 SQL 语句或存储过程。
TableMappings（继承自 **DataAdapter**）	获取提供源表与 *DataTable* 之间的主映射的集合。
UpdateCommand	获取或设置用来在数据库中更新记录的 SQL 语句或存储过程。

表 129. 公用实例方法

方法	描述
Clone	创建包含生成用来与远程对象通信的代理所需的所有相关信息的对象。
Fill（继承自 **DbDataAdapter**）	对 *DataSet* 或 *DataTable* 添加或刷新行以与数据源中的那些行相匹配。
FillSchema（继承自 **DbDataAdapter**）	将 *DataTable* 添加至 *DataSet* 并将模式配置为与数据源中的模式相匹配。
GetFillParameters（继承自 **DbDataAdapter**）	获取用户在执行 SQL SELECT 语句时设置的参数。
Update（继承自 **DbDataAdapter**）	对 DataSet 中的每个插入、更新或删除行调用各自的 INSERT、UPDATE 或 DELETE 语句。

表 130. 公用实例事件

事件	描述
FillError（继承自 **DbDataAdapter**）	在填充操作期间发生错误时返回。
RowUpdated	在对数据库执行命令后的更新操作期间发生。
RowUpdating	在对数据库执行命令之前的*更新*期间发生。

DB2eDataReader 成员

表 131. 公用实例属性

属性	描述
Depth	获取指示对当前行的嵌套的深度的值。
FieldCount	获取当前行中的列数。
IsClosed	指示 *DB2eDataReader* 是否已关闭。
Item	已重载。获取指定列在其本机格式（以给定列顺序）中的值。
RecordsAffected	获取通过执行 SQL 语句更改、插入或删除的行数。

表 132. 公用实例方法

方法	描述
Close	关闭 *DB2eDataReader* 对象。
GetByte	以字节的形式获取指定列的值。
GetBytes	以数组的形式将字节流从指定列位移读取到缓存中（从给定缓冲区位移开始）。
GetDataTypeName	获取源数据类型的名称。
GetDate	以 *DateTime* 对象的形式获取指定列的值。
GetDateTime	以 *DateTime* 对象的形式获取指定列的值。
GetDecimal	以 *Decimal* 对象的形式获取指定列的值。
GetDouble	以双精度浮点数的形式获取指定列的值。
GetFieldType	获取作为该对象的数据类型的*类型*。
GetFloat	以单精度浮点数的形式获取指定列的值。
GetInt16	以 16 位带符号整数的形式获取指定列的值。
GetInt32	以 32 位带符号整数的形式获取指定列的值。
GetInt64	以 64 位带符号整数的形式获取指定列的值。
GetName	获取指定列的名称。
GetOrdinal	获取列顺序（已给定该列的名称）。
GetSchemaTable	返回描述 *DB2eDataReader* 的列元数据的 *DataTable*。
GetString	以字符串的形式获取指定列的值。
GetTime	以 *TimeSpan* 对象的形式获取指定列的值。
GetValue	以其本机格式获取列在指定顺序中的值。
GetValues	获取当前行中的所有属性列。
IsDBNull	获取指示列是否包含不存在或缺少的值的值。
NextResult	使 *DB2eDataReader* 在读取批处理 SQL 语句的结果时进至下一个结果。DB2 Everyplace 目前不支持批处理 SQL 语句。
Read	使 *DB2eDataReader* 进至下一条记录。

DB2eError 成员

表 133. 公用实例属性

属性	描述
Message	获取错误的简短描述。
NativeError	从 DB2 Everyplace 获取错误信息。
SQLState	获取遵循数据库的 ANSI SQL 标准的 5 字符错误代码。

DB2eException 成员

表 134. 公用实例属性

属性	描述
Errors	获取给出有关由 DB2 Everyplace .NET Data Provider 生成的异常的详细信息的一个或多个 *DB2eError* 对象的集合。
Message	获取描述该错误的文本描述。

DB2eParameter 成员

表 135. 公用实例构造函数

构造函数	描述
DB2eParameter()	已重载。初始化 *DB2eParameter* 类的新实例。
DB2eParameter(string, object)	已重载。使用参数名和该参数的值来初始化 *DB2eParameter* 类的新实例。
DB2eParameter(string, DB2eType)	已重载。使用参数名和数据类型来初始化 *DB2eParameter* 类的新实例。
DB2eParameter(string, DB2eType, int)	已重载。使用参数名、数据类型和宽度来初始化 *DB2eParameter* 类的新实例。
DB2eParameter(string, DB2eType, int, string)	已重载。使用参数名、数据类型、宽度和源列名来初始化 *DB2eParameter* 类的新实例。
DB2eParameter(string, DB2eType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object)	已重载。使用下列各项来初始化 *DB2eParameter* 类的新实例: 参数的参数名、数据类型、宽度、参数方向、可空布尔值、数字精度、小数位、源列名、源版本和值。

表 136. 公用实例属性

属性	描述
DB2eType	获取或设置参数的 *DB2eType*。
DbType	
Direction	获取或设置指示参数是仅输入、仅输出、双向还是存储过程返回值参数的值。
IsNullable	获取或设置指示参数是否接受空值的值。
ParameterName	获取或设置 *DB2eParameter* 的名称。
Precision	获取或设置用来表示 *Value* 属性的最大位数。
Scale	获取或设置将 *Value* 解析为的小数位的数目。
Size	获取或设置列中数据的最大大小（以字节计）。
SourceColumn	获取或设置映射至 *DataSet* 并用于装入或返回 *Value* 的源列的名称。
SourceVersion	获取或设置在装入 *Value* 时要使用的 *DataRowVersion*。
Value	获取或设置该参数的值。

表 137. 公用实例方法

方法	描述
ToString	获取包含 *ParameterName* 的字符串。

DB2eTransaction 成员

表 138. 公用实例属性

属性	描述
Connection	指定与该事务相关联的 *DB2eConnection* 对象。
IsolationLevel	对此事务指定 *IsolationLevel*。

表 139. 公用实例方法

方法	描述
Commit	落实数据库事务。
Rollback	从暂挂状态回滚事务。

DB2eType 枚举

指定字段、属性或 *DB2eParameter* 的数据类型。

[Visual Basic]

Public Enum DB2eType

[C#]

public enum DB2eType

下表显示 *DB2eType* 数据类型、DB2 Everyplace 数据类型（显示在圆括号中）与 .NET Framework 类型之间的映射。

表 140. 数据类型映射

成员	描述
SmallInt	确切的数值，精度为 5，小数位为 0（带符号: –32,768 <= n <= 32,767，不带符号: 0 <= n <= 65,535）（SMALLINT）。这将映射为 Int16。
Integer	确切的数值，精度为 10，小数位为 0（带符号: –2[31] <= n <= 2[31] – 1，不带符号: 0 <= n <= 2[32] – 1）（INTEGER）。这将映射为 Int32。
Char	定长字符串（CHAR）。这将映射为 String。
VarChar	变长字符串（VARCHAR）。这将映射为 String。
Decimal	带符号的确切数值，精度至少为 p，小数位为 s，其中 1 <= p <= 31，而 <= p。（DECIMAL）。这将映射为 Decimal。

表 140. 数据类型映射 (续)

成员	描述
Date	格式为 yyyy-mm-dd 的日期数据（DATE）。这将映射为 DateTime。
Time	格式为 hh:mm:ss 的时间数据（TIME）。这将映射为 TimeSpan。
Timestamp	格式为 yyyy-mm-dd-hh.mm.ss.zzzzzz 的时间戳记数据（TIMESTAMP）。这将映射为 DateTime。
Blob	二进制数据流（BLOB）。这将映射为类型为 Byte 的数组。

需求:

NameSpace: IBM.Data.DB2.DB2e 名称空间

Assembly: IBM.Data.DB2.DB2e.dll

IBM Sync Client C-API

本章提供有关 IBM Sync Client C-API 的信息。涉及的主题包括:

IBM Sync Client C-API V8.1 与 V7.2 之间的比较

本节总结版本 8.1 中对 IBM Sync Client C-API 所做的主要更改:

- 现在有三个句柄可用: 服务、配置和引擎。(如果不想执行同步，则不必打开同步引擎句柄。)
- IBM Sync Client C-API V8.1 中的首选项不是持久的，已除去了某些首选项，这些首选项实际上是很重要的信息。例如，旧的 `isyncSetPref` 函数中的主机名、端口、用户名和密码现在是 `iscOpenService` 函数中打开服务句柄所必需的参数。
- 同步方式现在对应用程序是隐式的，当调用同步时同步方式参数不再是必需的。
- 同步侦听器的接口现在是基于事件的。现在将包含事件信息的事件结构传送到应用程序。
- 预订集的同步状态（来自其上次同步）是持久的，可在以后查询。
- 已除去缺省侦听器。当需要事件的缺省操作时，应用程序只返回 `ISCRTNCB_Default` 代码。
- DB2e Everyplace 现在支持数据加密来保护包含敏感数据的表。当开发同步客户机应用程序来使加密表同步时，可以从同步引擎实现（在侦听器中）查询来获取 DB2 Everyplace 用户名和密码。
- 现在通过侦听器将拒绝的记录（包括具有冲突和非法操作的记录）传送到应用程序。

- 现在由应用程序管理日志文件（LOGDB-ISYNC）。即版本 8.1 同步引擎不再像在版本 7.2.1 中一样以本机语言生成日志文件（LOGDB-ISYN）。相反，为了服务目的，同步引擎将生成跟踪文件（TRACE-ISYN），该文件只有英语版。
- IBM Sync Client 引擎将所有文件（包括配置、跟踪文件、数据和首选项（如果适用））存储在一个目录中:
  - 在 Windows CE® 操作系统上: \（根目录）
  - 在 EPOC 操作系统上: C:\Systems\Data\ISync\
  - 在 Palm 操作系统上: 主存储器
  - 在其它操作系统上: 当前目录
- 通过 API 包装器（isynce 库），IBM Sync Client API V7.2.1 的功能仍然受支持，该包装器将处理 API 的向下兼容性。API 包装器也像在版本 7.2.1 中一样在同一目录中以本机语言生成日志文件（LOGDB-ISYN），即:
  - 在 Windows CE® 操作系统上: \Progran Files\ISync\
  - 在 EPOC 操作系统上: C:\Systems\Apps\ISync\
  - 在 Palm 操作系统上: 主存储器
  - 在其它操作系统上: 当前目录

  另外，ISYNCOPTION_SkipConfig 和 ISYNCOPTION_UseAppSignature 选项将不使用 isyncGo 和 isyncSetSyncMoe 函数。

  注: 如果选择使用 IBM Sync Client API V8.1，则需要安装 API 包装器（isynce）库。

表 141 列示 IBM Sync Client C-API V8.1 和 IBM Sync Client V7.2 中函数之间的主要差别。

表 141. IBM Sync Client C-API V8.1 和 V7.2 比较

版本 8.1	版本 7.2	注释
iscGetVersion	isyncGetVersion	iscGetVersion 中不需要句柄。
iscServiceOpen iscConfigOpeniscEngineOpen	isyncOpen	需要打开三个句柄。 在 iscServiceOpen 中指定主机、端口、用户名和密码，且它们不是持久的。
iscServiceClose iscConfigClose iscEngineClose	isyncClose	需要关闭三个句柄。
iscEngineSetListener	isyncSetListener	已更改了侦听器原型和接口。
（无）	isyncDefaultListener	不再有外部缺省侦听器。对于缺省事件处理，返回 ISCRTNCB_Default 代码。
iscEngineSetPref iscEngineGetPref	isyncSetPref isyncGetPref	仅需要两个首选项（跟踪和超时）。这些首选项不是持久的。
iscEngineSync iscEngineSyncConfig	isyncGo	不再需要同步方式。 仅可以使用 iscEngineSyncConfig 更新配置。

表 141. IBM Sync Client C-API V8.1 和 V7.2 比较 (续)

版本 8.1	版本 7.2	注释
iscConfigEnableSubsSet iscConfigDisableSubsSet iscConfigResetSubsSet	isyncSetSyncMode	不再有常规同步方式设置。 可以通过 iscConfigDisableSubsSet 跳过（禁用）预订集的同步。
iscConfigOpenCursor iscConfigCloseCursor iscConfigGetNextSubsSet iscConfigSubsSetIsEnabled iscConfigSubsSetIsReset	isyncGetFirstApp isyncGetNextApp	在迭代预订集之前打开游标。 需要预订集标识才能查询预订集。
iscEngineGetInfo iscConfigPurge iscConfigGetSubsSetStatus		版本 8.1 中的新的 C-API。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 『IBM Sync Client C-API 函数摘要』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 282 页的『IBM Sync Client C-API 函数描述的关键』

IBM Sync Client C-API 函数摘要

表 142 列示由 DB2 Everyplace 所支持的 IBM Sync Client C-API 函数并包含每个函数的用途。

表 142. IBM Sync Client C-API 函数列表

函数名	用途
iscGetVersion	获取 Sync Client C-API 的版本号。

表 143. IBM Service API 函数列表

函数名	用途
iscServiceOpen	打开新的服务。
iscServiceOpenEx	打开使用属性的新服务。
iscServiceClose	关闭服务。

表 144. IBM Configuration API 函数列表

函数名	用途
iscConfigOpen	打开与配置存储的连接。
iscConfigClose	关闭与配置存储的连接。
iscConfigPurge	重新初始化配置。
iscConfigOpenCursor	获取游标（的句柄）以迭代预订集。
iscConfigCloseCursor	处置打开的游标。

表 144. IBM Configuration API 函数列表 (续)

函数名	用途
iscConfigGetNextSubsSet	获取下一个预订集的描述（如果有的话）。
iscConfigEnableSubsSet	启用预订集进行同步。
iscConfigDisableSubsSet	在预订集上禁用同步。
iscConfigResetSubsSet	将预订集更改回复位方式。
iscConfigSubsSetIsEnabled	查询是否启用预订集进行同步。
iscConfigSubsSetIsReset	查询是否复位预订集。
iscConfigGetSubsSetStatus	查询先前同步的同步状态。

表 145. IBM Sync Engine API 函数列表

函数名	用途
iscEngineOpen	打开同步引擎的句柄。
iscEngineClose	关闭同步引擎的打开的句柄。
iscEngineGetInfo	获取有关同步引擎的概要信息。
iscEngineSetListener	将要使用的用户定义的侦听器函数的信息通知同步。
iscEngineListenerPF	用户定义的侦听器函数的数据类型。
iscEngineSetPref	设置首选项。
iscEngineGetPref	检索首选项值。
iscEngineSync	启动同步会话。
iscEngineSyncConfig	使提供的配置与服务器同步。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 『IBM Sync Client C-API 数据类型』
- 第 282 页的『IBM Sync Client C-API 函数描述的关键』

IBM Sync Client C-API 数据类型

表 146 列示由 IBM Sync Client C-API 定义的新的数据类型。在调用 C-API 函数时，确保自变量类型符合函数的原型。

表 146. IBM Sync Client C-API 的数据类型

数据类型	描述
isy_VOID	Void 类型
isy_INT	整数
isy_UINT	无符号的整数
isy_INT16	两个字节的整数
isy_UINT16	两个字节无符号的整数

表 146. IBM Sync Client C-API 的数据类型 (续)

数据类型	描述
isy_INT32	四个字节的整数
isy_UINT32	四个字节无符号的整数
isy_ULONG	无符号的长整数
isy_BYTE	一个字节类型
isy_WORD	一个字类型
isy_DWORD	两个字类型
isy_TCHAR	字符类型
isy_BOOL	布尔类型
HISCSERV	服务句柄的数据类型
HISCCONF	配置句柄的数据类型
HISCENG	同步引擎句柄的数据类型
HISCCSR	预订集的迭代游标的数据类型

ISCEVT	侦听器事件的数据类型:
	`typedef struct {`
	`  isy_INT32  code;`
	`  isy_UINT32  type;`
	`  isy_INT32  retry;`
	`  ISCSTATE  state;`
	`  ISCLISTENARG  *info;`
	`} ISCEVT;`

ISCSTATE	事件状态的数据类型:
	`typedef struct {`
	`  isy_TCHAR  currSubsSet[ISCLEN_SubsSetName];`
	`  isy_TCHAR  currSubs[ISCLEN_SubsName];`
	`  isy_UINT32  subsType;`
	`  isy_INT32  syncProg;`
	`} ISCSTATE;`

ISCLISTENARG	同步侦听器的信息的数据类型，它由字符串自变量（argc 和 argv）的列表组成:
	`typedef struct {`
	`  isy_INT32  argc;`
	`  isy_TCHAR  **argv;`
	`} ISCLISTENARG;`

ISCLISTENCOLUMN	同步侦听器的信息的数据类型，它由包含列位置、主键序列、列类型、数据大小和实际列数据的表列组成:
	`typedef struct {`
	`  isy_INT32  pos;`
	`  isy_INT32  pkseq;`
	`  isy_INT32  type;`
	`  isy_INT32  size;`
	`  isy_BYTE  *data;`
	`} ISCLISTENCOLUMN;`
	
	在 DB2 Everyplace 头文件 `sqlcli.h` 中定义列类型的各种列类型常量。列数据表示为以空结束的文本字符串。这种情况除了 blob 列类型外都成立，对于这种列类型，实际列数据（数据字段）表示为纯字节字符串且不是以空结束的。另外，在大小字段中给出其大小（字节数）。

表 146. IBM Sync Client C-API 的数据类型 (续)

数据类型	描述

ISCLISTENCONFLICT	同步侦听器的信息的数据类型，它由包含表名、操作、列数和列信息数组（ISCLISTENCOLUMN）的表记录组成:
	`typedef struct {`
	`  isy_TCHAR  table[ISCLEN_Table];`
	`  isy_INT32  op;`
	`  isy_INT32  colc;`
	`  ISCLISTENCOLUMN *colv;`
	`} ISCLISTENCONFLICT;`
	
	op 字段指示拒绝的操作，它是以下操作常量之一（实际值在圆括号中）:
	• ISCCONST_OpDelete（1）
	• ISCCONST_OpInsert（2）
	• ISCCONST_OpUpdate（3）

相关概念:

• 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

• 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

• 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
• 第 279 页的『IBM Sync Client C-API 函数摘要』
• 『IBM Sync Client C-API 函数描述的关键』

IBM Sync Client C-API 函数描述

本章描述 IBM Sync Client C-API 中的函数。

IBM Sync Client C-API 函数描述的关键

每个 Sync Client C-API 函数的描述都包含下列各节:

用途 给出函数功能的简要概述。

语法 包含类属 C 原型。该类属原型用于所有环境，包括 Windows。

函数自变量
列示每个函数的自变量以及每个自变量的数据类型、描述和使用类型（输入或输出）。

用法 提供有关如何使用函数的信息并描述任何特殊注意事项。

返回码 列示所有可能的函数返回码。

限制 指示应用每个 Sync Client C-API 函数时的任何差异或限制。

参考 列示相关的 Sync Client C-API 函数。

注意: IBM Sync Client C-API 中没有“诊断”节。

相关概念:

• 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

• 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』

iscGetVersion()

用途: `iscGetVersion()` 获取 Sync Client C-API 的版本号。

语法:

```
isy_UINT32 iscGetVersion();
```

函数自变量:

无。

用法:

`iscGetVersion()` 用于检索 Sync Client C-API 的版本号。版本号作为 32 位无符号整数返回，格式为 *0xmmnnrrxx*，其中，*mm*、*nn* 和 *rr* 分别是主版本号、次版本号和修订版本号的十六进制表示法。*xx* 是保留值。

示例: :

```
isy_UINT32  version;
       int verMajor, verMinor, verModi;
       version = iscGetVersion();
       verMajor = (int) (version >> 24);
       verMinor = (int) ((version >> 16) & 0x000000FF);
       verModi  = (int) ((version >> 8)  & 0x000000FF);
```

返回码:

Sync Client C-API 版本号。

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 299 页的『iscEngineGetInfo()』
- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 282 页的『IBM Sync Client C-API 函数描述的关键』
- 第 279 页的『IBM Sync Client C-API 函数摘要』

iscServiceOpen()

用途: iscServiceOpen() 打开新的服务句柄。

语法:

```
isy_INT32 iscServiceOpen(
    isy_CONST isy_TCHAR* host,
    isy_CONST isy_TCHAR* port,
    isy_CONST isy_TCHAR* username,
    isy_CONST isy_TCHAR* password,
    isy_CONST isy_VOID*  reserved,
    HISCSERV*            phServ);
```

函数自变量:

表 147 列示用于 iscServiceOpen() 函数的有效自变量。

表 147. iscServiceOpen() 自变量

数据类型	自变量	使用	描述
isy_CONST isy_TCHAR*	*host*	输入	主机名或 IP
isy_CONST isy_TCHAR*	*port*	输入	端口号
isy_CONST isy_TCHAR*	*username*	输入	所请求服务的用户名
isy_CONST isy_TCHAR*	*password*	输入	所请求服务的密码
isy_CONST isy_TCHAR*	*reserved*	输入	（已保留）
HISCSERV*	*phServ*	输出	服务的句柄

用法:

iscServiceOpen() 用于请求由主机名或端口号标识的特定服务的新句柄。当请求服务时指定用户名和密码。成功后，服务句柄（HISCSERV）通过 *phServ 返回。否则，*phServ 为 NULL，返回错误代码。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: 内存不足
- ISCRTN_ResourceBusy: 资源被锁定（例如，被另一个应用程序）
- ISCRTN_NotPermitted: 资源不可访问（例如，不可读）
- ISCRTN_NotFound: 未找到资源（例如，未找到路径）
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』

- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 286 页的『iscServiceClose()』
- 『iscServiceOpenEx()』

iscServiceOpenEx()

用途: iscServiceOpenEx() 打开基于属性数组的新服务句柄。

语法:

```
isy_INT32 iscServiceOpenEx(
    isy_CONST isy_TCHAR* URL,
    ISCPROPERTY*          property,
    isy_INT32            propertyNum,
    HISCSERV*            phServ);
```

函数自变量:

表 148列示用于 iscServiceOpenEx() 函数的有效自变量。

表 148. iscServiceOpenEx() 自变量

数据类型	自变量	使用	描述
isy_CONST isy_TCHAR	*URL*	输入	URL 字符串形式的服务器信息

ISCPROPERTY	property	输入	ISCPROPERTY 类型的一组属性:
			`typedef struct {`
			`  isy_TCHAR *key;  //property ID string`
			`  isy_TCHAR *value; //property value string`
			`} ISCPROPERTY;`
			有三个属性可用:
			• *isync.user* – Sync Client 用户名
			• *isync.password* – Sync Client 密码
			• *isync.encoding* – 目标数据的字符编码
			用户名和密码属性是必需的。

isy_INT32	propertyNum	输入	属性的数目。
HISCSERV	phServ	输出	服务的句柄。

用法:

将 iscServiceOpenEx() 与表示为属性数组的设置配合使用以向服务器请求特定服务的新句柄。该服务器由“统一资源定位器”（URL）字符串标识，该字符串可能包含协议、主机名（或 IP）和端口号。如果将 Sync Server 配置为使用“安全套接字层”（SSL），URL 的协议部分应为 “https://”，否则为 “http://”。端口号可省略，而 SSL 和非 SSL 的缺省端口分别为端口 443 和端口 80。所有设置（包括用户名和密码）都是在属性数组中指定的。成功后，将通过 *phServ* 返回服务句柄（HISCSERV）；否则 *phServ* 为 NULL 且返回错误代码。完成后将使用 iscServiceClose() 关闭服务句柄。

示例:

```
      int rc = 0;
HISCSERV hSyncServ;
ISCPROPERTY properties[3] = {{"isync.user", "myUserName"},
                {"isync.password", "myPassword"},
                {"isync.encoding", "ISO8859_1"}}
rc = iscServiceOpenEx("http://localhost.mycom.com:80", properties, 3, &hSyncServ);
```

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: 内存不足
- ISCRTN_ResourceInUse: 资源被锁定（例如，被另一个应用程序）
- ISCRTN_NotPermitted: 资源不可访问（例如，不可读）
- ISCRTN_NotFound: 未找到资源（例如，未找到路径）
- ISCRTN_Failed: 其它情况

限制:

无。

相关参考:

- 第 288 页的『iscConfigClose()』

iscServiceClose()

用途: `iscServiceClose()` 关闭打开的服务句柄。

语法:

```
isy_INT32 iscServiceClose(
    HISCSERV  hServ);
```

函数自变量:

表 149 列示用于 `iscServiceClose()` 函数的有效自变量。

表 149. iscServiceClose() 自变量

数据类型	自变量	使用	描述
HISCSERV	*hServ*	输入	服务句柄

用法:

使用 `iscServiceClose()` 来释放先前打开的服务句柄的存储器。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

对 `iscServiceClose()` 的多个调用可能导致错误，应避免这种情况。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』

- 第 284 页的『iscServiceOpen()』

iscConfigOpen()

用途: iscConfigOpen() 打开与配置存储的连接。

语法:

```
isy_INT32 iscConfigOpen(
    HISCSERV   hServ,
    isy_TCHAR *path,
    HISCCONF  *phConf);
```

函数自变量:

表 150 列示用于 iscConfigOpen() 函数的有效自变量。

表 150. iscConfigOpen() 自变量

数据类型	自变量	使用	描述
HISCSERV	*hServ*	输入	服务句柄
isy_TCHAR*	*path*	输入	工作目录的路径
HISCCONF*	*phConf*	输出	配置连接

用法:

iscConfigOpen() 打开与配置存储的连接，该配置存储是在特定服务的给定路径中指定的。成功后，配置连接（HISCCONF）通过 *phServ 返回。否则，*phServ 为 NULL，返回错误代码。如果这是新服务（新主机或新端口），则会为该服务创建一个新的空配置。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: 内存不足
- ISCRTN_ResourceBusy: 资源被锁定（例如，被另一个应用程序）
- ISCRTN_NotPermitted: 资源不可访问（例如，不可读）
- ISCRTN_NotFound: 未找到资源（例如，未找到路径）
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』

- 『iscConfigClose()』

iscConfigClose()

用途: iscConfigClose() 关闭打开的配置存储连接。

语法:

```
isy_INT32 iscConfigClose(
    HISCCONF    hConf);
```

函数自变量:

表 151 列示用于 iscConfigClose() 函数的有效自变量。

表 151. iscConfigClose() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接

用法:

iscConfigClose() 关闭先前打开的配置存储连接。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 287 页的『iscConfigOpen()』

iscConfigPurge()

用途: iscConfigPurge() 腾空来自配置存储的所有预订信息。

语法:

```
isy_INT32 iscConfigPurge(
    HISCCONF    hConf);
```

函数自变量:

表 152 列示用于 iscConfigPurge() 函数的有效自变量。

表 152. iscConfigPurge() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接

用法:

iscConfigPurge() 除去配置存储中的所有用户预订信息。在下一次同步期间，引擎再次从服务器取装配置并在所有预订集上执行总刷新。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 294 页的『iscConfigResetSubsSet()』

iscConfigOpenCursor()

用途: iscConfigOpenCursor() 获取游标以便重复处理所有预订集。

语法:

```
isy_INT32 iscConfigOpenCursor(
    HISCCONF     hConf,
    HISCCSR    *phCursor);
```

函数自变量:

表 153 列示用于 iscConfigOpenCursor() 函数的有效自变量。

表 153. iscConfigOpenCursor() 自变量

数据类型	自变量	使用	描述
HSYNCCONF	*hConf*	输入	配置连接
HISCCSR*	*phCursor*	输出	用于迭代预订集的返回游标

用法:

当需要在所有预订集上执行迭代过程时，使用 iscConfigOpenCursor() 来获取适当的游标。然后，使用 iscConfigGetNextSubsSet() 获取每个预订集及其相应的描述。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

当调用 iscConfigOpenCursor() 时，所有先前打开的游标都无效且应关闭。处理具有关闭游标的预订集的任何尝试都会生成 ISCRTN_Failed 返回码。即预订集的迭代不能在另一个迭代内嵌套。同样，当使配置同步时（使用 iscEngineSync() 或 iscEngineSyncConfig()），打开的游标无效。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 『iscConfigCloseCursor()』
- 第 291 页的『iscConfigGetNextSubsSet()』

iscConfigCloseCursor()

用途: iscConfigCloseCursor() 处置打开的游标。

语法:

```
isy_INT32 iscConfigCloseCursor(
    HISCCONF     hConf,
    HISCCSR     hCursor);
```

函数自变量:

表 154 列示用于 iscConfigCloseCursor() 函数的有效自变量。

表 154. iscConfigCloseCursor() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
HISCCSR	*hCursor*	输入	用于迭代预订集的游标

用法:

当用 iscConfigOpenCursor() 打开游标但不需要游标时，用 iscConfigCloseCursor() 关闭此游标。否则，打开的游标可能导致内存泄漏或其它配置一致性问题。在游标关闭之后不要尝试使用关闭的句柄，因为这会导致意外错误。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 289 页的『iscConfigOpenCursor()』
- 『iscConfigGetNextSubsSet()』

iscConfigGetNextSubsSet()

用途: `iscConfigGetNextSubsSet()` 获取下一个预订集的描述（如果有的话）并将游标移动到下一个预订集。

语法:

```
isy_INT32 iscConfigGetNextSubsSet(
    HISCCONF     hConf,
    HISCCSR     hCursor,
            isy_TCHAR*  id,
    isy_TCHAR*  name);
```

函数自变量:

表 155 列示用于 `iscConfigGetNextSubsSet()` 函数的有效自变量。

表 155. iscConfigGetNextSubsSet() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
HISCCSR	*hCursor*	输入	用于迭代预订集的游标
isy_TCHAR*	*id*	输出	预订集的标识
isy_TCHAR*	*name*	输出	预订集的名称

用法:

`iscConfigGetNextSubsSet()` 从服务器获取预订集标识，检索预订集名称（如果有的话）并将游标移动到下一个预订集。

示例:

```
        isy_TCHAR  id[ISCLEN_SubsSetID];
     isy_TCHAR  name[ISCLEN_SubsSetName];
isy_INT32  isReset, isEnabled;
     HISCCSR   hCursor;
 isy_INT32 rc;

// start iteration of all subscription sets
rc = iscConfigOpenCursor(hConf, &hCursor);
while (rc == ISCRTN_Succeeded) {
    rc = iscConfigGetNextSubsSet(hConf, hCursor, id, name);
    if (rc == ISCRTN_Succeeded) {
        isReset = iscConfigSubsSetIsReset(hConf, id);
        isEnabled = iscConfigSubsSetIsEnabled(hConf, id);
        // processing the subscription set
        ...
        // get next subscription
    } // end of processing
} // end of iteration
     iscConfigCloseCursor(hConf, hCursor);
```

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Empty: 不再有预订集
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 296 页的『iscConfigSubsSetIsReset()』
- 第 295 页的『iscConfigSubsSetIsEnabled()』

iscConfigEnableSubsSet()

用途: `iscConfigEnableSubsSet()` 启用配置中的预订集进行同步。

语法:

```
isy_INT32 iscConfigEnableSubsSet(
    HISCCONF     hConf,
    isy_TCHAR*  id);
```

函数自变量:

表 156 列示用于 `iscConfigEnableSubsSet()` 函数的有效自变量。

表 156. iscConfigEnableSubsSet() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接

表 156. iscConfigEnableSubsSet() 自变量 (续)

数据类型	自变量	使用	描述
isy_TCHAR*	*id*	输入	预订集标识

用法:

最初启用所有预订集进行同步。iscConfigEnableSubsSet() 和 iscConfigDisableSubsSet() 函数启用与禁用预订集的同步能力，由给定的标识指定预订集。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_NotFound: 未找到预订集。
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 『iscConfigDisableSubsSet()』
- 第 295 页的『iscConfigSubsSetIsEnabled()』

iscConfigDisableSubsSet()

用途: iscConfigDisableSubsSet() 在预订集上禁用同步。

语法:

```
isy_INT32 iscConfigDisableSubsSet(
    HISCCONF     hConf,
    isy_TCHAR*  id);
```

函数自变量:

表 157 列示用于 iscConfigDisableSubsSet() 函数的有效自变量。

表 157. iscConfigDisableSubsSet() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
isy_TCHAR*	*id*	输入	预订集标识

用法:

最初启用所有预订集进行同步。`iscConfigEnableSubsSet()` 和 `iscConfigDisableSubsSet()` 函数启用与禁用预订集的同步能力，由给定的标识指定预订集。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_NotFound: 未找到预订集。
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 292 页的『iscConfigEnableSubsSet()』
- 第 295 页的『iscConfigSubsSetIsEnabled()』

iscConfigResetSubsSet()

用途: `iscConfigResetSubsSet()` 将配置中的预订集复位回复位方式。

语法:

```
isy_INT32 iscConfigResetSubsSet(
    HISCCONF     hConf,
    isy_TCHAR*  id);
```

函数自变量:

表 158 列示用于 `iscConfigResetSubsSet()` 函数的有效自变量。

表 158. iscConfigResetSubsSet() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
isy_TCHAR*	*id*	输入	预订集标识

用法:

如果同步时预订集处于复位方式，则同步引擎删除该预订集的客户机数据。同步引擎只取装（或重新取装）服务器数据；此过程称为刷新。在使预订集同步之后，此预订集*不再*处于复位方式。使用 `iscConfigResetSubsSet()` 来将指定的预订集更改回复位方式。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_NotFound: 未找到预订集。
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 296 页的『iscConfigSubsSetIsReset()』

iscConfigSubsSetIsEnabled()

用途: `iscConfigSubsSetIsEnabled()` 查询是否启用了预订集进行同步。

语法:

```
isy_INT32 iscConfigSubsSetIsEnabled(
    HISCCONF     hConf,
    isy_TCHAR*  id);
```

函数自变量:

表 159 列示用于 `iscConfigSubsSetIsEnabled()` 函数的有效自变量。

表 159. iscConfigSubsSetIsEnabled() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
isy_TCHAR*	*id*	输入	预订集标识

用法:

`iscConfigSubsSetIsEnabled()` 用于查询是否启用了由给定标识指定的预订集进行同步。最初启用所有预订集进行同步。

返回码:

- ISCRTN_True: 启用了预订集进行同步。

- ISCRTN_False: 未启用预订集进行同步。
- ISCRTN_NotFound: 未找到预订集。
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 『iscConfigSubsSetIsReset()』

iscConfigSubsSetIsReset()

用途: `iscConfigSubsSetIsReset()` 查询预订集是否处于复位方式。

语法:

```
isy_INT32 iscConfigSubsSetIsReset(
    HISCCONF     hConf,
    isy_TCHAR*  id);
```

函数自变量:

表 160 列示用于 `iscConfigSubsSetIsReset()` 函数的有效自变量。

表 160. iscConfigSubsSetIsReset() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
isy_TCHAR*	*id*	输入	预订集标识

用法:

最初将所有预订集设置为复位方式。然而，如果使预订集同步，则预订集的方式会更改。使用 `iscConfigResetSubsSet()` 将由给定标识指定的预订集更改回复位方式。

返回码:

- ISCRTN_True: 预订集处于复位方式。
- ISCRTN_False: 预订集未处于复位方式。
- ISCRTN_NotFound: 未找到预订集。
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 295 页的『iscConfigSubsSetIsEnabled()』

iscConfigGetSubsSetStatus()

用途: `iscConfigGetSubsSetStatus()` 获取预订集的同步状态。

语法:

```
isy_INT32 iscConfigGetSubsSetStatus(
    HISCCONF     hConf,
    isy_TCHAR*  id);
```

函数自变量:

表 161 列示用于 `iscConfigGetSubsSetStatus()` 函数的有效自变量。

表 161. iscConfigGetSubsSetStatus() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置连接
isy_TCHAR*	*id*	输入	预订集标识

用法:

使用 `iscConfigGetSubsSetStatus()` 来查询预订集在其上次同步期间（具朋提供的标识）的同步状态。

返回码:

- ISCRTN_Succeeded: 预订集的同步成功。
- ISCRTN_Ready: 启用了预订集。同步过程开始，但尚未使预订集同步。
- ISCRTN_Canceled: 取消了预订集的同步。
- ISCRTN_Failed: 预订集的同步失败。
- ISCRTN_NotFound: 未找到预订集。

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 310 页的『iscEngineSync()』
- 第 295 页的『iscConfigSubsSetIsEnabled()』

iscEngineOpen()

用途: `iscEngineOpen()` 打开同步引擎的句柄。

语法:

```
isy_INT32 iscEngineOpen(
    HISCCONF     hConf,
    HISCENG     *phEngine);
```

函数自变量:

表 162 列示用于`iscEngineOpen()` 函数的有效自变量。

表 162. iscEngineOpen() 自变量

数据类型	自变量	使用	描述
HISCCONF	*hConf*	输入	配置句柄
HISCENG*	*phEngine*	输出	同步引擎的句柄

用法:

当使指定的配置同步时，使用 `iscEngineOpen()` 来打开同步引擎（HISCENG）的句柄。在成功完成同步后，该句柄通过 *phEngine 返回。如果同步未成功完成，则 *phEngine 值为 NULL，返回错误代码。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_OutOfMemory: 内存不足
- ISCRTN_ResourceBusy: 资源被锁定（例如，被另一个应用程序）
- ISCRTN_NotPermitted: 资源不可访问（例如，资源不可读）
- ISCRTN_NotFound: 未找到资源（例如，未找到路径）
- ISCRTN_Failed: 其它情况

限制:

避免对 `iscEngineOpen()` 的多个调用，因为多个调用会打开同步引擎的多个句柄，且可能导致一致性问题。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 『iscEngineClose()』

iscEngineClose()

用途: `iscEngineClose()` 关闭同步引擎的已打开句柄。

语法:

```
isy_INT32 iscEngineClose(
    HISCENG        hEngine);
```

函数自变量:

表 163 列示用于`iscEngineClose()` 函数的有效自变量。

表 163. iscEngineClose() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄

用法:

使用 `iscEngineClose()` 来关闭同步引擎的已打开句柄。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

对 `iscEngineClose()` 的多个调用会导致错误，应避免这种情况。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 298 页的『iscEngineOpen()』

iscEngineGetInfo()

用途: `iscEngineGetInfo()` 获取有关同步引擎的概要信息。

语法:

```
isy_INT32 iscEngineGetInfo(
    HISCENG             hEngine,
    isy_TCHAR    *info,
    isy_INT32     infoLen);
```

函数自变量:

表 164 列示用于 iscEngineGetInfo() 函数的有效自变量。

表 164. iscEngineGetInfo() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄
isy_TCHAR*	*info*	输出	指向存储返回信息的缓冲区的指针
isy_INT32	*infoLen*	输入	提供的缓冲区的大小

用法:

iscEngineGetInfo() 为服务目的提供同步引擎信息。信息的内容和格式将来可能会更改。因此，应用程序应该只显示或记录此信息。不要使用此信息作为应用程序处理的输入。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_ValTruncated: 信息的实际长度比 infoLen 长。
- ISCRTN_Failed: 其它情况

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 283 页的『iscGetVersion()』

iscEngineSetListener()

用途: iscEngineSetListener() 向同步引擎注册用户定义的侦听器函数。在同步会话期间，当发生同步事件（如正在启动的同步）或出错时，将调用侦听器函数。

语法:

```
isy_INT32 iscEngineSetListener(
    HISCENG             hEngine,
    iscEngineListenerPF syncListener,
            isy_UINT32      syncListenerData)
```

函数自变量:

表 165 列示用于 iscEngineSetListener() 函数的有效自变量。

表 165. iscEngineSetListener() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄
iscEngineListenerPF	*syncListener*	输入	用户定义的侦听器函数的地址
isy_UINT32	*syncListenerData*	输入	应用程序要转发到用户定义的侦听器函数的数据

用法:

通过注册用户定义的侦听器函数，应用程序具有进入同步过程的视图。在同步期间，当发生事件或错误时通知应用程序。应用程序可以定制方法来对用户显示这些事件或错误。

示例:

```
// Function syncListener is defined with the following prototype:
isy_INT32 mySyncListener(
                isy_UINT32            listenerData,
    ISCEVT*     event,
    isy_VOID*   pExtraInfo);...
// Handle to the synchronization engine is passed to the listener function
iscEngineSetListener(hEngine, mySyncListener, (isy_UINT32) hEngine);
```

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_Failed: 其它情况

限制:

用户定义的侦听器函数应遵循同步引擎的协议，否则同步引擎可能无法正常工作。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 310 页的『iscEngineSync()』

iscEngineListenerPF

用途: iscEngineListenerPF 定义使用 iscEngineSetListener() 注册的用户定义的侦听器函数应符合的原型。

语法:

```
typedef isy_INT32 (*iscEngineListenerPF)(
                  isy_UINT32            listenerData,
    ISCEVT*     event,
    isy_VOID*   pExtraInfo);
```

函数自变量:

表 166 列示用于 iscEngineSetListenerPF 函数类型的有效自变量。

表 166. iscEngineListenerPF 自变量

数据类型	自变量	使用	描述
isy_UINT32	*listenerData*	输入	通过 *iscEngineSetListener()* 将 *syncListenerData* 自变量中的数据集转发回侦听器函数。
ISCEVT*	*event*	输入	事件对象
isy_VOID*	*pExtraInfo*	输入	保留

用法:

要使用用户定义的侦听器函数来监视同步的进度，必须首先使函数符合 iscEngineSetListenerPF 函数类型。下一步，使用 iscEngineSetListener() 函数注册侦听器函数。然后，当发生同步事件时，将通知用户定义的侦听器函数。event 自变量是一种结构，它包含关于该事件的各种信息。

表 167 列示事件结构中的所有字段和每个字段的用途。

表 167. iscEngineListenerPF 事件字段

字段	描述
类型	事件类型可以是以下值之一（实际值在圆括号中）： **ISCEVTTYPE_Info（1）** 有关同步进度的信息。 **ISCEVTTYPE_Conflict（2）** 同步过程中的冲突操作和已拒绝的操作。 **ISCEVTTYPE_Query（3）** 需要某些信息以便同步继续进行。应用程序必须提供某些必需的信息（基于事件代码）以便同步引擎继续进行。 **ISCEVTTYPE_Retry（4）** 发生异常，重试或取消指令需要继续同步。 **ISCEVTTYPE_Error（5）** 发生错误，同步引擎不能继续使当前预订集同步。 **ISCEVTTYPE_Fatal（6）** 发生致命错误，同步引擎不能继续使预订集同步。

表 167. iscEngineListenerPF 事件字段 (续)

字段	描述
状态	事件状态，它包含下列子字段: **currSubsSet** 预订集名称（如果不为空）。 **currSubs** 预订名称（如果不为空）。 **subsType** 预订类型（如果不为 0），排列如下: • 100–999: 保留 • 1000–9999: 注册的预订类型 • 10000+ : 定制预订类型 预定义的始发者是（实际值在圆括号内）: – ISCSUBSTYPE_Config（100）: 配置 – ISCSUBSTYPE_File（101）: 文件预订 – ISCSUBSTYPE_DB2e（102）: DB2 Everyplace 表预订 **syncProg** 表示为百分比的同步进度。
重试	同一事件的重试次数（如果不为 0）。
信息	可选的特定于事件的信息（如果不为 NULL），它是非冲突事件的字符串自变量的数组。对于冲突事件，数据类型是 ISCLISTENCONFLICT。

event.info 字段包含某些可选的特定于事件的信息。事件代码用于标识和解释此信息。

表 168 按事件类型的类别列示所有事件代码。

表 168. iscEngineListenerPF 事件代码. 事件类型: ISCEVTTYPE_Info

事件代码	事件信息（argc）	描述
ISCEVT_InfGeneral（1000）	NULL	概要信息（用于调试）
ISCEVT_InfSyncStarted（1001）	NULL	同步已启动。
ISCEVT_InfPrepMsg（1002）	NULL	正在准备消息。
ISCEVT_InfSendMsg（1003）	NULL	正在发送消息。
ISCEVT_InfWaitMsg（1004）	NULL	正在等待服务器应答。
ISCEVT_InfApplyMsg（1005）	NULL	应用服务器消息。
ISCEVT_InfCancelingSync（1006）	NULL	正在取消同步。
ISCEVT_InfSubsSetStarted（1007）	NULL	预订集同步已启动。
ISCEVT_InfSyncingSubs（1008）	NULL	预订同步已启动。
ISCEVT_InfSubsSetFailed（1009）	NULL	预订集同步失败。
ISCEVT_InfSubsSetCanceled（1010）	NULL	已取消预订集同步。
ISCEVT_InfSubsSetSucceeded（1011）	NULL	预订集同步成功完成。
ISCEVT_InfSyncSucceeded（1012）	NULL	同步成功。
SCEVT_InfSyncFailed（1013）	NULL	同步（在某些预订集上）失败。
ISCEVT_InfSyncCanceled（1014）	NULL	（用户）已取消同步。
ISCEVT_InfSyncProg（1015）	NULL	表示为百分比的同步进度。
ISCEVT_InfNoNewChange（1016）	NULL	没有新的服务器更改；跳过拉出和确认阶段。
ISCEVT_InfLoginFailed（1017）	NULL	指定的登录信息通过认证过程。

表 169. iscEngineListenerPF 事件代码. 事件类型: ISCEVTTYPE_Conflict

事件代码	事件信息（argc）	描述
ISCEVT_CftReject（2000）	ISCLISTENCONFLICT	在同步中发现数据冲突。实际冲突数据以 ISCLISTENCONFLICT 结构形式表示，而其参考指针将通过 *event.info* 给回应用程序。

表 170. iscEngineListenerPF 事件代码. 事件类型: ISCEVTTYPE_Retry

事件代码	事件信息（argc）	描述
ISCEVT_TryNetConn（4601）	NULL	重试连接到服务器。
ISCEVT_TrySendRequest（4602）	NULL	重试发送请求。
ISCEVT_TryRecvReply（4603）	NULL	重试接收应答。
ISCEVT_TryRecvTimeout（4604）	NULL	等待更长时间以接收应答。
SCEVT_TryRecvAck（4605）	NULL	重试接收确认。

表 171. iscEngineListenerPF 事件代码. 事件类型: ISCEVTTYPE_Query

事件代码	事件信息（argc）	描述
ISCEVT_QueCancel（5000）	NULL	查询用户是否取消并返回（实际值在圆括号内）: • ISCRTNCB_ReplyYes（3）: 如果用户取消 • ISCRTNCB_ReplyNo（2）: 如果用户选择继续 • ISCRTNCB_Default（0）: 缺省值（即 ISCRTNCB_ReplyNo）
ISCEVT_QueCancelUponError（5001）	NULL	查询用户是否取消并返回（实际值在圆括号内）: • ISCRTNCB_ReplyYes（3）: 如果用户取消 • ISCRTNCB_ReplyNo（2）: 如果用户选择继续 • ISCRTNCB_Default（0）: 缺省值（即 ISCRTNCB_ReplyNo）
ISCEVT_QueLogin（5002）	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	适配器请求的登录信息。侦听器必须在事件信息中提供请求的信息并应该用实际值（1）返回 *ISCRTNCB_Done*。 数据源的目标名称 用于保存用户名的空白缓冲区 用于保存密码的空白缓冲区
ISCEVT_QueSubsTarget (5003)	ISCLISTENARG(1) info->argv[0]	适配器请求的数据库请求。侦听器可能在事件信息中提供请求的信息并返回 ISCRTNCB_Done 或返回 ISCRTNCB_Default 以使用缺省目标目录。 用于预订的目录。

表 172. iscEngineListenerPF 事件代码. 事件类型: ISCEVTTYPE_Error

事件代码	事件信息（argc）	描述
ISCEVT_ErrOpenAdapter（300）	NULL	未能打开适配器 <*适配器名称*>。
ISCEVT_ErrLoadAdapter（301）	NULL	未能装入适配器 <*适配器名称*>。
ISCEVT_ErrCloseAdapter（302）	NULL	未能关闭适配器 <*适配器名称*>。
ISCEVT_ErrAuthenticateKey（306）	NULL	认证失败（加密密钥无效）; 同步异常终止。
ISYNCEVT_ErrClientCryptoFailed（307）	NULL	客记机加密或解密失败; 同步异常终止。
ISCEVT_ErrEncryptNotAvail（308）	NULL	加密不可用。
ISCEVT_ErrEncryptLibOpen（309）	NULL	未能打开加密库。
ISCEVT_ErrSubsNotFound（311）	NULL	服务器未找到预订。
ISCRTN_ErrSubsNotAvail（312）	NULL	服务器阻塞了预订。
ISCRTN_ErrSubsDefAltered（316）	NULL	自上次同步引擎使配置同步之后预订定义已改变。
ISCEVT_ErrAllocResource（400）	NULL	未能分配适配器资源。
ISCEVT_ErrConnectData（401）	NULL	未能连接到目标数据。
ISCEVT_ErrDisconnectData（402）	NULL	未能与目标数据断开连接。
ISCEVT_ErrNoData（403）	NULL	未找到数据。
ISCEVT_ErrMessageFormat（412）	NULL	意外的消息格式。

表 172. iscEngineListenerPF 事件代码 (续). 事件类型: ISCEVTTYPE_Error

事件代码	事件信息（argc）	描述
ISCEVT_ErrNotFound（413）	ISCLISTENARG(2) info->argv[0] info->argv[1]	未找到请求的数据。 数据源的目标名称 数据名称
ISCEVT_ErrEndOfData（414）	NULL	意外的数据结束。
ISCEVT_ErrDataTooLong（415）	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	数据太长被截断。 数据源的目标名称 数据名称 数据元素名称（如果不为空）
ISCEVT_ErrSyncDisabled（417）	NULL	服务器报告未启用用户。
ISCEVT_ErrServerException（418）	NULL	服务器报告未知异常。
ISCEVT_ErrReadOnly（420）	ISCLISTENARG(2) info->argv[0] info->argv[1]	试图更新只读数据。 数据源的目标名称 数据名称
ISCEVT_ErrOperation（421）	NULL	对数据的非法操作。
ISCEVT_ErrUnauthorized（423）	NULL	无权存取目标数据。
ISCEVT_ErrNotAvailable（424）	ISCLISTENARG(2) info->argv[0] info->argv[1]	请求的数据不可用。 数据源的目标名称 数据名称
ISCEVT_ErrNotSupported（425）	ISCLISTENARG(3) info->argv[0] info->argv[1] info->argv[2]	不支持请求的数据。 数据源的目标名称 数据名称 数据元素名称（如果不为空）
ISCEVT_ErrNetConn（601）	NULL	未能连接到服务器。
ISCEVT_ErrSendRequest（602）	NULL	未能发送请求。
ISCEVT_ErrRecvReply（603）	NULL	未能接收应答。
ISCEVT_ErrRecvTimeout（604）	NULL	接收应答时发生超时。
ISCEVT_ErrRecvAck（605）	NULL	未能接收到确认。
ISCRTN_ErrCloseNetLib（608）	NULL	未能关闭网络库
ISCEVT_ErrOutOfMemory（610）	NULL	内存不足。
ISCEVT_ErrInternal（698）	ISCLISTENARG(1) info->argv[0]	发生了其它内部错误。 错误状态（作为字符串）。

表 173. iscEngineListenerPF 事件代码. 事件类型: ISCEVTTYPE_Fatal

事件代码	事件信息（argc）	描述
ISCEVT_FatSyncCfgAbort（303）	NULL	配置同步失败; 同步异常终止。
ISCEVT_FatAuthenticateFailed（304）	NULL	认证失败; 同步异常终止。
ISCEVT_FatIncompVersion（310）	NULL	不兼容的同步客户机版本。
ISCEVT_FatInvalidSession（313）	NULL	无效的会话标识。
ISCEVT_FatSyncGroup（314）	NULL	用户不属于任何同步组。
ISCEVT_FatRegisterDevice（315）	NULL	未能为用户注册设备。
ISCEVT_FatNetOpenConn（600）	NULL	未能打开与服务器的连接。
ISCEVT_FatOpenNetLib（606）	NULL	未装入“网络”库。
ISCEVT_FatResolveHost（609）	NULL	未能解析主机名。

表 173. iscEngineListenerPF 事件代码 (续). 事件类型: ISCEVTTYPE_Fatal

事件代码	事件信息（argc）	描述
ISCEVT_FatServerForbidden（611）	NULL	禁止与服务器同步。
ISCEVT_FatServerNotFound（612）	NULL	未找到服务器
ISCEVT_FatServer（613）	NULL	服务器错误。
ISCEVT_FatServerNotAvail（614）	NULL	服务器不响应。
ISCEVT_FatNetUnknown（699）	NULL	未知网络错误。

示例:

```
isy_INT32 mySyncListener(
                isy_UINT32            listenerData,
    ISCEVT*     event,
    isy_VOID*  pExtraInfo)
{
    char *statusMsg = appEventCodeToMessage(event);
    int   timesRetried;

    switch (event->type) {
        case ISCEVTTYPE_Info:
        appStatusBar(statusMsg);
        // appStatusBar can be any routine which shows the statusMsg (e.g., in a
        // status bar)
                return ISCRTNCB_Done;

        case ISCEVTTYPE_Retry:
            timesRetried = event->retry;
        if (timesRetried >= 3) // Try no more than 3 times
                        return ISCRTNCB_ReplyNo;
                else
            return appRetryCancelBox(statusMsg, 10); // 10 sec timeout
        // appRetryCancelBox can be any routine which shows a window with two
        // buttons: Cancel and Retry.  It returns
        //   ISCRTNCB_ReplyYes, if user clicks Retry
        //   ISCRTNCB_ReplyNo, if user clicks Cancel
        // If the user doesn't make choice, it returns ISCRTNCB_Default.
                    break;

        // all other event types, don't care
      default:
            return ISCRTNCB_Default;
    } // switch (event->type)
} // mySyncListener
```

返回码:

- ISCRTNCB_ReplyYes ： 用户对查询应答是。
- ISCRTNCB_ReplyNo* ： 用户对查询应答否。
- ISCRTNCB_Default ： 没有应答，执行缺省操作。

如果事件类型是 ISCEVTTYPE_Retry，侦听器函数返回下列其中一个代码:

如果事件类型是 ISCEVTTYPE_Query，返回码的意义取决于事件代码的值。换句话说，侦听器检查事件代码并返回适当的值。但如果用户没有应答查询，应用程序返回下列代码:

- ISCRTNCB_Default ： 没有应答，执行缺省操作。

对于除 ISCEVTTYPE_Retry 和 ISCEVTTYPE_Query 以外的事件类型，同步引擎忽略返回码。侦听器只返回 ISCRTNCB_Done。

注意: 对于不感兴趣的那些事件，侦听器函数只返回 ISCRTNCB_Default 并允许同步引擎执行缺省操作。

注意: 以上的星号（*）指示各种事件类型的缺省操作。

限制:

用户定义的侦听器函数应遵循同步引擎的协议。否则，同步引擎可能无法正常工作。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 310 页的『iscEngineSync()』
- 第 300 页的『iscEngineSetListener()』

iscEngineSetPref()

用途: `iscEngineSetPref()` 设置同步引擎的首选项。

语法:

```
isy_INT32 iscEngineSetPref(
    HISCENG              hEngine,
    isy_CONST isy_INT32  prefID,
            isy_CONST isy_TCHAR *prefVal);
```

函数自变量:

表 174 列示用于 `iscEngineSetPref()` 函数的有效自变量。

表 174. iscEngineSetPref() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄
isy_CONST isy_INT32	*prefID*	输入	首选项标识，它具有下列值之一: • ISCPREF_Timeout: 接收消息的超时长度 • ISCPREF_Trace: 详细跟踪。

表 174. iscEngineSetPref() 自变量 (续)

数据类型	自变量	使用	描述
isy_CONST isy_TCHAR*	*prefVal*	输入	要设置的新的首选项值。有某些预定义的首选项常量。 对于 ISCPREF_Trace 首选项: • ISCCONST_TraceON: 打开详细的调试跟踪 • ISCCONST_TraceOFF: 关闭详细的调试跟踪 对于 ISCPREF_Timeout 首选项: • ISCCONST_TimeoutNever: 等待服务器应答时从不超时。 • ISCCONST_TimeoutMinimum: 最小超时长度

用法:

使用 `iscEngineSetPref()` 来设置同步引擎的首选项。这些首选项不是持久的，每次打开同步引擎的新句柄时必须复位它们。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_UnknownID: 未知
- ISCRTN_ValTooLong: 给定的 prefVal 的长度太长。
- ISCRTN_Failed: 其它错误

限制:

提供的首选项值应在指定的首选项限制内:

- ISCPREF_Trace: 1
- ISCPREF_Timeout: 11

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 『iscEngineGetPref()』

iscEngineGetPref()

用途: `iscEngineGetPref()` 检索当前首选项设置。

语法:

```
isy_INT32 iscEngineGetPref(
    HISCENG              hEngine,
    isy_CONST isy_INT32  prefID,
    isy_TCHAR           *prefVal,
    isy_CONST isy_INT32  prefLen);
```

函数自变量:

表 175 列示用于 iscEngineGetPref() 函数的有效自变量。

表 175. iscEngineGetPref() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄
isy_CONST isy_INT32	*prefID*	输入	首选项标识，它具有下列值之一: • ISCPREF_Timeout: 接收消息的超时长度 • ISCPREF_Trace: 详细跟踪。
isy_TCHAR*	*prefVal*	输出	指向用来存储返回的首选值的缓冲区的指针。有某些预定义的首选项常量。 对于 ISCPREF_Trace 首选项: • ISCCONST_TraceON: 打开详细的调试跟踪 • ISCCONST_TraceOFF: 关闭详细的调试跟踪 对于 ISCPREF_Timeout 首选项: • ISCCONST_TimeoutNever: 等待服务器应答时从不超时。 • ISCCONST_TimeoutMinimum: 最小超时长度
isy_CONST isy_INT32	*prefLen*	输入	提供的缓冲区的大小（prefVal）

用法:

使用 iscEngineGetPref() 获取同步引擎的首选项设置（它是缺省值或由 iscEngineSetPref() 设置的值）。

返回码:

- ISCRTN_Succeeded: 正常
- ISCRTN_UnknownID: 提供了未知 prefID
- ISCRTN_ValTruncated： 首选项值的实际长度大于 prefLen。
- SCRTN_Failed： 其它错误

限制:

提供的缓冲区应足够大，以存储各种首选项的值:

- ISCPREF_Trace: 1
- ISCPREF_Timeout: 11
- ISCPREF_CodePage: 15

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 307 页的『iscEngineSetPref()』

iscEngineSync()

用途: `iscEngineSync()` 启动同步会话。

语法:

```
isy_INT32 iscEngineSync(
    HISCENG        hEngine);
```

函数自变量:

表 176 列示用于`iscEngineSync()` 函数的有效自变量。

表 176. iscEngineSync() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄

用法:

使用 `iscEngineSync()` 来启动同步会话，该同步会话使在 `iscEngineOpen()` 中指定的配置同步。如果从未使预订集同步，则预订集处于复位方式。当同步引擎在该预订集上执行同步时，同步客户机从 Sync Server 取装数据；此过程称为刷新。完成刷新以后，当使预订集再次同步时，同步引擎使更改的数据同步；此过程称为同步。同步引擎始终首先使配置同步。如果配置同步失败，则同步引擎不会继续处理后续预订集，同步会话停止。如果同步引擎在一个预订集上（但不是在配置上）失败，同步引擎继续处理剩余的预订集（如果有的话）。

返回码:

- ISCRTN_Succeeded: 同步成功结束。
- ISCRTN_Failed: 同步失败。
- ISCRTN_Canceled: 用户取消了同步。

`iscEngineSync()` 的返回码是它已同步的所有预订集的同步状态的聚集（以下列示的首选项）:

ISCRTN_Canceled > ISCRTN_Failed > ISCRTN_Succeeded

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』
- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 288 页的『iscConfigPurge()』
- 『iscEngineSyncConfig()』

iscEngineSyncConfig()

用途: `iscEngineSyncConfig()` 启动只使配置同步的同步会话。

语法:

```
isy_INT32 iscEngineSyncConfig(
    HISCENG        hEngine);
```

函数自变量:

表 177 列示用于 `iscEngineSyncConfig()` 函数的有效自变量。

表 177. iscEngineSyncConfig() 自变量

数据类型	自变量	使用	描述
HISCENG	*hEngine*	输入	同步引擎的句柄。

用法:

当配置在服务器上更改时，`iscEngineSyncConfig()` 更新配置而不会使所有预订集重新同步。

返回码:

- ISCRTN_Succeeded: 同步成功结束。
- ISCRTN_Failed: 同步失败。
- ISCRTN_Canceled: 用户取消了同步。

限制:

无。

相关概念:

- 第 109 页的『样本 Sync Client C/C++ 应用程序』

相关任务:

- 第 15 页的『使用 C/C++ 开发 DB2 Everyplace Sync Client 应用程序』

相关参考:

- 第 277 页的『IBM Sync Client C-API V8.1 与 V7.2 之间的比较』

iscEngineSyncConfig()

- 第 280 页的『IBM Sync Client C-API 数据类型』
- 第 279 页的『IBM Sync Client C-API 函数摘要』
- 第 310 页的『iscEngineSync()』
- 第 288 页的『iscConfigPurge()』

第 19 章 DB2 Everyplace 系统目录基本表

数据库管理器创建并维护系统目录基本表集。本附录包含每一系统目录基本表的描述，此表包括列名和数据类型。所有的系统目录基本表都由数据库管理器创建。不能显式地创建或删除系统目录基本表。在响应 SQL 数据定义语句、环境例程和某些实用程序的正常操作期间，更新系统目录基本表。通过常规 SQL 查询工具，系统目录基本表中的数据是可用的。不能使用常规 SQL 数据操作命令修改系统目录基本表。为了访问系统目录表，需要使用定界标识。

表 178. 系统目录基本表

描述	目录基本表
表	313
列	313
引用约束	314
用户	314

DB2eSYSTABLES:

此系统目录基本表为每个创建的表包含一行。所有的目录表在 DB2eSYSTABLES 目录中都有条目。

表 179. DB2eSYSTABLES 系统目录基本表

列名	数据类型	可空	描述
TNAME	VARCHAR (19)		表名
NUMCOLS	INTEGER (4)		列数
FLAGS	INTEGER (4)		（仅内部使用）
NUMKEY	INTEGER (4)		主键中的列数
CHK	BLOB (512)	是	检查约束（仅内部使用）
IDXINFO	BLOB (700)	是	索引（仅内部使用）
NUMREFS	INTEGER (4)	是	主键和外键（仅内部使用）
F_ID	INTEGER (4)	是	（仅内部使用）
PD	BLOB (4096)	是	（仅内部使用）

DB2eSYSCOLUMNS:

此系统目录基本表为每个为表定义的列包含一行。

表 180. DB2eSYSCOLUMNS 系统目录基本表

列名	数据类型	可空	描述
CNAME	VARCHAR (19)		列名
TNAME	VARCHAR (19)		表名
TYPE	INTEGER (4)		数据类型
ATTR	INTEGER (4)		（仅内部使用）
LENGTH	INTEGER (4)		列长度
POS	INTEGER (4)		列号
FLAGS	INTEGER (4)		（仅内部使用）
KEYSEQ	INTEGER (4)		主键中列的顺序位置
SCALE	INTEGER (4)		小数列的小数位
DEF	VARCHAR (128)	是	缺省值（内部使用）

DB2eSYSRELS:

此系统目录基本表为每个引用约束包含一行。

表 181. DB2eSYSRELS 系统目录基本表

列名	数据类型	可空	描述
RMD_ID	INTEGER (4)		主键和外键（仅内部使用）
PKTABLE_NAME	VARCHAR (19)		父表名称
PKCOLUMN_NAME	VARCHAR (19)		父表主键列
FKTABLE_NAME	VARCHAR (19)		子表名称
FKCOLUMN_NAME	VARCHAR (19)		子表外键列名
ORDINAL_POSITION	INTEGER (4)		外键引用中的列位置

DB2eSYSUSERS:

当创建第一个已加密的表或执行第一个 GRANT 语句时，自动创建 DB2eSYSUSERS 表。此表严格地受数据库和加密数据约束，不能将它移动到包含不同加密数据的另一个 DB2 Everyplace 数据库。

此系统目录基本表为对数据库定义的每个注册的用户名包含一行。

表 182. DB2eSYSCOLUMNS 系统目录基本表

列名	数据类型	可空	描述
USERNAME	VARCHAR (19)		主键的一部分且区分大小写。与此行相关的用户的名称。
DATABASENAME	VARCHAR (19)		供以后使用。存储空字符串。主键的一部分。
TABLENAME	VARCHAR (19)		供以后使用。存储空字符串。主键的一部分。
ENCMETHOD	VARCHAR (198)		供以后使用。存储空字符串。主键的一部分。
PRIVILEGES	CHAR (19)	是	定义用户特权。当前仅允许值“E”，它指加密。
ENCKEYDATA	BLOB (64)	是	用于重新生成加密密钥。
ATTIME	TIMESTAMP (26)	是	添加用户或最近修改记录的时间，以最近的时间为准。
VALIDATE	BLOB (64)	是	验证记录是可信的且用户是由已认证的用户添加的。
GRANTOR	VARCHAR (19)	是	在列 1 中注册用户名的用户名。
INTERNALDATA	BLOB (255)	是	（内部将来使用）

第 20 章 DB2 Everyplace 限制

下表描述了某些 DB2 Everyplace 和 SQL 限制。遵循最严格的约束条件有助于程序员设计易于移植的应用程序。这些限制中的许多限制可能会因设备所施加的物理内存和系统限制而进一步地受到限制。

表 183. DB2 Everyplace 数据库和 SQL 限制

描述	限制
最大表大小（在 32 位系统上）	2 吉字节
数据库的最大长度	75 字节
数据仓库中表的最大数	65535
表中索引的最大数	15
表中外键的最大数	8
索引中列的最大数	8
主键中列的最大数	8
SQL 语句的最大长度	64 千字节
至数据仓库路径的连接的最大数	1
表中的最大行数	受表大小限制
表中的最大列数	256
CHAR 列的最大长度	32767 字节
VARCHAR 或 BLOB 列的最大长度	32767 字节
行的 32767 定长列的最大累积长度	32767 字节
每个连接的最大语句句柄数	20
检查约束的最大长度	512 字节
小数位的最大大小	31 位
单个索引中的每列的最大长度	1024 字节

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』

第 21 章 DB2 Everyplace 保留字

下列 DB2 Everyplace 保留字不能用作标识，除非它们被指定为定界标识。例如:

以下语句将导致 SQL 错误:

```
CREATE TABLE tab1 (select int)
```

使用双引号标记，这样才不会导致 SQL 错误:

```
CREATE TABLE tab1 ("select" int)
```

DB2 Everyplace 保留字:

```
ALL, AND, AS, ASC,
BEGIN, BLOB, BY,
DATABASE
CALL, CHAR, CHAR, CHECK, COMMIT, CONCAT, CREATE, CURRENT,
DATE, DECIMAL, DEFAULT, DELETE, DESC, DISTINCT, DROP,
ENCRYPT
FETCH, FOR, FOREIGN, FROM,
GRANT, GROUP,
IN, INDEX, INSERT, INT, INTEGER, INTO, IS,
KEY,
LIKE, LIMIT,
NEW, NOT, NULL,
OF, ON, ONLY, OR, ORDER,
PRIMARY,
READ, REFERENCES, REORG,REVOKE,ROLLBACK,
SELECT, SET, SMALLINT,
TABLE, TIME, TIMESTAMP, TO,TRANSACTION
UPDATE, UPSERT, USING
VALUES, VARCHAR,
WHERE, WITH
```

为了将来具有兼容性，不要将下列 IBM SQL 和 ISO/ANSI SQL92 保留字用作标识。DB2 Everyplace 当前未使用的 IBM SQL 保留字是:

```
ACQUIRE ADD AFTER ALIAS ALLOCATE
ALLOW ALTER ANY ASUTIME AUDIT AUTHORIZATION
AUX AUXILIARY AVG BEFORE BETWEEN BINARY
BUFFERPOOL CALLED CAPTURE CASCADED CASE
CAST CCSID CHARACTER CLOSE CLUSTER COLLECTION
COLLID COLUMN COMMENT CONDITION CONNECT
CONNECTION CONSTRAINT CONTAINS CONTINUE
COUNT COUNT_BIG CROSS CURRENT_DATE CURRENT_LC_PATH
CURRENT_PATH CURRENT_SERVER CURRENT_TIME
CURRENT_TIMESTAMP CURRENT_TIMEZONE CURRENT_USER
CURSOR DATA DATABASE DAY DAYS DBA DBINFO
DBSPACE DB2GENERAL DB2SQL DECLARE DESCRIPTOR
DETERMINISTIC DISALLOW DISCONNECT DO DOUBLE
DSSIZE DYNAMIC EDITPROC ELSE ELSEIF END
END-EXEC ERASE ESCAPE EXCEPT EXCEPTION
EXCLUSIVE EXECUTE EXISTS EXIT EXPLAIN
EXTERNAL FENCED  FIELDPROC FILE FINAL FREE
FULL FUNCTION GENERAL GENERATED GO GOTO
GRANT GRAPHIC HANDLER HAVING HOUR HOURS
IDENTIFIED IF IMMEDIATE INDICATOR INNER
INOUT INSENSITIVE INTEGRITY INTERSECT
ISOBID ISOLATION JAVA JOIN LABEL LANGUAGE
LC_CTYPE LEAVE LEFT LINKTYPE LOCAL LOCALE
LOCATOR LOCATORS LOCK LOCKSIZE LONG LOOP
MAX MICROSECOND MICROSECONDS MIN MINUTE
```

```
MINUTES MODE MODIFIES MONTH MONTHS NAME
NAMED NHEADER NO NODENAME NODENUMBER NULLS
NUMPARTS OBID OPEN OPTIMIZATION OPTIMIZE
OPTION OUT OUTER PACKAGE PAGE PAGES
PARAMETER PART PARTITION PATH PCTFREE
PCTINDEX PIECESIZE PLAN POSITION PRECISION
PREPARE PRIQTY PRIVATE PRIVILEGES PROCEDURE
PROGRAM PSID PUBLIC QUERYNO READS RECOVERY
RELEASE RENAME REPEAT RESET RESOURCE RESTRICT
RESULT RETURN RETURNS REVOKE RIGHT ROW ROWS
RRN RUN SCHEDULE SCHEMA SCRATCHPAD SECOND
SECONDS SECQTY SECURITY SHARE SIMPLE SOME
SOURCE SPECIFIC SQL STANDARD STATIC STATISTICS
STAY STOGROUP STORES STORPOOL STYLE SUBPAGES
SUBSTRING SUM SYNONYM TABLESPACE THEN TO
TRANSACTION TRIGGER TRIM TYPE UNDO UNION
UNIQUE UNTIL USAGE USER USING VALIDPROC
VARIABLE VARIANT VCAT VIEW VOLUMES WHEN
WHILE WLM WORK WRITE YEAR YEARS
```

IBM SQL 未使用的 ISO/ANS SQL92 保留字如下所示。

```
ABSOLUTE ACTION ARE ASSERTION AT BIT_LENGTH
BOTH CATALOG CHAR_LENGTH CHARACTER_LENGTH
COALESCE COLLATE COLLATION CONSTRAINTS CONVERT
CORRESPONDING DEALLOCATE DEC DEFERRABLE DEFERRED
DESCRIBE DIAGNOSTICS DOMAIN EXEC EXTRACT FALSE
FIRST FLOAT FOUND FULL GET GLOBAL IDENTITY
INITIALLY INPUT INTERVAL LAST LEADING LEVEL
LOWER MATCH MODULE NAMES NATIONAL NATURAL
NCHAR NEXT NULLIF NUMERIC OCTET_LENGTH OUTPUT
OVERLAPS PAD PARTIAL PRESERVE PRIOR REAL
RELATIVE SCROLL SECTION SESSION SESSION_USER
SIZE SPACE SQLCODE SQLERROR SQLSTATE SYSTEM_USER
TEMPORARY TIMEZONE_HOUR TIMEZONE_MINUTE
TRAILING TRANSLATION TRUE UNKNOWN UPPER
VALUE VARYING WHENEVER ZONE
```

相关参考:

- 第 127 页的『DB2 Everyplace SQL 语句支持的概述』

第 22 章 本地语言支持（NLS）

本章包含有关 DB2 Everyplace 提供的本地语言支持（NLS）的信息，包括有关国家或地区、语言、支持的代码页（代码集）的信息以及如何配置 DB2 Everyplace NLS 功能及将其与设备和应用程序配合使用的信息。DB2 Everyplace 支持单字节、双字节、多字节字符集和 UNICODE。所有 Win32 操作系统均支持 UNICODE 和非 UNICODE（ANSI）。阅读下列主题以获取如何支持代码页和 UNICODE 的解释。

- 『DB2 Everyplace NLS 支持（根据操作系统）』
- 第 320 页的『Java 应用程序中的字符编码』
- 第 321 页的『DB2 Everyplace 语言使能器』
- 第 322 页的『DB2 Everyplace UNICODE 支持』

DB2 Everyplace NLS 支持（根据操作系统）

表 184 列出哪些操作系统以及对应的语言具有 NLS 支持。

表 184. NLS 支持

语言	Win32	WinCE	Linux	Palm OS	Symbian OS	Neutrino
英语	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页	UNICODE	UTF-8
法语	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页	UNICODE	UTF-8
德语	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页	UNICODE	UTF-8
意大利语	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页	UNICODE	UTF-8
西班牙语	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页	UNICODE	UTF-8
简体中文	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页 • 安装使能器	N/A	UTF-8
繁体中文	代码页 / UNICODE	UNICODE • 安装 Pocket PC 的使能器	代码页 / UTF-8	代码页 • 安装使能器 Acer S60 具有内置繁体中文 Palm OS。	N/A	UTF-8
韩国语	代码页 / UNICODE	UNICODE • 安装使能器	代码页 / UTF-8	代码页 • 安装使能器	N/A	UTF-8

表 184. NLS 支持 (续)

语言	Win32	WinCE	Linux	Palm OS	Symbian OS	Neutrino
日语	代码页 / UNICODE	UNICODE	代码页 / UTF-8	代码页	N/A	UTF-8
希伯莱语	N/A	N/A	N/A	代码页 • 安装使能器	N/A	N/A
捷克语	代码页 / UNICODE	UNICODE • 安装使能器	N/A	代码页 • 安装使能器	UNICODE	N/A
阿拉伯语	N/A	N/A	N/A	代码页 • 安装使能器	N/A	N/A

对于不带 UNICODE 支持的 Palm OS、QNX Neutrino、Linux、Windows NT 和 Windows 2000 操作系统，使用语言环境信息来确定正确的代码页。DB2 Everyplace 内没有内部字符串转换。每个传送的字符串都是按现状存储的。查询应用程序必须使用在存储时使用的代码页设置。这类似于 DB2 通用数据库提供 NLS 的方式。DB2 Everyplace 未提供代码页转换功能。在使用特定代码页的系统上创建的 DB2 Everyplace 数据库仅能在使用相同代码页的系统上部署。使用特定代码页创建的表在支持该代码页的所有设备上都是可用的，需要特定语言使能器时除外。存取数据库的应用程序负责正确地解释字符数据。

DB2 Everyplace 通过检查当前的代码集或可用的语言环境来检测当前使用的编码格式。

在 Palm OS 上，语言使能器还用来确定代码页。

相关参考:

- 『Java 应用程序中的字符编码』
- 第 321 页的『DB2 Everyplace 语言使能器』
- 第 322 页的『DB2 Everyplace UNICODE 支持』

Java 应用程序中的字符编码

Java 程序以内部方式使用 UNICODE 文本；然而，DB2 Everyplace 表中的字符数据可以是除 UNICODE 以外的格式，取决于创建表时所使用的操作系统和语言。对于 Windows CE 和 Symbian OS 操作系统，DB2 Everyplace JDBC 驱动程序以 UTF-8 格式检索和插入文本。对于其它支持的操作系统，使用系统的缺省字符编码。缺省值通常由 Java 系统属性的 “file.encoding” 属性确定。

例如，在 Win32 操作系统上，用户可以选择使用 CLI 界面的 UNICODE 版本或非 UNICODE 版本；因此，在同一机器上，一个数据库可以具有 UTF-8 格式编码和一个本地代码页编码。要使 JDBC 应用程序能够存取两个数据库中的数据，DB2 Everyplace 为用户提供了一个方法，以便动态指示应用程序应使用哪种数据编码格式。

DB2 Everyplace JDBC 驱动程序根据应用程序指定的格式将 Java 字符串转换为字节。应用程序指定的格式覆盖操作系统的缺省字符编码。

用户可以通过 JDBC 界面动态指定应用程序的数据编码格式。为此:

1. 创建 java.util.Properties 对象。
   - 键: DB2e_ENCODING
   - 值: 字符编码。

   使用值 UTF-8 指定使用 UTF-8 编码的 DB2 Everyplace, 或使用 JVM 所支持的任何字符编码。
2. 使用以下两种方法的其中一种来传送 java.util.properties 对象:
   - 要建立与给定数据库 URL 的连接:

     使用 java.sql 包中的 DriverManager 类中的静态方法 Connection getConnection(String url, Properties info)。
   - 要建立与给定 URL 的数据库连接:

     使用 java.sql 包中的 “接口驱动程序” 类中的 Connection connect(String url, Properties info) 方法。

相关参考:

- 第 319 页的『DB2 Everyplace NLS 支持（根据操作系统）』
- 『DB2 Everyplace 语言使能器』
- 第 322 页的『DB2 Everyplace UNICODE 支持』

DB2 Everyplace 语言使能器

要确保移动式设备可以显示所使用的语言的所有字符, 可以在移动式设备上安装语言使能器。下表列出了可以用于 DB2 Everyplace 的使能器。

表 185. 移动式设备的语言使能器

语言	使能器和操作系统
阿拉伯语	Sakhr Arabic Palm 2.0
简体中文	CWP v1.0 for Palm
繁体中文	• 用于 Palm OS 彩色设备的 CJKOS 3.21（CJK 选项中的排序记录可能会导致意外的结果。） • 仅用于 Pocket PC 的 Gismosoft Chinese Small_Knife 2.0 • Acer S60 具有内置繁体中文 Palm OS
捷克语	• RedGrep GNU-czech0.71 for Palm OS • Sunnysoft InterWrite5.5P Pro for Windows CE
希伯莱语	Penticon Technologies Ltd. Hebrew Support+3.20c for Palm OS
韩国语	• HANME 2.0 for Palm OS • Windows CE 上的 HANTIP 2.01 for Palm OS CessHan for Casio E-115 1.0

相关参考:

- 第 320 页的『Java 应用程序中的字符编码』

- 第 319 页的『DB2 Everyplace NLS 支持（根据操作系统）』
- 『DB2 Everyplace UNICODE 支持』

DB2 Everyplace UNICODE 支持

在支持 UNICODE 的操作系统（Windows CE、Symbian OS、Windows NT 和 Windows 2000）上，DB2 Everyplace 只使用 UNICODE 字符串作为输入／输出字符串。这些 UNICODE 字符串以 UTF–8 格式保存在 DB2 Everyplace 引擎内。UNICODE 字符在经过 UTF–8 转换后可能需要 1 到 3 个字节的存储空间。如果存储在数据库服务器（如 IBM DB2 通用数据库）中的字符串是下载的且存储在 Windows CE 设备上的 DB2 Everyplace 中，则该字符串可能需要更多的空间。

CLI UNICODE 接口注意事项:

- DB2 Everyplace CLI UNICODE 函数在末尾追加有字符 “W”。通过定义宏 UNICODE（在 Windows CE 上这是系统缺省值），常规 CLI 函数会自动映射到相应的 UNICODE 函数。要编写可移植代码，定义宏 “UNICODE” 并让系统执行转换。
- 当启用 UNICODE 支持时，数据类型 SQL_C_CHAR、SQL_C_TCHAR 和 SQL_C_WCHAR 具有相同的含义。
- 许多 CLI 函数将字符串（或缓冲区）长度用作输入／输出参数。
  - 对于*自变量类型*为 SQLCHAR*（对于 W 函数，为 SQLWCHAR*）的函数，长度为字符数。例如:

    ```
    SQLRETURN   SQLExecDirect    (SQLHSTMT          hstmt,
                                  SQLCHAR     FAR   *szSqlStr,
                                  SQLINTEGER        cbSqlStr);
    ```

    UNICODE 字符串 L"ABCD" 是四个字符。
  - 对于*自变量类型*为 SQLPOINTER 的函数，长度为字节数。例如:

    ```
    SQLRETURN  SQLGetData       (SQLHSTMT          hstmt,
                                 SQLUSMALLINT      icol,
                                 SQLSMALLINT       fCType,
                                 SQLPOINTER        rgbValue,
                                 SQLINTEGER        cbValueMax,
                                 SQLINTEGER  FAR   *pcbValue);
    ```

    输入参数 cbValueMax 和输出参数 *pcbValue 的长度都是以字节计的。UNICODE 字符串 L"ABCD" 是八个字节。
- UNICODE 函数还会采用 SQL_NTS 来表示以空结束的字符串。

编写可移植代码的技巧:

- 使用 SQLTCHAR 而不是 SQLCHAR 或 SQLWCHAR。
- 使用 _tcs*XXXX* 函数而不是 str*XXXX*（ANSI）或 wcs*XXXX*（UNICODE）。例如，使用 _tcslen() 而不是 wcslen() 或 strlen()。
- 使用 _TEXT()（或 TEXT()）来将文字串括起来。例如，根据宏定义，_TEXT("ABCD") 可解释为 ANSI 或 UNICODE 字符串。
- 使用 sizeof(ArrayName)/sizeof(TCHAR) 来查明字符数组的大小。

有关示例，请参阅 DB2 Everyplace 附带包括的 Windows CE SampleCLP 样本代码。

相关参考:

- 第 320 页的『Java 应用程序中的字符编码』
- 第 321 页的『DB2 Everyplace 语言使能器』
- 第 319 页的『DB2 Everyplace NLS 支持（根据操作系统）』

第 23 章 DB2 Everyplace 信息集

DB2 Everyplace 库由 HTML 格式的联机帮助和 PDF 及 HTML 格式的书籍组成。本节描述提供的信息以及访问这些信息的方法。

同时，还以在线方式在以下网址处提供了所有产品信息：
www.ibm.com/software/data/db2/everyplace/library.html

DB2 Everyplace PDF 和 HTML 文件

可以从 CD-ROM 直接以 HTML 和 PDF 格式查看 DB2 Everyplace 书籍和“发行说明”。DB2 Everyplace 资料被翻译成多种语言；然而，并未将所有资料都翻译成每一种语言。当没有特定语言版本的资料时，提供了英文版的资料。

当在工作站上安装 DB2 Everyplace 时，文档存储在 `\DB2everyplace\docs` 中。下表列示存储在 **docs** 目录中的书籍。

表 186. 为 DB2 Everyplace 提供的书籍

书名	描述	PDF 文件名	HTML 目录
《DB2 Everyplace 安装与用户指南》（S152–0063–01）	• 将 DB2 Everyplace 组件安装至工作站。 • 将 DB2 Everyplace 数据库和样本应用程序安装至移动式或嵌入式设备。 • 配置和维护移动式设备或嵌入式设备。 • 使用 DB2 Everyplace 样本应用程序。	dsyiug.pdf	dsyiug
《DB2 Everyplace 应用程序开发指南》（S152–0065–01）	• 在可用的平台上构建 DB2 Everyplace 应用程序。 • DB2 Everyplace 样本应用程序和源代码。 • 支持的 SQL 语句、SQLSTATE、DB2 CLI/ODBC、JDBC 方法、IBM Sync Client C-API、IBM Java Sync API 和“本地语言支持”。 • 存取 DB2 Everyplace 数据库 • 使用本地数据加密。	dsyadg.pdf	dsyadg

表 186. 为 DB2 Everyplace 提供的书籍 (续)

书名	描述	PDF 文件名	HTML 目录
《DB2 Everyplace Sync Server 管理指南》（S152–0083–01）	• 配置和维护 Sync Server。 • 将 Sync Server 连接到数据源。 • 配置 Sync Server 与移动及嵌入式设备之间的通信。 • 配置和维护本地和远程数据库。 • 管理用户和数据。	dsysag.pdf	dsysag

DB2 Everyplace 联机文档

“DB2 Everyplace Sync Server 移动式设备管理中心”和 DB2 Everyplace Mobile Application Builder 提供有联机帮助。

第 5 部分 附属资料

声明

IBM 可能并未在所有国家或地区提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可证。您可以用书面方式将许可证查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可证查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: 国际商业机器公司以“按现状”的基础提供本出版物，不附有任何形式的（无论是明示的，还是默示的）保证，包括（但不限于）对非侵权性、适销性和适用于某特定用途的默示保证。某些国家或地区在某些交易中不允许免除明示或默示的保证，因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和／或程序进行改进和／或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其它程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario

M3C 1H7
CANADA

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可证协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其它操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其它可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其它关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本信息可能包含日常商业运作所使用的数据和报告的示例。为了尽可能充分地说明它们，这些示例包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可证:

本信息可能包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。

凡这些实例程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

©（贵公司的名称）（年份）。此部分代码是根据 IBM 公司的样本程序衍生出来的。© Copyright IBM Corp. _输入年份_。All rights reserved.

This product includes software developed by 3Com and its contributors.:

Copyright (c) 1998 3Com/Palm Computing Division. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by 3Com and its contributors.
4. Neither 3Com nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE 3COM AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL 3COM OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

商标

以下各项（可能由星号（*）指示）是国际商业机器公司在美国和／或其它国家或地区的商标:

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2
z/OS

下列各项是其它公司的商标或注册商标:

Microsoft、Windows 和 Windows NT 是 Microsoft Corporation 的商标或注册商标。

Java 或所有基于 Java 的商标和徽标以及 Solaris 是 Sun Microsystems, Inc. 在美国和／或其它国家或地区的商标。

Tivoli 和 NetView 是 Tivoli Systems Inc. 在美国和／或其它国家或地区的商标。

UNIX 是在美国和／或其它国家或地区的注册商标，且经 X/Open Company Limited 唯一许可。

其它公司、产品或服务名称（可能由双星号（**）指示）可能是其它公司的商标或服务标记。

词汇表

〔A〕

按例查询: 一个允许用户动态查看和修改 DB2 Everyplace 表中存储的数据的应用程序。

〔B〕

绑定（bind）: 在 SQL 中，是指将 SQL 预编译程序的输出转换为可使用的结构（称为存取方案）的过程。此过程中，选择数据的存取路径，并执行一些权限检查。

本地数据库（local database）: 一个以物理形式存在于正在使用的计算机上的数据库。对照*远程数据库（remote database）*。

〔C〕

查询（query）: 根据特定条件对数据库信息的请求；例如，对客户表中余额大于 1000 美元的所有客户的列表的请求。

持久性（persistent）: 与跨会话边界维护的数据（通常在非易失性存储器如数据库系统或目录中）相关的内容。

冲突检测（conflict detection）: 在用户应用程序已更新的目标表中检测过时的行的过程。当检测到冲突时，导致冲突的事务被拒绝。

〔D〕

大对象（LOB）（large object（LOB））: 一个字节序列，长度最多可为 2G 字节。它可以是以下三种类型中的任意一种: BLOB（二进制）、CLOB（单字节字符或混合字符）或 DBCLOB（双字节字符）。

单击（tap）: 使用指示笔与手持设备交互作用。

动态主机配置协议（DHCP）（Dynamic Host Configuration Protocol（DHCP））: 一个用于自动配置使用 TCP/IP 的计算机的网际协议。

对象（object）:

1. 任何可以使用 SQL 创建和处理的内容 – 例如，表、视图、索引或包。
2. 在面向对象的设计或编程中，这是由数据以及与该数据相关的操作组成的抽象概念。

〔E〕

二进制大对象（BLOB）（binary large object（BLOB））: 一个字节序列，序列的大小介于 0 到 2G 字节之间。此字节序列没有相关联的代码页和字符集。图像、音频和视频对象都以 BLOB 形式存储。

〔F〕

复制源（replication source）: 一个定义为复制的源的数据库表。在将数据库表定义为复制源之后，该表可以接受复制请求。

复制（replication）: 获取存储在源服务器上的数据库记录或日志中的更改并对目标服务器应用这些更改的过程。

〔G〕

个人数字助理（PDA）（personal digital assistant（PDA））: 一个用于个人组织任务（如管理日历和作笔记）的手持设备，包括电话、传真和联网功能部件。

管理员（MDAC）: 一个允许您创建、编辑和查看同步对象及其相互关系的图形界面。“管理员”还允许您分别查看各客户机的同步状态以及错误消息。

过滤器（filter）: 一个根据指定标准分隔数据、信号或资料的设备或程序。

〔J〕

键（key）: 表、索引或引用约束的描述中标识的列或有序列集。

结构化查询语言（SQL）（Structured Query Language（SQL））: 用来定义和处理关系数据库中的数据的编程语言。

镜像数据库（mirror database）: “Sync 服务器”在内部用来存储同步和复制所需要的数据的数据库。

〔K〕

开放式数据库连接（ODBC）（Open Database Connectivity（ODBC））: 一个允许使用可调用 SQL 访问数据库管理系统的 API，它不需要使用 SQL 预处理器。ODBC 体系结构允许用户添加将应用程序与它们在运行时

选择的数据库管理系统相链接的模块（称为数据库驱动程序）。应用程序无需直接与所有支持的数据库管理系统的模块相链接。

客户机（client）： 一个与数据库服务器通信并访问数据库服务器的程序或用户。您是使用“管理员”定义客户机的。

控制中心（Control Center）： 一个显示数据库对象（如数据库和表）及其相互关系的图形界面。在“控制中心”中，您可以执行“DBA 实用程序”、Visual Explain 和“性能监视器”工具提供的任务。

〔L〕

连接（join）： 允许根据匹配的列值来从两个或多个表检索数据的关系运算。

临时表（temporary table）： 处理 SQL 语句期间创建的用来存放中间结果的表。

〔M〕

目标表（target table）： 一个表，源表中的数据被复制到其中。中间层服务器上的镜像表是目标，移动式设备上的 DB2 Everyplace 表也是目标。

目标数据库（target database）： 一个驻留在移动式设备上的 DB2 Everyplace 数据库，数据从源数据库被复制到其中。

〔P〕

普及计算（PVC）（pervasive computing（PVC））：包括专用工具（称为信息工具）的计算基础结构的使用，利用这些工具，用户可以访问大范围的基于网络的服务（包括通常通过因特网提供的服务）。这些信息工具包括电视、汽车、电话、电冰箱和微波炉。普及计算提供对相关信息的便捷访问以及对该信息进行处理的能力。

〔Q〕

企业服务器（enterprise server）： 请参阅*源服务器（source server）*。

企业数据库（enterprise database）： 请参阅*源数据库（source database）*。

〔R〕

认证（authenticaiton）： 对管理控制数据库中的项验证用户的标识和密码以确保该用户有权使用 Sync Server 来执行数据同步的过程。

日志（log）： 一个包含同步错误消息及其描述的“管理员”对象。

〔S〕

视图（view）： 一个由查询所生成的数据组成的逻辑表。

手持设备（handheld device）： 任何可以手持的计算设备。手持设备包括巴掌大小的 PC 和个人数字助理（PDA）。

授权（authorization）： 在计算机安全性中，授予用户与计算机系统通信及使用计算机系统的权限。

数据过滤器（data filter）： 请参阅*过滤器（filter）*。

数据库服务器（database server）： 为数据库提供数据库服务的功能部件。

数据库管理系统（DBMS）（database management system（DBMS））： 一个通过提供下列服务来对数据进行管理的计算机程序: 集中控制、数据独立性、以及用于高效存取、完整性、恢复、并行控制、保密性和安全性的复杂物理结构。

数据同步（data synchronization）： 请参阅*移动数据同步（mobile data synchronization）*。

刷新（refresh）： 一个将用户表中所有感兴趣的数据复制到目标表并替换现有数据的过程。

双字节字符集（DBCS）（double-byte character set（DBCS））： 一组字符，其中每个字符由两个字节表示。

〔T〕

特权（privilege）： 以特定方式访问特定数据库对象的权限。这些权限由具有 SYSADM （系统管理员）权限或 DBADM（数据库管理员）权限的用户或由对象的创建者控制。特权包括诸如在表中创建、删除和选择数据的权限。

同步对象（synchronization object）： “管理员”中的一个可管理项，它包含关于您的组织中的同步过程的各个方面。有 5 种类型的同步对象: 组、客户机、预订集、预订和日志。

同步会话（synchronization session）： 一个事务，在此事务中，移动用户（或*客户机（client）*）提交他们对源数据的本地副本所作的更改，并接收他们自上次同步后对源数据（驻留在远程服务器上）所作的所有更改。

同步（synchronization）： *请参阅*移动数据同步。

〔W〕

无线 LAN（wireless LAN）： 在无线使用中，移动用户可以通过无线电连接与局域网（LAN）相连接。用于 LAN 连接的无线技术包括高速频谱、微波和红外线。

〔Y〕

移动数据同步（mobile data synchronization）： 一个由两个步骤组成的过程，其中移动用户（或客户机（*client*））提交他们对源数据的本地副本所作的更改，并接收自从上次同步之后对源数据（在远程数据库中）所作的所有更改。

移动（mobile）： 与在各种位置之间频繁移动并使用不同类型网络连接（如拨号、LAN 或无线连接）的用户在便携式计算机或手持设备上执行的计算有关的内容。

预订集（subscription set）： 包含复制预订的“管理员”对象。为了向组成员提供对复制预订中定义的数据和文件的存取权，创建预定集并对其指定预订，然后将该预定集给一个组。预订集对象替换应用程序对象。

预订（subscription）： 一个有关如何将源数据库中的信息复制到目标数据库的规范。预订允许您定义可以从源数据库复制哪些数据和文件的子集。您可以创建两种类型的预订：对存储在源服务器上文件的文件预订，及对在源数据库中表的表预订。

源表（source table）： 包含要复制到目标表的数据的表。源表必须是复制源表。请对照*目标表（target table）*。

源服务器（source server）： 复制源的数据库位置。

源数据库（source database）： 驻留在包含要复制到目标系统的数据的源服务器上的数据库。

远程访问服务（RAS）（Remote Access Service（RAS））： 一个管理两个系统之间的连接的 Windows 程序。

远程数据库（remote database）： 物理上存在于非当前使用中的计算机上的数据库。对照本地数据库。远程计算设备可以是固定和不可移动的，也可以是便携式的。

〔Z〕

中间层系统（mid-tier system）： 在其上安装了 DB2 Everyplace Sync Server 的机器。在双层同步配置中，中间层系统和源系统指的是同一台机器。

主键（primary key）： 作为表定义一部分的唯一键。主键是引用约束定义的缺省父键。对于 DB2 Everyplace Sync Server V7，每个复制源都必须有且只能有一个主键。

主数据库（master database）： 请参阅*源数据库（source database）*。

组（group）： 具有相似移动数据同步需求的客户机集合。对每个组定义同步特征，如组中的用户需要访问哪些应用程序才能执行他们的作业以及他们需要访问的企业数据子集。

A

Apply 限定符（Apply qualifier）： 一个字符串，它标识对 DataPropagator Apply 程序的每个实例唯一的预订定义。

B

BLOB： 请参阅*二进制大对象（binary large object）*。

D

DB2 控制中心（DB2 Control Center）： 请参阅*控制中心（Control Center）*。

DB2 DataPropagator： 一个提供将数据从源复制到目标的自动方法的复制产品。移动数据同步期间，镜像和远程数据库既作为源又作为目标。DataPropagator 将客户机的更改从镜像复制到远程数据库，并且也将更改从远程数据库复制到镜像数据库。

DBCS： 请参阅*双字节字符集（double-byte character set）*。

DHCP： 请参阅*动态主机配置协议（Dynamic Host Configuration Protocol）*。

DPROP： 请参阅 *DB2 DataPropagator*。

I

IBM Sync： 表示 DB2 Everyplace Sync Server 软件的客户机组件的图标名。

L

LOB： 请参阅*大对象（large object）*。

M

MDAC： 请参阅*管理员*。

O

ODBC: 请参阅*开放式数据库连接（Open Database Connectivity）*。

P

PDA: 请参阅*个人数字助理（personal digital assistant）*。

PVC: 请参阅*普及计算（pervasive computing）*。

Q

QBE: 请参阅*按例查询（Query-by-Example）*。

R

RAS: 请参阅*远程访问服务（Remote Access Service）*。

S

SQL: 请参阅*结构化查询语言（Structured Query Language）*。

索引

[K]

[L]

[M]

[Q]

[S]

[T]

[W]

[X]

[Y]

[Z]

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

[特别字符]

与 IBM 联系

要了解 DB2 Everyplace 产品的信息或订购任何 DB2 Everyplace 产品，请与当地的 IBM 分部的 IBM 代表联系，或与任何 IBM 已授权的软件分销商联系。

如果您住在美国，可以拨打下面其中一个电话号码:

- 1-800-237-5511，用于客户支持
- 1-888-426-4343，用于了解可用的服务选项

产品信息

如果您住在美国，可以拨打下面其中一个电话号码:

- 1-800-IBM-CALL (1-800-426-2255) 或 1-800-3IBM-OS2 (1-800-342-6672)，用于订购产品或获取概要信息。
- 1-800-879-2755，用于订购出版物。

http://www.ibm.com/software/data/db2/everyplace/

DB2 Everyplace 万维网页面提供关于新闻、产品描述和培训安排等内容的 DB2 Everyplace 最新信息。

http://www.ibm.com/software/data/db2/everyplace/library.html

DB2 Everyplace Technical Library 提供对常见问题、修订、书籍和 DB2 Everyplace 最新技术信息的访问。

注: 此信息可能只有英文版。

http://www.ibm.com/software/data/

DB2 万维网页面提供了有关新闻、产品描述、培训安排及其它项目的最新 DB2 信息。

http://www.ibm.com/software/data/db2/library/

DB2 Product and Service Technical Library 提供对常见问题、修订、书籍和 DB2 最新技术信息的访问。

注: 此信息可能只有英文版。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

International Publications ordering Web 站点提供有关如何订购书籍的信息。

http://www.ibm.com/education/certify/

Professional Certification Program from the IBM Web 站点提供各种 IBM 产品（包括 DB2）的验证测试信息。

ftp.software.ibm.com

以匿名形式登录。在 `/ps/products/db2` 目录中，您可以找到与 DB2 以及许多其它产品相关的演示、修订、信息和工具。

comp.databases.ibm-db2, bit.listserv.db2-l

这些因特网新闻组供用户讨论他们使用 DB2 产品的经验。

On Compuserve: GO IBMDB2

输入此命令可访问 IBM DB2 Family 论坛。这些论坛支持所有 DB2 产品。

有关如何在美国以外的地区与 IBM 联系的信息，参阅 *IBM Software Support Handbook* 的 Appendix A。要访问此文档，请访问以下 Web 页面：http://www.ibm.com/support/，然后选择页面底部附近的 IBM Software Support Handbook 链接。

注： 在某些国家或地区，IBM 已授权的分销商应与他们的分销商支持机构联系，而不是与“IBM 支持中心”联系。

程序号: 5724–D04

中国印刷

S152-0065-01