IBM DB2 10.1
for Linux, UNIX, and Windows

开发 ADO.NET 和 OLE DB 应用程序

S151-1765-00

IBM DB2 10.1
for Linux, UNIX, and Windows

开发 ADO.NET 和 OLE DB 应用程序

S151-1765-00

注意

使用此信息及其支持的产品前，请先阅读第 163 页的附录 B，『声明』下的常规信息。

修订版声明

此文档包含 IBM 的所有权信息。它在许可协议中提供，且受版权法的保护。本出版物中包含的信息不包括对任何产品的保证，且提供的任何语句都不需要如此解释。

您可在线或通过当地的 IBM 代表处订购 IBM 出版物。

- 要在线订购出版物，请转至 IBM 出版物中心，网址为: http://www.ibm.com/shop/publications/order
- 要查找当地的 IBM 代表处，请转至 IBM 全球联系人目录，网址为: http://www.ibm.com/planetwide/

要从美国或加拿大的 DB2 市场和销售部订购 DB2 出版物，请致电 1-800-IBM-4YOU（426-4968）。

您发送信息给 IBM 后，即授予 IBM 非独占权限，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。

目录

第 1 章 ADO.NET 应用程序开发

最近几年，Microsoft 一直致力推广一款新的 Windows 软件开发平台，即 .NET Framework。.NET Framework 是 Microsoft 对组件对象模型 (COM) 技术的替代。.NET Framework 的关键功能如下:

- 您可以使用 40 种以上不同的编程语言来编码 .NET 应用程序。进行 .NET 开发时，最流行的语言是 C# 和 Visual Basic .NET。
- .NET Framework 类库提供了用于构建 .NET 应用程序的构建块。这个类库与语言无关，它提供操作系统和应用程序服务的接口。
- 无论使用哪种语言，.NET 应用程序都将编译成中间语言 (IL)，这是一种字节码。
- 公共语言运行时 (CLR) 是 .NET Framework 的核心，它能够即时编译 IL 代码并接着运行该代码。在运行经过编译的 IL 代码时，CLR 将激活对象、验证其是否不存在安全性问题、为其分配内存、执行这些对象并在执行完成后清除其内存。

借助这些功能，.NET Framework 有利于各种应用程序实现（例如 Windows 表单、Web 表单和 Web Service）、快速应用程序开发以及安全应用程序部署。COM 和 COM+ 已被证实无法实现或难以实现上述所有功能。

.NET Framework 通过 ADO.NET 提供了广泛的数据访问支持。ADO.NET 既支持连接式访问也支持断开连接式访问。在 ADO.NET 中，断开连接式数据访问的关键组件是 DataSet 类，它的实例充当驻留在应用程序内存中的数据库高速缓存。

进行连接式访问和断开连接式访问时，应用程序都通过数据提供程序来使用数据库。各种数据库产品都包括它们自己的 .NET 数据提供程序，其中包括 DB2® for Windows。

.NET 数据提供程序提供了下列基本类的实现:

- Connection: 建立并管理数据库连接。
- Command: 对数据库执行 SQL 语句。
- DataReader: 从数据库读取结果集数据并返回该数据。
- DataAdapter: 将 DataSet 实例与数据库链接。通过 DataAdapter 实例，DataSet 可以读写数据库表数据。

Microsoft 提供了两个数据提供程序，即: OLE DB .NET Data Provider 和 ODBC .NET Data Provider。OLE DB .NET Data Provider 是通过 COM 互操作模块向 IBM® OLE DB Provider 反馈 ADO.NET 请求的桥接提供程序。ODBC .NET Data Provider 是向 IBM ODBC Driver 反馈 ADO.NET 请求的桥接提供程序。建议您不要使用这些 .NET 数据提供程序来访问 DB2 系列数据库。IBM Data Server Provider for .NET 是高性能的受管 ADO.NET 数据提供程序。建议您将此 .NET 数据提供程序与 DB2 系列数据库配合使用。使用 IBM Data Server Provider for .NET 进行 ADO.NET 数据库访问时，限制较少，并且性能显著优于 OLE DB 和 ODBC .NET 桥接提供程序。

部署 .NET 应用程序 (Windows)

为了简化 .NET 应用程序部署工作，IBM 提供了 IBM Data Server Driver Package，这是一个适用于大规模部署方案的小规模客户机。如果相对于 IBM Data Server Driver Package 您更需要使用 IBM Data Server Runtime Client 的附加功能，那么可以改为使用该客户机。

开始之前

- 在执行部署之前，必须通过 Visual Studio 或命令行来构建 .NET 应用程序。有关构建 .NET 应用程序的更多信息，请参阅相关任务。
- 用于构建 .NET 应用程序的计算机以及将要部署 .NET 应用程序的计算机除了运行『受支持的 .NET 开发软件』所述的其他软件以外，还必须运行受支持的 Windows 操作系统版本:
  - 构建系统
    - Windows 操作系统
    - Visual Studio
    - .NET Framework Redistributable Package
    - .NET Framework Software Development Kit
  - 部署系统
    - Windows 操作系统
    - .NET Framework Redistributable Package

过程

要部署 .NET 应用程序，请执行下列操作:

1. 在将要部署应用程序的计算机上安装 IBM Data Server Driver Package。在安装期间，请将 IBM Data Server Driver Package 安装版本设置为缺省数据库客户机接口副本。

   注: 任何面向 IBM 数据服务器运行的现有数据库应用程序都将使用这个新的 IBM Data Server Driver Package 安装版本。在将所部署的 .NET 应用程序投入使用之前，请针对新驱动程序来测试那些应用程序。
2. 在将要运行应用程序的计算机上安装所构建的应用程序。

受支持的 .NET 开发软件

要开发和部署面向 IBM 数据服务器运行的 .NET 应用程序，您将需要使用受支持的开发软件和操作系统。

支持开发和部署 .NET Framework 2.0、3.0、3.5 和 4.0 应用程序的操作系统

- Windows XP Service Pack 2（32 位和 64 位版本）
- Windows Server 2003（32 位和 64 位版本）
- Windows Vista（32 位和 64 位版本）
- Windows Server 2008（32 位和 64 位版本）
- Windows Server 2008 R2（64 位版本）
- Windows 7（32 位和 64 位版本）

支持 .NET Framework 应用程序的开发软件

除 IBM 数据服务器客户机或驱动程序包之外，您还需要下列其中一个受支持工具来开发 .NET Framework 应用程序。

- Visual Studio 2008
- Visual Studio 2010

支持 .NET Framework 应用程序的部署软件

除 IBM 数据服务器客户机或驱动程序包之外，您还需要下列其中一个受支持程序包来部署 .NET Framework 应用程序。在大多数情况下，Windows 安装将附带提供其中的一个选项。

- .NET Framework V2.0 Redistributable Package
- .NET Framework V3.0 Redistributable Package
- .NET Framework V3.5 Redistributable Package
- .NET Framework V4.0 Redistributable Package

在 Visual Studio 中集成 DB2

IBM Database Add-Ins for Visual Studio 是一组功能部件，这些功能部件无缝地集成到 Visual Studio 开发环境中，以使您能够使用 DB2 服务器以及开发 DB2 过程、函数和对象。

IBM Database Add-Ins for Visual Studio 针对 DB2 数据库提供一个简单接口。例如，可以使用设计器和向导来创建数据库对象，而不必使用 SQL。对于您需要编写 SQL 代码的情况，集成式 DB2 SQL 编辑器提供了下列功能:

- 彩色 SQL 文本，旨在提高可读性
- 与 Microsoft Visual Studio IntelliSense 功能部件集成，这在您输入 DB2 脚本时支持智能自动补全

使用 IBM Database Add-Ins for Visual Studio，您可以完成下列任务:

- 打开各种 DB2 开发和管理工具。
- 在解决方案资源管理器中创建和管理 DB2 项目。
- 通过服务器资源管理器访问和管理 DB2 数据连接。
- 创建和修改 DB2 脚本，其中包括用于创建存储过程、函数、表、视图、索引和触发器的脚本。

Visual Studio 2008 和 2010

IBM Database Add-Ins for Visual Studio 作为可单独安装的组件随 DB2 客户机和 DB2 服务器一起提供。安装 DB2 产品完成后，您可以选择安装 IBM Database Add-Ins for Visual Studio。如果计算机上未安装 Visual Studio，那么将不会安装此加载件。在安装 Visual Studio 之后，您随时可以从 DB2 产品的设置菜单中安装此加载件。

有关使用 IBM Database Add-Ins 和 数据服务器 .NET 提供程序 进行快速应用程序开发的更多详细信息，请访问 IBM Information Management and Visual Studio .NET 专区，网址为: http://www.ibm.com/developerworks/data/zones/vstudio/index.html。

第 2 章 外部例程

外部例程是一些使用数据库服务器的文件系统中，位于数据库外部的编程语言应用程序来实现其逻辑的例程。

通过在例程的 CREATE 语句中指定 EXTERNAL 子句来声明例程与外部代码应用程序的关联。

您可以创建外部过程、外部函数以及外部方法。虽然它们全都可以使用外部编程语言来实现，但是每种例程功能类型都具有不同的功能。在决定实现外部例程前，必须先通过阅读主题“外部例程概述”来了解何谓外部例程、外部例程的实现和使用方式。在了解这些信息后，您可以在相关链接所指向的主题中了解外部例程的更多信息，以就何时以及如何在数据库环境中使用外部例程作出有见识的决策。

使用例程的益处

使用例程有下列益处:

封装可以从 SQL 接口调用的应用程序逻辑

在包含许多具有公共要求的不同客户机应用程序的环境中，有效地使用例程有助于简化代码复用、代码标准化和代码维护。如果需要在使用了例程的环境中更改公共应用程序行为的特定方面，那么只需修改封装了该行为的例程。如果未使用例程，就必须在每个应用程序中更改应用程序逻辑。

启用对其他数据库对象的受控访问

可以使用例程来控制对数据库对象进行的访问。用户可能无权一般化地发出特定 SQL 语句（例如 CREATE TABLE）；但是，您可以授权该用户调用包含该语句的一个或多个特定实现的例程，从而通过封装特权简化特权管理工作。

通过降低网络流量来提高应用程序性能

在客户机上运行应用程序时，将把每个 SQL 语句单独地从客户机发送到数据库服务器以便执行，并且将单独地返回每个结果集。这可能会导致网络流量过高。如果能够确定需要进行大量数据库交互但不需要进行很多用户交互的工作，那么最好将此部分工作安装在服务器上，以便最大程度地降低网络流量以及在功能更强大的数据库服务器上完成该工作。

能够更快更高效地执行 SQL

因为例程是数据库对象，所以它们在传输 SQL 请求和数据方面比客户机应用程序更高效。因此，在例程中执行 SQL 语句的效果比在客户机应用程序中执行这些语句更好。如果创建例程时指定了 NOT FENCED 子句，那么这些例程将在数据库管理器所在的进程中运行，并因此可以使用共享内存进行通信，这有助于提高应用程序性能。

允许使用不同编程语言实现的逻辑进行互操作

由于代码模块可能由不同程序员使用不同编程语言实现，并且通常最好尽可能地复用代码，因此 DB2 例程设计成支持较高的互操作性水平。

- 使用一种编程语言编写的客户机应用程序可以调用使用另一编程语言实现的例程。例如，C 客户机应用程序可以调用 .NET 公共语言运行时例程。

- 例程可以调用其他例程，而与例程类型或例程实现无关。例如，Java 过程可以调用嵌入式 SQL 标量函数。
- 在运行于一款操作系统上的 DB2 客户机中，可以调用在运行于另一款操作系统上的数据库服务器中创建的例程。

这些益处仅仅是使用例程可以实现的众多益处的一部分。使用例程可以使各种用户受益，这些用户包括数据库管理员、数据库架构设计师和数据库应用程序开发者。因此，您可能想对例程的许多非常有用的应用领域进行探查。

您可以通过各种类型的例程来满足特定功能需求和各种例程实现。对例程类型和实现所作的选择可影响以前列出的益处的表现水平。通常，例程是一种功能强大的逻辑封装方法，使您能够扩展 SQL、改善结构和维护以及有可能提高应用程序性能。

外部例程实现

外部例程实现是指，例程逻辑由驻留在数据库外部的编程语言代码定义。与其他例程实现相同，您通过执行 CREATE 语句在数据库中创建具有外部实现的例程。

例程逻辑存储在经过编译的库中，该库驻留在数据库服务器上的特殊目录路径中。例程名称与外部代码应用程序之间的关联由 CREATE 语句中指定的 EXTERNAL 子句推断。

可使用任何受支持外部例程编程语言来编写外部例程。请参阅『用于开发外部例程的受支持 API 和编程语言』。

外部例程实现可能比 SQL 例程实现略为复杂。但是，其功能极为强大，这是因为它们允许您充分利用所选实现编程语言的功能和性能。外部函数还具有能够访问和处理驻留在数据库外部的实体（例如网络或文件系统）这一优点。对于只需要与 DB2 数据库进行少量交互但必须包含大量逻辑或非常复杂的逻辑的例程而言，外部例程实现是良好的选择。

例如，外部例程适合用于实现对内置数据类型执行操作以及提高其利用率的新函数，例如对 VARCHAR 数据类型执行操作的新字符串函数或者对 DOUBLE 数据类型执行操作的复杂数学函数。外部例程实现还适合于实现可能涉及外部操作（例如发送电子邮件）的逻辑。

如果您已熟悉使用其中一种受支持的外部例程编程语言进行编程，并且需要封装更着重于编程逻辑而非数据访问的逻辑，那么在了解创建具有外部实现的例程所涉及的步骤之后，您很快就会发现它们的巨大潜力。

用于开发外部例程的受支持 API 和编程语言

您可以使用下列 API 及相关联的编程语言来开发 DB2 外部例程（过程和函数）：

- ADO.NET
  - .NET 公共语言运行时编程语言
- CLI
- 嵌入式 SQL
  - C
  - C++

  - COBOL（只支持过程）
- JDBC
  - Java
- OLE
  - Visual Basic
  - Visual C++
  - 任何其他支持此 API 的编程语言。
- OLE DB（只支持表函数）
  - 任何支持此 API 的编程语言。
- SQLJ
  - Java

对支持用于开发外部例程的 API 和编程语言的比较

在开始实现外部例程之前，必须考虑各种支持的外部例程应用程序编程接口 (API) 和编程语言的特征与限制。这将确保您从开始就选择正确的实现，以及您所需的例程功能是可用的。

表 1. 外部例程 API 和编程语言比较

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
SQL（包括 SQL PL）	• SQL 是高级语言，易于学习和使用，使实现能快速进行。 • SQL 过程语言 (SQL PL) 元素允许将控制流逻辑用于 SQL 操作和查询。	• 非常好。 • SQL 例程的性能高于 Java 例程。 • SQL 例程的性能相当于使用 NOT FENCED 子句来创建的 C 和 C++ 外部例程。	• 非常安全。 • SQL 过程始终与数据库管理器在同一内存中运行。这对应于缺省情况下使用关键字 NOT FENCED 来创建的例程。	• 高度可伸缩。	• 无法访问数据库服务器文件系统。 • 无法调用位于数据库外部的应用程序。

表 1. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
嵌入式 SQL（包括 C 和 C++）	• 低级但功能强大的编程语言。	• 非常好。 • C 和 C++ 例程的性能高于 Java 例程。 • 使用 NOT FENCED 子句创建的 C 和 C++ 例程的性能相当于 SQL 例程。	• C 和 C++ 例程容易发生编程错误。 • 程序员必须精通 C，才能避免犯一些常见的内存和指针操作错误，此类错误会使例程实现更冗长、更耗时。 • 应该使用 FENCED 子句和 NOT THREADSAFE 子句来创建 C 和 C++ 例程，以避免在运行时例程发生异常的情况下，损坏数据库管理器。这些是缺省子句。使用这些子句会对性能造成一定的负面影响，但是可确保安全执行。请参阅例程安全性。	• 使用 FENCED 和 NOT THREADSAFE 子句来创建 C 和 C++ 例程时，会降低可伸缩性。在数据库管理器进程之外的单独 **db2fmp** 进程中运行这些例程。每个并发执行的例程都需要一个 **db2fmp** 进程。	• 多种支持的参数传递样式会造成混淆。用户应该尽可能地使用参数样式 SQL。
嵌入式 SQL（COBOL）	• 高级编程语言适合于开发业务应用程序（通常面向文件）。 • 过去广泛用于生产业务应用程序，虽然其普及面现正在缩小。 • COBOL 不包含指针支持，是线性迭代编程语言。	• COBOL 例程的性能低于使用任何其他外部例程实现选项来创建的例程。	• 目前未提供任何信息。	• 目前未提供任何信息。	• 您可以在 64 位 DB2 实例中创建和调用 32 位 COBOL 过程实例，但是这些例程的性能低于 64 位 DB2 实例中 64 位 COBOL 过程。

表 1. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
JDBC（Java）和 SQLJ（Java）	• 高级面向对象程序设计语言，适合于开发独立应用程序、applet 以及 servlet。 • Java 对象和数据类型可方便建立数据库连接、执行 SQL 语句以及操作数据。	• Java 例程的性能低于 C 和 C++ 例程或 SQL 例程。	• Java 例程较之 C 和 C++ 例程安全，因为是由 Java 虚拟机（JVM）处理对危险操作的控制。这将提高可靠性，让一个 Java 例程的代码很难损害正在同一进程中运行的另一个例程。	• 良好的可伸缩性 • 使用 FENCED THREADSAFE 子句（缺省子句）创建的 Java 例程具有良好的可伸缩性。所有 FENCED Java 例程将共享几个 JVM。如果特定 db2fmp 进程的 Java 堆接近耗竭，那么系统上可能正在使用多个 JVM。	• 只有不允许从 Java 例程执行 Java 本机接口（JNI）调用，才能避免潜在的危险操作。
.NET 公共语言运行时支持的语言（包括 C#、Visual Basic 以及其他）	• Microsoft .NET 受管代码模型的一部分。 • 会将源代码编译为可由 Microsoft .NET Framework 公共语言运行时进行解释的中间语言（IL）字节码。 • 可以从子组合件（从不同的 .NET 编程语言源代码进行编译）构建 CLR 组合件，这允许用户复用和集成使用各种语言编写的代码模块。	• 只能使用 FENCED NOT THREADSAFE 子句来创建 CLR 例程，才能将运行时中断数据库管理器的可能性降至最小。这会对性能造成一定的负面影响 • 使用缺省子句值可将运行时中断数据库管理器的可能性降至最小；但是，因为 CLR 例程必须以 FENCED 方式运行，所以它们的性能可能稍小于可以指定为以 NOT FENCED 方式来运行的外部例程。	• 只能使用 FENCED NOT THREADSAFE 子句来创建 CLR 例程。如此才能让 CLR 例程安全运行，因为它们将在数据库管理器外部的单独 **db2fmp** 进程中运行。	• 未提供任何信息。	• 请参阅“.NET CLR 例程限制”主题。

表 1. 外部例程 API 和编程语言比较 (续)

API 和编程语言	功能支持	性能	安全性	可伸缩性	限制
• OLE	• 可以使用 Visual C++、Visual Basic 以及其他受 OLE 支持的语言来实现 OLE 例程。	• OLE 自动化例程的速度取决于用来实现这些例程的语言。通常，它们的速度要慢于非 OLE C/C++ 例程。 • OLE 例程只能以 FENCED NOT THREADSAFE 方式运行，因此 OLE 自动化例程没有良好的可伸缩性。	• 未提供任何信息。	• 未提供任何信息。	• 未提供任何信息。
• OLE DB	• 可以使用 OLE DB 来创建用户定义的表函数。 • OLE DB 函数连接至外部 OLE DB 数据源。	• OLE DB 函数的性能取决于 OLE DB 提供者，但是，通常 OLE DB 函数的性能高于逻辑上等价的 Java 函数，但低于逻辑上等价的 C、C++ 或 SQL 函数。但是，查询（在其中调用了函数）中的某些谓词可能在 OLE DB 提供者处进行求值，因此减少了 DB2 数据库系统必须处理的行数，这常常可以改进性能。	• 未提供任何信息。	• 未提供任何信息。	• OLE DB 只能用来创建用户定义的表函数。

外部例程功能

外部例程提供大多数公共例程功能以及 SQL 例程所不支持的附加功能。

外部例程的独特功能如下所示:

访问驻留在数据库外部的文件、数据和应用程序

外部例程可以访问和处理驻留在数据库本身外部的数据或文件。它们还可以调用驻留在数据库外部的应用程序。例如，数据、文件或应用程序可以驻留在数据库服务器文件系统或可用的网络中。

各种外部例程参数样式选项

可以使用选择的参数样式以编程语言实现外部例程。虽然所选编程语言可能有

首选的参数样式，但有时也有选项供您选择。某些参数样式支持通过名为 dbinfo 的结构与例程传递附加的数据库和例程属性信息，此结构在例程逻辑中可能非常有用。

通过暂存区在外部函数调用之间保存状态

外部用户定义的函数支持为一组值在函数调用之间保存状态。此任务通过名称 scratchpad 的结构完成。这对于返回聚集值的函数以及需要初始设置逻辑（例如初始化缓冲区）的函数而言特别有用。

调用类型标识各个外部函数调用

将针对一组值多次调用外部用户定义的函数。每次调用都由可以在函数逻辑中引用的调用类型值标识。例如，对于函数的第一次调用、数据访存调用和最终调用都由特殊的调用类型。由于特定的逻辑可以与特定的调用类型相关联，因此调用类型非常有用。

外部标量函数

外部标量函数是一些使用外部编程语言来实现其逻辑的标量函数。

可以开发并使用这些函数来扩展现有 SQL 函数的集合，并且这些函数的调用方式与 DB2 内置函数（例如 LENGTH 和 COUNT）的调用方式相同。即，可以在 SQL 语句中表达式有效的任何位置引用这些函数。

可以在 DB2 数据库服务器上执行外部标量函数逻辑，然而，与内置或用户定义 SQL 标量函数不同的是，外部函数的逻辑可以访问数据库服务器文件系统、执行系统调用或访问网络。

外部标量函数可以读取 SQL 数据，但无法修改 SQL 数据。

可以针对函数的单一引用重复调用外部标量函数，并且外部标量函数可以使用暂存区（内存缓冲区）在这些调用之间维护状态。如果函数需要一些初始但开销很大的设置逻辑，那么此方法的功能很强大。可以在第一次调用时，使用暂存区来存储一些可供标量函数的后续调用访问或更新的值，以完成设置逻辑。

外部标量函数的功能

- 可以在 SQL 语句中支持表达式的任何位置，将外部标量函数作为 SQL 语句的一部分来引用。
- 可直接通过调用 SQL 语句来使用标量函数的输出。
- 对于用户定义的外部标量函数，可以使用暂存区在函数迭代调用之间维护状态。
- 在谓词中使用时可以提供性能优势，因为它们是在服务器上执行。如果可以将函数应用于服务器上的候选行，那么该函数通常不考虑此行，就将其传输至客户机，从而减少必须从服务器传递客户机的数据量。

限制

- 无法在标量函数中执行事务管理。即，无法在标量函数中发出 COMMIT 或 ROLLBACK。
- 无法返回结果集。
- 标量函数可以根据输入集来返回单一标量值。
- 外部标量函数无法用于单一调用。这样的设计是针对函数的单一引用以及给定的输入集，根据输入调用函数一次，并返回单一标量值。在第一次调用

时，标量函数可设计为执行一些设置工作，或存储一些可以在后续调用中访问的信息。SQL 标量函数更适合于需要单一调用的功能。

- 在单一分区数据库中，外部标量函数可以包含 SQL 语句。这些语句可以从表中读取数据，但无法修改表中的数据。如果数据库具有多个分区，那么外部标量函数中不能具有 SQL 语句。SQL 标量函数可以包含读取或修改数据的 SQL 语句。

常见用法

- 扩展 DB2 内置函数集。
- 在 SQL 语句中执行 SQL 无法以本机方式执行的逻辑。
- 在 SQL 语句中封装通常作为子查询来复用的标量查询。例如，给定邮政编码，在表中搜索可找到该邮政编码的城市。

支持的语言

- C
- C++
- Java
- OLE
- .NET 公共语言运行时语言

注:

1. 用于创建聚集函数的功能是受限的。这些函数又称为列函数，可接收一组类似值（数据列）并返回单一答案。只有在以内置聚集函数作为源时，才能创建用户定义的聚集函数。例如，如果存在定义了基本类型 INTEGER 的单值类型 SHOESIZE，那么可以将函数 AVG(SHOESIZE) 定义为以现有内置聚集函数 AVG(INTEGER) 作为源的聚集函数。
2. 您也可以创建返回行的函数。这些函数称为行函数，且只能用作结构化类型的变换函数。行函数的输出是单一行。

外部标量函数和方法处理模型

定义了 FINAL CALL 规范的方法和标量 UDF 的处理模型如下所示:

FIRST 调用

这是 NORMAL 调用的特殊情况，标识为 FIRST 的目的是使函数能够执行任何初始处理。将对自变量进行求值，并将结果传递至函数。正常情况下，此函数对于此调用将返回一个值，但也可能返回错误，在出错情况下，将不会执行 NORMAL 或 FINAL 调用。如果 FIRST 调用返回了错误，那么方法或 UDF 在返回前必须执行清理工作，这是因为，将不会执行 FINAL 调用。

NORMAL 调用

这些是对函数进行的第二次直至倒数第二次调用，这由语句的数据和逻辑指定。对于每次 NORMAL 调用，期望函数在对自变量进行求值和传递后返回一个值。如果 NORMAL 调用返回了错误，那么将不会执行进一步的 NORMAL 调用，但将执行 FINAL 调用。

FINAL 调用

这是在执行语句结束处理时（或关闭游标时）进行的特殊调用，进行此调用的

条件是 FIRST 调用成功。执行 FINAL 调用时，不会传递任何自变量值。会作出此调用以便函数可以清除任何资源。在执行此调用时，函数不会返回值，但可能会返回错误。

对于未使用 FINAL CALL 定义的方法或标量 UDF，将只对函数进行 NORMAL 调用，这通常会对每次调用返回一个值。如果 NORMAL 调用返回了错误，或者语句遇到另一个错误，那么不会对该函数进行其他调用。

注: 此模型描述方法和标量 UDF 的普通错误处理。在发生系统故障或通信问题时，无法进行此错误处理模型所指示的调用。例如，对于 FENCED UDF，如果 `db2udf` 受保护进程不知何故提前终止，那么 DB2 无法执行所指示的调用。

外部表函数

用户定义的表函数将表传递至引用该表的 SQL。

表 UDF 引用只有在 SELECT 语句的 FROM 子句才有效。使用表函数时，请遵循下列事项:

- 即使表函数传递表，在 DB2 数据库系统和 UDF 之间的物理接口也是一次一行。会对表函数发出五种调用: OPEN、FETCH、CLOSE、FIRST 以及 FINAL。是否存在 FIRST 和 FINAL 调用取决于您如何定义 UDF。可用于标量函数的同一*调用类型*机制也可用来区分这些调用。
- 表函数的 CREATE FUNCTION 语句的 RETURNS 子句中所定义的每个结果列不一定都返回。CREATE FUNCTION 的 DBINFO 关键字以及对应的 dbinfo 自变量会启用优化，这样就只需返回特定表函数引用所需的那些列。
- 所返回的各个列值在格式上符合标量函数所返回的值。
- 表函数的 CREATE FUNCTION 语句具有 CARDINALITY 规范。此规范使定义者能够通知 DB2 优化器大致的结果大小，以便优化器可以作出更好的决策来确定何时引用函数。

  不论将什么内容指定为表函数的 CARDINALITY，在编写具有无限基数的函数（即，只要执行 FETCH 调用，函数总是会返回一行）时，必须格外小心。在许多情况下，DB2 数据库系统期望表末尾条件作为其查询处理中的触媒。使用 GROUP BY 或 ORDER BY 就是这样的示例。DB2 数据库系统无法构成聚集组（除非达到表末尾）且无法排序（除非具有所有数据）。因此，如果将永不返回表末尾条件 (SQL-state value '02000') 的表函数与 GROUP BY 或 ORDER BY 子句搭配使用，那么会导致无限处理循环。

外部表函数处理模型

定义了 FINAL CALL 规范的表 UDF 的处理模型如下所示:

FIRST 调用

在执行第一个 OPEN 调用之前发出此调用，其目的是使函数能够执行任何初始处理。在此调用之前清除暂存区。将对自变量进行求值，并将结果传递至函数。此函数不会返回行。如果此函数返回错误，那么不对此函数作出进一步调用。

OPEN 调用

作出此调用以使函数能够执行特定于扫描的特殊 OPEN 处理。不会在调用之前清除暂存区（如果存在）。会对自变量进行求值，并传递结果。在执行 OPEN

调用时，此函数不会返回行。如果函数从 OPEN 调用返回错误，那么不会作出任何 FETCH 或 CLOSE 调用，但仍会在语句结束时作出 FINAL 调用。

FETCH 调用

继续作出 FETCH 调用，除非函数返回指示表结束的 SQLSTATE 值。UDF 就是在这些调用上发起并返回数据行。可以将自变量值传递至函数，但这些值指向执行 OPEN 调用时所传递的值。因此，自变量值可能不是当前值，不可靠。如果确实需要在表函数的两次调用之间维护当前值，请使用暂存区。函数可能会在执行 FETCH 调用时返回错误，但仍将发出 CLOSE 调用。

CLOSE 调用

在扫描或语句结束时作出此调用（只要成功执行了 OPEN 调用）。任何自变量值都不会是当前值。函数可能会传回错误。

FINAL 调用

在语句结束时作出 FINAL 调用（只要成功执行了 FIRST 调用）。会作出此调用以便函数可以清除任何资源。在执行此调用时，函数不会返回值，但可能会返回错误。

对于未定义 FINAL CALL 的表 UDF，只会对函数作出 OPEN、FETCH 以及 CLOSE 调用。在每个 OPEN 调用前，会清除暂存区（如果存在）。

在检查涉及连接或子查询（其中，表函数访问是“内部”访问）的方案时，可以查看定义了 FINAL CALL 的表 UDF 和定义了 NO FINAL CALL 的表 UDF 之间的差别。例如，在下列类似语句中:

```
SELECT x,y,z,... FROM table_1 as A,
  TABLE(table_func_1(A.col1,...)) as B
  WHERE ...
```

在这种情况下，优化器会扫描 table_func_1 以获取 table_1 的每一行。这是因为使用 table_1 的 col1 值（传递至 table_func_1）来定义表函数扫描。

对于 NO FINAL CALL 表 UDF，会针对 table_1 的每一行重复 OPEN、FETCH、FETCH、...、CLOSE 调用序列。请注意，每个 OPEN 调用都将获取全新的暂存区。因为表函数不知道在每次扫描结束时是否有更多扫描，所以它必须在 CLOSE 处理期间执行彻底清除。如果必须重复重要的一次性打开处理，那么这样做可能效率不高。

FINAL CALL 表 UDF，提供一次性 FIRST 调用和一次性 FINAL 调用。使用这些调用在表函数的所有扫描之间分摊初始化开销和终止成本。如前所述，会对外部表的每一行作出 OPEN、FETCH、FETCH、...、CLOSE 调用，但是表函数知道它将获取 FINAL 调用，因此它不需要在执行其 CLOSE 调用时清除所有内容（以及在执行后续 OPEN 时重新分配）。另请注意，不会在扫描之间清除暂存区，这主要是因为表函数资源将跨越扫描。

以管理两种其他调用类型为代价，表 UDF 可以取得高于这些连接和子查询方案的效率。决定是否将表函数定义为 FINAL CALL 取决于其期望用途。

外部函数和方法的暂存区

*暂存区*使用户定义的函数或方法能够在两次调用之间保存其状态。

例如，在下列两种情况下，在两次调用之间保存状态很有益处:

1. 函数或方法是否正确取决于保存状态。

   此类函数或方法的示例是一个简单计数器函数，该计数器会在第一次调用时增加“1”，然后在后续每次调用时将结果增加 1。在某些情况下，可以使用此类函数来计算 SELECT 结果的行数:

   ```
   SELECT counter(), a, b+c, ...
     FROM tablex
     WHERE ...
   ```

   该函数需要一个位置来存储两次调用之间该计数器的当前值，在位置中，将保证值对于后续调用是相同的。随后在每次调用时，会增加此值并将其作为函数结果来返回。

   此类型的例程是 NOT DETERMINISTIC。它的输出并不只取决于它的 SQL 自变量值。

2. 可通过执行某些初始化操作来改进性能的函数或方法。

   此类函数或方法（可能是文档应用程序的一部分）的示例是 *match* 函数，如果给定的文档包含给定的字符串，那么返回“Y”；否则，返回“N”:

   ```
   SELECT docid, doctitle, docauthor
     FROM docs
     WHERE match('myocardial infarction', docid) = 'Y'
   ```

   此语句返回包含第一个自变量所表示的特定文本字符串值的所有文档。*match* 的可能用途如下:

   - 仅第一次调用时。

     在 DB2 数据库系统外部维护的文档应用程序中，检索所有包含字符串“myocardial infarction”的文档标识列表。此检索过程的开销很大，因此函数只会执行此过程一次，并将列表保存在便于后续调用的某个位置。

   - 每次调用时。

     使用第一次调用期间保存的文档标识列表，来查看列表中是否包含作为第二个自变量来传递的文档标识。

   此类型的例程是 DETERMINISTIC。它的应答只取决于它的输入自变量值。此处显示的是函数的性能（而非正确性），取决于能否在两次调用之间保存信息。

可通过在 CREATE 语句中指定 SCRATCHPAD 来满足这两种需求:

```
CREATE FUNCTION counter()
  RETURNS int ... SCRATCHPAD;

CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000;
```

SCRATCHPAD 关键字告知 DB2 数据库系统分配和维护例程的暂存区。暂存区的缺省大小是 100 字节，但是您可以确定暂存区的大小（以字节计）。*match* 示例是 10000 字节长。在第一次调用之前，DB2 数据库系统会将暂存区初始化为二进制零。如果正在为表函数定义暂存区，且表函数也定义了 NO FINAL CALL（缺省值），那么 DB2 数据库系统会在每个 OPEN 调用之前刷新暂存区。如果您指定表函数选项 FINAL CALL，那么 DB2 数据库系统不会在暂存区初始化之后，检查或更改暂存区的内容。对于定义

了暂存区的标量函数，DB2 数据库系统也不会在暂存区初始化之后，检查或更改暂存区的内容。暂存区的指针会在每次调用时传递至例程，并且 DB2 数据库系统会将例程的状态信息保存在暂存区中。

因此，对于 *counter* 示例，会将返回的最后一个值保留在暂存区中。如果暂存区足够大，那么 *match* 示例会将文档列表保留在暂存区中；否则，它会为列表分配内存，并将所获取内存的地址保留在暂存区中。暂存区的长度可变：长度是在例程的 CREATE 语句中定义。

暂存区只适用于语句中对例程的个别引用。如果对语句中的例程进行了多次引用，那么每个引用都有自己的暂存区，因此无法使用暂存区在两个引用之间通信。暂存区只适用于单一 DB2 代理程序（代理程序是对语句的所有方面进行处理的 DB2 实体）。未提供“全局暂存区”来协调暂存区信息在代理程序之间的共享。这对于 DB2 数据库系统建立多个代理程序来处理语句（在单分区或多分区数据库中）这一情况特别重要。在这些情况下，即使可能只对语句中的例程进行单一引用，也可能有多个代理程序正在处理工作，且每个代理程序都会有自己的暂存区。在多分区数据库（其中，引用 UDF 的语句正在处理多个分区上的数据，且正在调用每个分区上的 UDF）中，暂存区将只适用于单一分区。因此，在执行 UDF 的每个分区上，都会有一个暂存区。

如果函数的正确执行取决于对函数的每个引用都具有单一暂存区，那么会将函数注册为 DISALLOW PARALLEL。这将强制函数在单一分区上运行，从而保证对函数的每个引用将只有单一暂存区。

因为已知 UDF 或方法可能需要系统资源，所以可以在 UDF 或方法中定义 FINAL CALL 关键字。此关键字会指示 DB2 数据库系统在处理语句结尾时调用 UDF 或方法，以便 UDF 或方法可以释放其系统资源。例程释放它所获取的任何资源非常重要；在重复调用该语句的环境中，即使很小的泄漏也会变成很大的泄漏，而大泄漏会导致 DB2 数据库崩溃。

因为暂存区具有固定大小，所以 UDF 或方法本身可以包含内存分配，并因此可以利用最终调用来释放内存。例如，前面的 *match* 函数无法预测与给定文本字符串匹配的文档数。因此，下列 *match* 定义更合适:

```
CREATE FUNCTION match(varchar(200), char(15))
  RETURNS char(1) ... SCRATCHPAD 10000 FINAL CALL;
```

对于使用暂存区且在子查询中引用的 UDF 或方法，DB2 数据库系统可能会执行最终调用（如果以此方式指定 UDF 或方法），并在两次调用子查询之间刷新暂存区。如果曾在子查询中使用 UDF 或方法，那么可通过在 UDF 或方法中定义 FINAL CALL 并使用调用类型自变量，或通过始终检查暂存区的二*进制零* 状态来作出保护，以免出现此可能性。

如果的确指定 FINAL CALL，请注意您的 UDF 或方法会接收类型为 FIRST 的调用。这可用来获取和初始化某些持久资源。

下列是使用 Java 编写的简单 UDF 示例，此 UDF 使用暂存区来计算列中条目的平方之和。本示例接受列，并返回包含条目（从列顶部到当前行条目）的平方之累积和的列:

```
CREATE FUNCTION SumOfSquares(INTEGER)
RETURNS INTEGER
EXTERNAL NAME 'UDFsrv!SumOfSquares'
DETERMINISTIC
```

```
NO EXTERNAL ACTION
FENCED
NOT NULL CALL
LANGUAGE JAVA
PARAMETER STYLE DB2GENERAL
NO SQL
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO@


// Sum Of Squares using Scratchpad UDF
public void SumOfSquares(int inColumn,
                         int outSum)
throws Exception
{
  int sum = 0;
  byte[] scratchpad = getScratchpad();

  // variables to read from SCRATCHPAD area
  ByteArrayInputStream byteArrayIn = new ByteArrayInputStream(scratchpad);
  DataInputStream dataIn = new DataInputStream(byteArrayIn);

  // variables to write into SCRATCHPAD area
  byte[] byteArrayCounter;
  int i;
  ByteArrayOutputStream byteArrayOut = new ByteArrayOutputStream(10);
  DataOutputStream dataOut = new DataOutputStream(byteArrayOut);

  switch(getCallType())
  {
    case SQLUDF_FIRST_CALL:
      // initialize data
   sum = (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
   break;
    case SQLUDF_NORMAL_CALL:
      // read data from SCRATCHPAD area
      sum = dataIn.readInt();
      // work with data
      sum = sum + (inColumn * inColumn);
      // save data into SCRATCHPAD area
      dataOut.writeInt(sum);
      byteArrayCounter = byteArrayOut.toByteArray();
      for(i = 0; i < byteArrayCounter.length; i++)
      {
        scratchpad[i] = byteArrayCounter[i];
      }
      setScratchpad(scratchpad);
break;
  }
  //set the output value
  set(2, sum);
} // SumOfSquares UDF
```

请注意，有一个内置 DB2 函数可以执行 SumOfSquares UDF 所执行的任务。选择本示例是为了演示暂存区的用法。

32 位和 64 位操作系统上的暂存区

要使 UDF 或方法能够在 32 位和 64 位操作系统之间移植，您必须特别注意创建和使用包含 64 位值的暂存区的方法。建议不要针对包含一个或多个 64 位值的 scratchpad 结构声明显式长度变量，例如 64 位指针或 sqlint64 BIGINT 变量。

下列是暂存区的样本结构声明:

```
struct sql_scratchpad
{
   sqlint32 length;
   char data[100];
};
```

在例程针对暂存区定义自己的结构时，有两个选项:

1. 重新定义整个暂存区 sql_scratchpad，在这种情况下，它需要包含一个显式长度字段。例如:

   ```
   struct sql_spad
   {
     sqlint32 length;
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
   void SQL_API_FN routine( ..., struct sql_spad* scratchpad, ... )
   {
     /* Use scratchpad */
   }
   ```

2. 只重新定义暂存区 sql_scratchpad 的数据部分，在这种情况下，不需要长度字段。

   ```
   struct spaddata
   {
     sqlint32 int_var;
     sqlint64 bigint_var;
   };
   void SQL_API_FN routine( ..., struct sql_scratchpad* spad, ... )
   {
     struct spaddata* scratchpad = (struct spaddata*)spad->data;
     /* Use scratchpad */
   }
   ```

因为应用程序无法更改暂存区的长度字段中的值，所以编写如示例 1 中所示的例程没有明显的益处。示例 2 也可以在具有不同字大小的计算机之间进行移植，因此是编写例程的首选方法。

外部例程中的 SQL

使用诸如 C、Visual Basic、C# 和 Java 之类的外部编程语言编写的所有例程都可以包含 SQL。

例程（存储过程和 UDF）的 CREATE 语句，或方法的 CREATE TYPE 语句可以包含一个定义例程或方法 SQL 访问级别的子句。根据例程中所包含 SQL 的性质，您必须选择适用的子句:

NO SQL

例程根本不包含任何 SQL

CONTAINS SQL

包含 SQL，但是既不读取也不写入数据（例如: SET SPECIAL REGISTER）。

READS SQL DATA

包含可以读取表的 SQL（SELECT 和 VALUES 语句），但不会修改表数据。

MODIFIES SQL DATA

包含可以更新表的 SQL: 直接更新用户表（INSERT、UPDATE 以及 DELETE 语句）或隐式地更新 DB2 的目录表（DDL 语句）。此子句只适用于存储过程和 SQL 主体表函数。

DB2 数据库系统将在执行时验证例程是否超出其定义级别。例如，如果定义为 CONTAINS SQL 的例程尝试对表执行 SELECT，那么会因为尝试读取 SQL 数据，而导致错误 (SQLCODE -579, SQLSTATE 38004)。另请注意，嵌套的例程引用必须处于相同 SQL 级别，或处于限制性更强且包含该引用的 SQL 级别。例如，修改 SQL 数据的例程可以调用读取 SQL 数据的例程，但只能读取 SQL 数据的例程（使用 READS SQL DATA 子句进行定义）无法调用修改 SQL 数据的例程。

例程会在调用应用程序的数据库连接范围内执行 SQL 语句。例程无法建立它自己的连接，也无法重置调用应用程序的连接 (SQLCODE -751, SQLSTATE 38003)。

只有定义为 MODIFIES SQL DATA 的存储过程才能发出 COMMIT 和 ROLLBACK 语句。其他类型的例程（UDF 和方法）无法发出 COMMIT 或 ROLLBACK (SQLCODE -751, SQLSTATE 38003)。即使定义为 MODIFIES SQL DATA 的存储过程尝试对事务执行 COMMIT 或 ROLLBACK，也建议从调用应用程序执行 COMMIT 或 ROLLBACK，这样就不会意外地落实更改。如果已从应用程序（与数据库建立了 2 类连接）调用存储过程，那么存储过程无法发出 COMMIT 或 ROLLBACK 语句。

此外，只有定义为 MODIFIES SQL DATA 的存储过程才能建立它们自己的保存点，以及在保存点中回滚它们自己的工作。其他类型的例程（UDF 和方法）无法建立它们自己的保存点。当存储过程完成时，不会释放在存储过程中创建的保存点。应用程序将能够回滚该保存点。类似地，存储过程可以回滚应用程序中所定义的保存点。当例程返回时，DB2 数据库系统将隐式地释放该例程所建立的任何保存点。

例程通过将 SQLSTATE 值指定给 DB2 数据库系统传递至该例程的 sqlstate 自变量，可以通知 DB2 它是否已成功执行。某些参数样式（PARAMETER STYLE JAVA、GENERAL 以及 GENERAL WITH NULLS）不支持交换 SQLSTATE 值。

如果在处理例程所发出的 SQL 时，DB2 数据库系统遇到错误，那么会将该错误返回给例程（对任何应用程序也是执行此操作）。对于一般用户错误，例程有机会执行备用操作或更正操作。例如，如果例程尝试对表执行 INSERT，但获取“键重复”错误 (SQLCODE -813)，那么它可以改为对现有表行执行 UPDATE 操作。

但是，可能会发生某些较严重的错误，导致 DB2 数据库系统无法以正常方式继续执行。这些错误示例包括死锁、数据库分区失败或用户中断。其中一些错误会一直传播到调用应用程序。其他与工作单元相关的严重错误，会一直传播到回退时下列两项中先出现的一项: (a) 应用程序，或 (b) 有权发出事务控制语句（COMMIT 或 ROLLBACK）的存储过程。

如果在执行例程所发出的 SQL 期间发生其中一个错误，那么会将该错误返回给例程，但是 DB2 数据库系统会记住已发生严重错误。此外，在此情况下，DB2 数据库系统将自动使此例程及任何调用例程所发出的任何后续 SQL 失败 (SQLCODE -20139, SQLSTATE 51038)。此情况的唯一例外是，错误只回退到有权发出事务控制语句的最外层存储过程。在此情况下，此存储过程可以继续发出 SQL。

例程可以发出静态和动态 SQL，在任一情况下，如果使用嵌入式 SQL，那么必须预编译和绑定例程。对于静态 SQL，在预编译/绑定进程中使用的信息，与使用嵌入式 SQL 的任何客户机应用程序中所使用的信息相同。对于动态 SQL，您可以使用 **DYNAMICRULES** precompile/bind 选项来控制嵌入式动态 SQL 的当前模式和当前认证标识。对于例程和应用程序，此行为会有所不同。

会遵循针对例程程序包或语句定义的隔离级别。这会导致例程以限制性较强或较弱的隔离级别（相对于调用应用程序）来运行。调用其隔离级别不太受限（相对于调用语句）的例程时，必须考虑此重要事项。例如，如果从可重复读的应用程序调用游标稳定性函数，那么 UDF 可以表现出不可重复读的特征。

例程对专用寄存器值所作的更改不会影响调用应用程序或例程。例程从调用程序继承可更新的专用寄存器。不会将对可更新专用寄存器所作的更改传递回至调用程序。不可更新的专用寄存器会获取它们的缺省值。有关可更新及不可更新专用寄存器的更多详细信息，请参阅相关主题『专用寄存器』。

例程可以使用客户机应用程序所使用的方法，对游标执行 OPEN、FETCH 以及 CLOSE 操作。多次调用（例如，在递归情况下）同一函数可分别获取其自己的游标实例。UDF 和方法必须关闭其游标，然后调用语句才能完成，否则将发生错误 (SQLCODE -472, SQLSTATE 24517)。对 UDF 或方法所作的最终调用是关闭任何仍处于打开状态的游标的好时机。存储过程中任何未在完成前关闭的已打开游标，会作为结果集返回至客户机应用程序或调用例程。

不会自动将传递至例程的自变量视为主变量。这意味着，如果例程要在其 SQL 中将参数用作主变量，那么必须声明其自己的主变量并将参数值复制到此主变量。

注: 必须在 **DATETIME** 选项设为 ISO 的情况下，预编译和绑定嵌入式 SQL 例程。

外部例程的参数样式

外部例程实现必须符合特定约定才能交换例程参数值。这些约定称为*参数样式*。

通过指定 PARAMETER STYLE 子句来创建例程时，会指定外部例程参数样式。参数样式描述规范和顺序（将参数值传递至外部例程实现的顺序）的特征。它们也指定在将任何其他值传递至外部例程实现时该怎么办。例如，某些参数样式会作出下列指定：对于每个例程参数值，会将其他单独的 null 指示符值传递至例程实现，以提供参数可空性的信息（否则，无法使用本机编程语言数据类型来轻松确定）。

下表提供可用参数样式列表、支持每种参数样式的例程实现、支持每种参数样式的功能例程类型以及参数样式的描述:

表 2. 参数样式

参数样式	支持的语言	支持的例程类型	描述
SQL [1]	• C/C++ • OLE • .NET 公共语言运行时语言 • COBOL [2]	• UDF • 存储过程 • 方法	除调用期间传递的参数之外，会按下列顺序将下列自变量传递至例程: • 在 CREATE 语句中声明的每个参数或结果的 null 指示符。 • 要返回给 DB2 数据库系统的 SQLSTATE。 • 例程的限定名。 • 例程的特定名称。 • 要返回给 DB2 数据库系统的 SQL 诊断字符串。 可以按下列顺序将下列自变量传递至例程，取决于选项（在 CREATE 语句中指定）和例程类型: • 暂存区的缓冲区。 • 例程的调用类型。 • dbinfo 结构（包含数据库的信息）。
DB2SQL [1]	• C/C++ • OLE • .NET 公共语言运行时语言 • COBOL	• 存储过程	除调用期间传递的参数之外，会按下列顺序将下列自变量传递至存储过程: • 包含 CALL 语句上每个参数的 null 指示符的向量。 • 要返回给 DB2 数据库系统的 SQLSTATE。 • 存储过程的限定名。 • 存储过程的特定名称。 • 要返回给 DB2 数据库系统的 SQL 诊断字符串。 如果在 CREATE PROCEDURE 语句中指定 DBINFO 子句，那么会将 dbinfo 结构（包含数据库的信息）传递至存储过程。
JAVA	• Java	• UDF • 存储过程	PARAMETER STYLE JAVA 例程使用符合 Java 语言和 SQLJ 例程规范的参数传递约定。 对于存储过程，会将 INOUT 和 OUT 参数作为单一项目数组来传递，以加快值的返回。除 IN、OUT 以及 INOUT 参数之外，存储过程的 Java 方法特征符包含每个结果集（在 CREATE PROCEDURE 语句的 DYNAMIC RESULT SETS 子句中指定）的参数（类型为 ResultSet[]）。 对于 PARAMETER STYLE JAVA UDF 和方法，不会传递例程调用中指定的对象的其他自变量。 PARAMETER STYLE JAVA 例程不支持 DBINFO 或 PROGRAM TYPE 子句。对于 UDF，如果未将任何结构化数据类型指定为参数，且未将任何结构化类型、CLOB、DBCLOB 或 BLOB 数据类型指定为返回类型 (SQLSTATE 429B8)，那么只能指定 PARAMETER STYLE JAVA。此外，PARAMETER STYLE JAVA UDF 不支持表函数、调用类型或暂存区。
DB2GENERAL	• Java	• UDF • 存储过程 • 方法	此类型的例程将使用定义为与 Java 方法一起使用的参数传递约定。除非要开发表 UDF 或具有暂存区的 UDF，或者需要访问 dbinfo 结构，否则，建议您使用 PARAMETER STYLE JAVA。 对于 PARAMETER STYLE DB2GENERAL 例程，不会传递例程调用中指定的对象的其他自变量。

表 2. 参数样式 (续)

参数样式	支持的语言	支持的例程类型	描述
GENERAL	• C/C++ • .NET 公共语言运行时语言 • COBOL	• 存储过程	PARAMETER STYLE GENERAL 存储过程会在调用应用程序或例程中从 CALL 语句接收参数。如果在 CREATE PROCEDURE 语句中指定 DBINFO 子句，那么会将 dbinfo 结构（包含数据库的信息）传递至存储过程。 GENERAL 等价于 DB2 z/OS® 版的 SIMPLE 存储过程。
GENERAL WITH NULLS	• C/C++ • .NET 公共语言运行时语言 • COBOL	• 存储过程	PARAMETER STYLE GENERAL WITH NULLS 存储过程会在调用应用程序或例程中从 CALL 语句接收参数。此外，还附带一个包含 CALL 语句上每个参数的 null 指示符的向量。如果在 CREATE PROCEDURE 语句中指定 DBINFO 子句，那么会将 dbinfo 结构（包含数据库的信息）传递至存储过程。 GENERAL WITH NULLS 等价于 DB2 z/OS 版的 SIMPLE WITH NULLS 存储过程。

注:

1. 对于 UDF 和方法，PARAMETER STYLE SQL 等价于 PARAMETER STYLE DB2SQL。
2. 只能使用 COBOL 来开发存储过程。
3. 不支持 .NET 公共语言运行时运行时。

开发例程时的性能注意事项

开发例程的一个最显著益处不是扩展客户机应用程序，而是性能。

选择例程实现的方法时，请考虑下列性能影响。

NOT FENCED 方式

NOT FENCED 例程与数据库管理器在同一进程中运行。通常，以 NOT FENCED 方式运行例程，较之以 FENCED 方式运行例程具有更高的性能，因为 FENCED 例程是在引擎地址空间外部的特殊 DB2 进程中运行。

虽然以 NOT FENCED 方式运行例程时可以期望改进的例程性能，但用户代码可能会意外地或恶意地毁坏数据库或损坏数据库控制结构。只有在您需要将性能益处发挥至极致，且认为例程安全时，才应该使用 NOT FENCED 例程。有关评估和缓解将 C/C++ 例程注册为 NOT FENCED 的风险的信息，请参阅『例程安全性注意事项』主题。如果例程不够安全，不能在数据库管理器的进程中运行，请在注册例程时使用 FENCED 子句。为了限制创建和运行可能不安全的代码，DB2 数据库系统要求用户具有特权 CREATE_NOT_FENCED_ROUTINE 才能创建 NOT FENCED 例程。

如果在您运行 NOT FENCED 例程时发生异常终止，且例程已注册为 NO SQL，那么数据库管理器将尝试适当的恢复操作。但是，对于未定义为 NO SQL 的例程，数据库管理器将失败。

如果 NOT FENCED 例程使用 GRAPHIC 或 DBCLOB 数据，那么必须使用 WCHARTYPE NOCONVERT 选项预编译 NOT FENCED 例程。

FENCED THREADSAFE 方式

FENCED THREADSAFE 例程与其他例程在同一进程中运行。更具体地说，非 Java 例程会共享一个进程，而 Java 例程会共享另一个进程（和使用其他语言编写的例程是分离的）。此分离可保护 Java 例程，使其不受其他语言编写且可能更易出错的例程影响。此外，Java 例程的进程还包含一个 JVM，此 JVM 会引发高昂的内存成本，因此不为其他例程类型所使用。FENCED THREADSAFE 例程的多个调用会共享资源，因此所引发的系统开销小于 FENCED NOT THREADSAFE 例程（每个 FENCED NOT THREADSAFE 例程都在其自己的专用进程中运行）。

如果您认为例程足够安全，可以与其他例程在同一例程中运行，请在注册该例程时使用 THREADSAFE 子句。和 NOT FENCED 例程一样，有关评估和缓解将 C/C++ 例程注册为 FENCED THREADSAFE 的风险的信息可在『例程安全性注意事项』主题中找到。

如果 FENCED THREADSAFE 例程会异常结束，那么将仅终止正在运行此例程的线程。进程中的其他例程会继续运行。但是，导致此线程异常结束的故障会对进程中的其他例程线程产生负面影响，使这些线程进入陷阱、挂起或具有损坏的数据。在一个线程异常结束后，进程就不再用于新的例程调用。在所有活动用户完成其在此进程中的作业后，会终止此进程。

当您注册 Java 例程时，除非另有声明，否则会将这些例程视为 THREADSAFE。缺省情况下，所有其他 LANGUAGE 类型都是 NOT THREADSAFE。不能将使用 LANGUAGE OLE 和 OLE DB 的例程指定为 THREADSAFE。

NOT FENCED 例程必须为 THREADSAFE。无法将例程注册为 NOT FENCED NOT THREADSAFE (SQLCODE -104)。

UNIX 上的用户可通过查找 `db2fmp (Java)` 或 `db2fmp (C)` 来查看其 Java 和 C THREADSAFE 进程。

FENCED NOT THREADSAFE 方式

每个 FENCED NOT THREADSAFE 例程都在其自己的专用进程中运行。如果正在运行许多例程，那么会对数据库系统性能产生不利影响。如果例程不够安全，不能与其他例程在同一进程中运行，请在注册该例程时使用 NOT THREADSAFE 子句。

在 UNIX 上，NOT THREADSAFE 进程对于合用的 NOT THREADSAFE db2fmp 显示为 `db2fmp（pid）`（其中 pid 是使用 FENCED 方式进程的代理程序的进程标识）或 `db2fmp（空闲）`。

Java 例程

如果打算运行存在大量内存需求的 Java 例程，建议您将该例程注册为 FENCED NOT THREADSAFE。对于 FENCED THREADSAFE Java 例程调用，DB2 数据库系统会尝试选择其 Java 堆大小足以运行例程的线程 Java FENCED 方式进程。如果未能将大型堆使用者限制于其自己的进程，那么会导致多线程 Java `db2fmp` 进程发生“Java 堆不足”错误。如果 Java 例程不属于此类别，那么 FENCED 例程以线程安全方式运行时将实现更高的性能（在该方式下，例程可以共享几个 JVM）。

当前不支持 NOT FENCED Java 例程。将调用定义为 NOT FENCED 的 Java 例程，如同它定义为 FENCED THREADSAFE 一样。

C/C++ 例程

C 或 C++ 例程通常比 Java 例程要快，但更易出现错误、内存毁坏以及崩溃。由于这些原因，执行内存操作的能力使 C 或 C++ 例程成为 THREADSAFE 或 NOT FENCED 方式注册的风险候选者。这些风险可通过遵循安全例程的编程实践来缓解（请参阅『例程安全性注意事项』主题），以及通过彻底测试您的例程来缓解。

SQL 例程

SQL 例程（特别是 SQL 过程）通常也比 Java 例程要快，且性能往往与 C 例程相当。SQL 例程始终以 NOT FENCED 方式运行，提供远非外部例程可比的性能益处。如果包含复杂逻辑，那么以 C 编写的 UDF 的运行速度要快于以 SQL 编写的 UDF。如果逻辑很简单，那么 SQL UDF 将可以与任何外部 UDF 相比。

暂存区 暂存区是可以分配给 UDF 和方法的内存块。暂存区只适用于 SQL 语句中对例程的个别引用。如果对语句中的例程进行了多次引用，那么每个引用都有自己的暂存区。暂存区使 UDF 或方法能够在两次调用之间保存其状态。

对于具有复杂初始化的 UDF 和方法，您可以使用暂存区来存储第一次调用中所需的任何值，以在后续的所有调用中使用。其他 UDF 和方法的逻辑可能也要求在两次调用之间保存中间值。

使用 VARCHAR 参数取代 CHAR 参数

您可以在例程定义中使用 VARCHAR 参数取代 CHAR 参数来改进例程的性能。使用 VARCHAR 数据类型取代 CHAR 数据类型可以防止 DB2 数据库系统在传递参数之前使用空格来填充参数，并且可以缩短通过网络传输参数所需的时间量。

例如，如果您的客户机应用程序将字符串“A SHORT STRING”传递至期望 CHAR(200) 参数的例程，那么 DB2 数据库系统必须使用 186 个空格来填充该参数，以 null 结束字符串，然后通过网络将整个由 200 个字符组成的字符串和一个 null 终止符发送至例程。

相比之下，将同一字符串“A SHORT STRING”传递至期望 VARCHAR(200) 参数的例程，会导致 DB2 数据库系统只需通过网络传递一个由 14 个字符组成的字符串和一个 null 终止符。

例程安全性注意事项

开发和部署例程向您提供一个机会来极大地改进数据库应用程序的性能和效率。然而，如果数据库管理员未正确地管理例程部署，那么可能存在安全性风险。

下列部分描述安全性风险及可用来减缓这些风险的方法。安全性风险后面的部分描述如何安全地部署安全性未知的例程。

安全性风险

NOT FENCED 例程可以访问数据库管理器资源

NOT FENCED 例程与数据库管理器在同一进程中运行。由于它们与数据库引擎非常接近，因此 NOT FENCED 例程可能会意外地或恶意地毁坏数据库管理器的共享内存，或损坏数据库控制结构。任一形式的损坏都会导致数据库管理器失败。NOT FENCED 例程也可能会毁坏数据库及其表。

要确保数据库管理器及其数据库的完整性，您必须彻底地屏蔽您打算注册为 NOT FENCED 的例程。这些例程必须经过完整测试、调试，且不会显示任何意外的负面效应。在检查例程时，请密切注意内存管理和静态变量的使用。当代码错误地管理内存或错误地使用静态变量时，极有可能会发生毁坏。这些问题在 Java 和 .NET 编程语言之外的语言中非常普遍。

需要 CREATE_NOT_FENCED_ROUTINE 权限才能注册 NOT FENCED 例程。授予 CREATE_NOT_FENCED_ROUTINE 权限时，请注意接收方可能会取得对数据库管理器及其所有资源的不受限访问权。

FENCED THREADSAFE 例程可以访问其他 FENCED THREADSAFE 例程所使用的内存 FENCED THREADSAFE 例程作为共享进程中的线程来运行。其中的每个例程都可以读取同一进程中其他例程线程所使用的内存。因此，一个线程例程可从线程进程中的其他例程收集敏感数据。共享单一进程固有的另一个风险是一个内存管理存在故障的例程线程可能会毁坏其他例程线程，或导致整个线程进程崩溃。

要确保其他 FENCED THREADSAFE 例程的完整性，您必须彻底地屏蔽您打算注册为 FENCED THREADSAFE 的例程。这些例程必须经过完整测试、调试，且不会显示任何意外的负面效应。在检查例程时，请密切注意内存管理和静态变量的使用。这是最可能发生毁坏的情况，特别是在非 Java 语言中。

需要 CREATE_EXTERNAL_ROUTINE 权限才能登录 FENCED THREADSAFE 例程。授予 CREATE_EXTERNAL_ROUTINE 权限时，请注意接收方可能会监视或毁坏其他 FENCED THREADSAFE 例程的内存。

FENCED 进程所有者对数据库服务器的写访问权可能会导致数据库管理器毁坏

由 **db2icrt**（创建实例）或 **db2iupdt**（更新实例）系统命令定义运行 FENCED 进程所使用的用户标识。此用户标识不能对例程库和类的存储目录具有写访问权（在 UNIX 环境中，此目录是 `sqllib/function`；在 Windows 环境中，此目录是 `sqllib\function`）。此用户标识也不能对数据库服务器上的任何数据库、操作系统或其他重要文件和目录具有读或写访问权。

如果 FENCED 进程所有者确实对数据库服务器上的各种重要资源具有写访问权，那么系统可能会发生毁坏。例如，数据库管理员会将从未知源接收到的例程注册为 FENCED NOT THREADSAFE，并认为可通过将例程隔离在其自己的进程中来防止任何可能的危害。但是，拥有 FENCED 进程的用户标识对 `sqllib/function` 目录具有写访问权。如果用户调用此例程，但此例程对于他们是未知的，那么它会使用注册为 NOT FENCED 的例程实体备用版本来覆盖 `sqllib/function` 中的库。第二个例程对整个数据库管理器具有不受限的访问权，并且会因此分发数据库表中的敏感信息，毁坏数据库，收集认证信息，或使数据库管理器崩溃。

确保拥有 FENCED 进程的用户标识对数据库服务器上的重要文件或目录不具有写访问权（特别是 `sqllib/function` 和数据库数据目录）。

例程库和类的脆弱性

如果未控制对例程库和类的存储目录的访问权，那么可能会删除或覆盖例程库和类。如先前项中所讨论的那样，使用恶意（或编程质量不高）的例程替换 NOT FENCED 例程主体会严重地破坏数据库服务器及其资源的稳定性、完整性以及保密性。

要保护例程的完整性，您必须管理对包含例程库和类的目录的访问权。确保尽可能少的用户可以访问此目录及其文件。分配对此目录的写访问权时，请注意此特权可以向用户标识的所有者，提供对数据库管理器及其所有资源的不受限访问权。

部署可能不安全的例程

如果您偶尔从未知源获取例程，请确保准确地了解其用途，然后再构建、注册以及调用该例程。建议将该例程注册为 FENCED 和 NOT THREADSAFE，除非您已彻底测试该例程并且它未显示任何意外的负面效果。

如果您需要部署一个不符合安全例程标准的例程，请将该例程注册为 FENCED 和 NOT THREADSAFE。FENCED 和 NOT THREADSAFE 例程必须执行下列操作，才能确保维护数据库完整性:

- 在单独的 DB2 进程中运行（不与任何其他例程共享此进程）。如果它们异常终止，那么不会影响数据库管理器。
- 使用未被数据库所使用的内存。意外的赋值错误将不会影响数据库管理器。

例程代码页注意事项

会将字符数据传递至代码页中的外部例程（由建立例程时使用的 PARAMETER CCSID 选项隐式指定）。类似地，数据库假设从例程输出的字符串使用由 PARAMETER CCSID 选项隐式指定的代码页。

如果客户机程序（例如使用代码页 C）访问具有不同代码页（例如，代码页 S）的节，而该代码页会使用不同的代码页（例如，代码页 R）来调用例程，那么会出现下列事件:

1. 调用 SQL 语句时，会将输入字符数据从客户机应用程序的代码页 (C) 转换为与该节相关联的代码页 (S)。对于 BLOB 或将用作 FOR BIT DATA 的数据，不会进行转换。
2. 如果例程的代码页与节的代码页不同，那么会在调用例程前将输入字符数据（BLOB 和 FOR BIT DATA 除外）转换为例程的代码页 (R)。

   强烈建议您使用调用服务器例程所依据的代码页 (R) 来预编译、编译以及绑定服务器例程。并非在所有情况下都可以这样做。例如，您可以在 Windows 环境中创建 Unicode 数据库。然而，如果 Windows 环境没有 Unicode 代码页，那么您必须预编译、编译以及绑定使用 Windows 代码页来创建的应用程序。如果应用程序不含预编译器无法理解的任何特殊定界字符，那么该例程将起作用。
3. 如果例程完成，那么数据库管理器会根据需要将所有输出字符数据，从例程代码页 (R) 转换为节代码页 (S)。如果例程在执行期间抛出错误，那么也会将例程中的 SQLSTATE 和诊断消息，从例程代码页转换为节代码页。BLOB 或 FOR BIT DATA 字符串不会进行转换。
4. 如果语句完成，那么会将输出字符数据，从节代码页 (S) 转换回为客户机应用程序的代码页 (C)。对于 BLOB 或已用作 FOR BIT DATA 的数据，不会进行转换。

通过在 CREATE FUNCTION、CREATE PROCEDURE 以及 CREATE TYPE 语句上使用 DBINFO 选项，会将例程代码页传递至例程。使用此信息，可以编写对代码页敏感的例程，以在许多不同的代码页中操作。

32 位和 64 位应用程序和例程支持

DB2 Database for Linux, UNIX, and Windows 支持在各种平台上开发和部署应用程序和例程，其中包括过程和用户定义的函数 (UDF)。要让应用程序和例程正常工作，必须复审和了解 DB2 数据库 32 位和 64 位支持注意事项。

要开始，最好是先澄清下列几点:

- 32 位硬件平台正在运行 32 位操作系统，而 64 位硬件平台正在运行 64 位操作系统。
- 您可以将 DB2 数据库的 32 位实例安装至 32 位操作系统或 64 位操作系统，但只能将 DB2 实例的 64 位实例安装至 64 位操作系统。
- 32 位应用程序是在 32 位操作系统上构建的应用程序。
- 64 位应用程序是在 64 位操作系统上构建的应用程序。

下表概述了客户机应用程序和例程的 DB2 数据库 32 位和 64 位支持，且作出下列假设:

表 3. 支持在 32 位或 64 位硬件平台上运行 32 位和 64 位应用程序

	32 位硬件 + 操作系统	**64 位硬件 + 操作系统**
32 位应用程序	是	是
64 位应用程序	否	是

下表指示支持从 DB2 客户机应用程序创建与 DB2 数据库服务器的连接。

表 4. 支持从 32 位和 64 位客户机连接至 32 位和 64 位服务器

	32 位服务器	**64 位服务器**
32 位客户机	是	是
64 位应用程序	是	是

表 5. 支持在 32 位或 64 位硬件平台上运行 32 位和 64 位应用程序

	32 位硬件和操作系统	**64 位硬件和操作系统**
32 位应用程序	是	是
64 位应用程序	否	是

下表指示支持从 DB2 客户机应用程序创建与 DB2 数据库服务器的连接。

表 6. 支持从 32 位和 64 位客户机连接至 32 位和 64 位服务器

	32 位服务器	**64 位服务器**
32 位客户机	是	是
64 位应用程序	是	是

表 7. 支持在 32 位和 64 位服务器上运行 FENCED 和 NOT FENCED 过程与 UDF

	32 位服务器	**64 位服务器**
32 位 FENCED 过程或 UDF	是	是[1、2、3]
64 位 FENCED 过程或 UDF	否	是

表 7. 支持在 32 位和 64 位服务器上运行 FENCED 和 NOT FENCED 过程与 UDF (续)

	32 位服务器	**64 位服务器**
32 位 NOT FENCED 过程或 UDF	是	否[2]
64 位 NOT FENCED 过程或 UDF	否	是

注:

1. 在 64 位服务器上运行 32 位过程可能很慢。
2. 32 位例程必须创建为 FENCED 和 NOT THREADSAFE，才能在 64 位服务器上工作。
3. 无法在 Linux/IA-64 数据库服务器上调用 32 位例程。

对外部例程的 32 位和 64 位支持

对 32 位和 64 位外部例程的支持由例程的 CREATE 语句中下列两个子句的其中一个来确定: FENCED 子句或 NOT FENCED 子句。

外部例程的例程主体是使用编程语言来编写，且编译为可以在调用例程时装入和运行的库或类文件。FENCED 或 NOT FENCED 子句的规范确定外部例程是在有别于数据库管理器的 FENCED 环境中运行，还是在数据库管理器所在的寻址空间中运行（通过使用共享内存取代 TCPIP 进行通信，从而取得更好的性能）。缺省情况下，不论选择的其他子句为何，例程始终创建为 FENCED。

下表说明在运行相同操作系统的 32 位和 64 位数据库服务器上，DB2 数据库系统对运行 FENCED 和 NOT FENCED 32 位和 64 位例程的支持。

表 8. 对 32 位和 64 位外部例程的支持

例程的位宽	**32 位服务器**	**64 位服务器**
32 位 FENCED 过程或 UDF	支持	支持[1]
64 位 FENCED 过程或 UDF	不支持[3]	支持
32 位 NOT FENCED 过程或 UDF	支持	支持[1、2]
64 位 NOT FENCED 过程或 UDF	不支持[3]	支持

注:

1. 在 64 位服务器上运行 32 位例程不如在 64 位服务器上运行 64 位例程那样快。
2. 32 位例程必须创建为 FENCED 和 NOT THREADSAFE，才能在 64 位服务器上工作。
3. 不能在 32 位寻址空间中运行 64 位应用程序和例程。

表中需要注意的重要事项是 32 位 NOT FENCED 过程不能在 64 位 DB2 数据库服务器上运行。如果必须将 32 位 NOT FENCED 例程部署至 64 位平台，请在对这些例程进行编目之前，从这些例程的 CREATE 语句中除去 NOT FENCED 子句。

64 位数据库服务器上使用 32 位库的例程的性能

可以在 64 位 DB2 数据库服务器上调用使用 32 位例程库的例程。但是，此调用的性能低于在 64 位服务器上调用 64 位例程。

性能降级的原因是，每次尝试在 64 位服务器上执行 32 位例程之前，会先尝试将其作为 64 位库来调用。如果这样做失败，那么随后会将库作为 32 位库来调用。尝试将 32 位库作为 64 位库来调用失败时，会在 db2diag 日志文件中产生错误消息 (SQLCODE -444)。

Java 类独立于位宽。只有 Java 虚拟机（JVM）才会分类为 32 位或 64 位。DB2 数据库系统仅支持使用位宽与实例（在其中使用 JVM）相同的 JVM。换句话说，在 32 位 DB2 实例中，只能使用 32 位 JVM，而在 64 位 DB2 实例中，只能使用 64 位 JVM。这将确保 Java 例程正常运作，并尽可能将性能发挥至极致。

外部例程中的 XML 数据类型支持

使用下列编程语言编写的外部过程和函数支持数据类型为 XML 的参数和变量:

- C
- C++
- COBOL
- Java
- .NET CLR 语言

外部 OLE 和 OLEDB 例程不支持数据类型为 XML 的参数。

在外部例程代码中表示 XML 数据类型值的方法与表示 CLOB 数据类型的方法相同。

声明数据类型为 XML 的外部例程参数时，将用来在数据库中创建例程的 CREATE PROCEDURE 和 CREATE FUNCTION 语句，必须指定会将 XML 数据类型存储为 CLOB 数据类型。CLOB 值的大小应该接近 XML 参数所表示 XML 文档的大小。

下列 CREATE PROCEDURE 语句显示使用 C 编程语言来实现且具有名为 parm1 的 XML 参数的外部过程的 CREATE PROCEDURE 语句:

```
CREATE PROCEDURE myproc(IN parm1 XML AS CLOB(2M), IN parm2 VARCHAR(32000))
LANGUAGE C
FENCED
PARAMETER STYLE SQL
EXTERNAL NAME 'mylib!myproc';
```

在创建外部 UDF 时适用类似注意事项，如下列示例中所示:

```
CREATE FUNCTION myfunc (IN parm1 XML AS CLOB(2M))
RETURNS SMALLINT
LANGUAGE C
PARAMETER STYLE SQL
DETERMINISTIC
NOT FENCED
NULL CALL
NO SQL
NO EXTERNAL ACTION
EXTERNAL NAME 'mylib1!myfunc'
```

将 XML 数据作为 IN、OUT 或 INOUT 参数传递至存储过程时，会具体化此 XML 数据。如果使用的是 Java 存储过程，那么可能需要根据 XML 自变量的数量和大小，以及正在并发执行的外部存储过程数来增加堆大小（**java_heap_sz** 配置参数）。

在外部例程代码中访问、设置以及修改 XML 参数和变量值的方法，与在数据库应用程序中执行相应操作的方法相同。

外部例程限制

以下限制适用于外部例程，在开发或调试外部例程时应该考虑这些限制。

适用于所有外部例程的限制:

- 无法在外部例程中创建新线程。
- 无法从外部函数或外部方法中调用连接级别 API。
- 外部例程无法从键盘接收输入并将输出显示至标准输出。请不要使用标准输入/输出流。例如:
  - 在外部 Java 例程代码中，请不要发出 System.out.println() 方法。
  - 在外部 C 或 C++ 例程代码中，请不要发出 printf()。
  - 在外部 COBOL 例程代码中，请不要发出 display。

  虽然外部例程无法将数据显示至标准输出，但是它们可以包含将数据写入数据库服务器文件系统上的文件的代码。

  对于在 UNIX 环境中运行的 FENCED 例程，要在其中创建文件的目标目录或文件本身必须具有适当的许可权，如此才能让 `sqllib/adm/.fenced` 文件的所有者创建或写入文件。对于 NOT FENCED 例程，实例所有者必须对在其中打开文件的目录，具有创建和读/写许可权。

  注: DB2 数据库系统不会尝试将例程所执行的任何外部输入或输出与其自己的事务同步。因此，如果 UDF 在事务期间写入文件，并且稍后由某些原因而回退该事务，那么不会尝试发现或撤销对文件所作的写入。
- 无法在外部例程中执行连接相关的语句或命令。此限制适用于下列语句和命令:
  - **BACKUP DATABASE**
  - CONNECT
  - CONNECT TO
  - CONNECT RESET
  - **CREATE DATABASE**
  - **DROP DATABASE**
  - FORWARD RECOVERY
  - **RESTORE DATABASE**
- 不推荐在例程中使用操作系统函数。不会限制使用这些函数，下列情况除外:
  - 不能为外部例程安装用户定义的信号处理程序。如果未遵循此限制，那么可能会导致意外的外部例程运行时失败、数据库异常结束或其他问题。安装信号处理程序也可能会干扰 Java 例程的 JVM 操作。
  - 终止进程的系统调用可能会异常终止其中一个 DB2 数据库系统进程，并导致数据库系统或数据库应用程序失败。

    如果其他系统调用干扰 DB2 数据库管理器的正常操作，那么也可能会导致问题。例如，如果函数尝试从内存卸载包含用户定义的函数的库，那么可能会导致严重问题。在编写和测试包含系统调用的外部例程时，请格外小心。

- 从 DB2 pureScale® for Linux V9.8 FP2 开始，不能在不受保护的例程中使用导致创建新进程的操作系统函数用法。这些函数包括 fork()、popen() 和 system()。使用这些函数可能会干扰 DB2 服务器与集群高速缓存设施之间的通信并导致该例程返回 SQL0430N 错误。
- 外部例程不能包含会终止当前进程的命令。外部例程必须始终在不终止当前进程的情况下，将控制权返回给 DB2 数据库管理器。
- 当数据库处于活动状态时，不能更新外部例程库、类或组合件（特殊情况除外）。如果在 DB2 数据库管理器处于活动状态时需要更新，且不能选择停止和启动实例，请使用不同的实例为例程创建新的库、类或组合件。然后，使用 ALTER 语句来更改外部例程的 EXTERNAL NAME 子句值，以便它引用新库、类或组合件文件的名称。
- 环境变量 **DB2CKPTR** 在外部例程中不可用。会在启动数据库管理器时捕获所有其他名称以“DB2”开头的环境变量，供外部例程使用。
- FENCED 外部例程无法使用某些名称不是以“DB2”开头的环境变量。例如，无法使用 **LIBPATH** 环境变量。但是，NOT FENCED 外部例程可以使用这些变量。
- 外部例程无法使用在启动 DB2 数据库管理器后设置的环境变量值。
- 应该限制在外部例程中使用受保护资源（这些资源一次只能供一个进程访问）。如果使用，请尝试降低在两个外部例程试图访问受保护资源发生死锁的可能性。如果试图访问受保护资源时两个或更多外部例程发生死锁，那么 DB2 数据库管理器将无法检测到或解决这种情况。这将导致外部例程进程挂起。
- 不应该在 DB2 数据库服务器上显式地分配外部例程参数的内存。DB2 数据库管理器会根据例程的 CREATE 语句中的参数声明来自动分配存储器。请不要在外部例程中改变参数的任何存储器指针。如果试图使用本地创建的存储器指针来更改指针，那么可能会导致内存泄漏、数据毁坏或异常结束。
- 请不要在外部例程中使用静态或全局数据。DB2 数据库系统无法保证静态或全局变量所使用的内存在两次外部例程调用之间保留不变。对于 UDF 和方法，您可以使用暂存区来存储两次调用之间使用的值。
- 会缓存所有 SQL 参数值。这意味着会创建值副本并将其传递至外部例程。如果对外部例程的输入参数作出了更改，那么这些更改将不会影响 SQL 值或处理。但是，如果外部例程将超出 CREATE 语句所指定数量的数据写入至输入或输出参数，那么会发生内存毁坏，而且例程会异常结束。
- **LOAD** 实用程序不支持装入含有引用受防护过程的列的表中。如果对此类表发出 **LOAD** 命令，那么将接收到错误消息 SQL1376N。要解决此限制，可以重新定义要取消防护的例程或使用导入实用程序。

仅适用于外部过程的限制

- 从嵌套存储过程中返回结果集时，您可以在多个嵌套级别使用同一名称打开游标。但是，低于版本 8 的应用程序将只能访问打开的第一个结果集。使用不同程序包级别打开的游标不适用此限制。

仅适用于外部函数的限制

- 外部函数无法返回结果集。在完成函数的最终调用时，必须关闭外部函数中打开的所有游标。
- 应该在外部例程返回之前释放外部例程中的动态内存分配。如果未这样做，那么将导致内存泄漏，且 DB2 进程消耗会持续增长，导致数据库系统内存不足。

对于用户定义的外部函数和外部方法，可以使用暂存区来分配多个函数调用所需的动态内存。如果以此方式使用暂存区，请在 CREATE FUNCTION 或 CREATE METHOD 语句中指定 FINAL CALL 属性。这将确保在例程返回之前释放所分配的内存。

创建外部例程

使用创建包括其他实现的例程的相似方法来创建包括过程和函数的外部例程，然而，需要执行其他几个步骤，因为例程实现需要编写、编译以及部署源代码。

开始之前

- 必须安装 IBM 数据服务器客户机。
- 数据库服务器必须正在运行支持所选实现编程语言编译器和开发软件的操作系统。
- 必须在数据库服务器上安装所选编程语言的必需编译器和运行时支持。
- 有权执行 CREATE PROCEDURE、CREATE FUNCTION 或 CREATE METHOD 语句。

限制

有关与外部例程相关联的限制列表，请参阅:

- 第 30 页的『外部例程限制』

关于此任务

在下列情况下，您不妨选择实现外部例程:

- 您需要在例程中封装复杂逻辑，以访问数据库或在数据库外部执行操作。
- 您要求从下列任何项调用封装的逻辑: 多个应用程序、CLP、另一个例程（过程、函数 (UDF) 或方法）或触发器。
- 您最擅长使用编程语言（而不是 SQL 和 SQL PL 语句）来编写此逻辑。
- 您需要例程逻辑执行数据库外部操作，例如写入或读取数据库服务器上的文件、运行另一个应用程序或运行无法使用 SQL 和 SQL PL 语句来表示的逻辑。

过程

1. 使用所选编程语言来编写例程逻辑。
   - 有关外部例程、例程功能以及例程功能实现的常规信息，请参阅“先决条件”部分中所引用的主题。
   - 使用或导入支持 SQL 语句执行时所需的任何必需头文件。
   - 正确地使用映射至 DB2 SQL 数据类型的编程语言数据类型来声明变量和参数。
2. 必须根据所选编程语言的参数样式所需的格式来声明参数。有关参数和原型声明的更多信息，请参阅:
   - 第 20 页的『外部例程的参数样式』
3. 将代码构建到库或类文件。
4. 将库或类文件复制到数据库服务器上的 DB2 *function* 目录。建议将 DB2 例程的相关联组合件或库存储至函数目录。要了解有关函数目录的更多信息，请参阅下列任一语句的 EXTERNAL 子句: CREATE PROCEDURE 或 CREATE FUNCTION。

如果需要，您可以将组合件复制到服务器上的另一个目录，但是为了成功调用例程，您必须记下组合件的标准路径名，因为下一步需要该标准路径名。

5. 动态或静态地执行适合于例程类型的 SQL 语言 CREATE 语句：CREATE PROCEDURE 或 CREATE FUNCTION。
   - 使用所选 API 或编程语言的适当值来指定 LANGUAGE 子句。示例包括：CLR、C 以及 JAVA。
   - 使用支持的参数样式（在例程代码中实现）的名称来指定 PARAMETER STYLE 子句。
   - 使用库、类或组合件文件（将通过下列其中一个值与例程相关联）的名称来指定 EXTERNAL 子句：
     – 例程库、类或组合件文件的标准路径名。
     – 例程库、类或组合件文件的相对路径名（相对于函数目录）。

     缺省情况下，DB2 将在函数目录中按名称查找库、类或组合件文件，除非在 EXTERNAL 子句中指定库、类或组合件文件的标准或相对路径名。
   - 如果您的例程是过程且它会将一个或多个结果集返回给调用者，请指定具有数值的 DYNAMIC RESULT SETS。
   - 指定描述例程的特征时所需的任何其他子句。

下一步做什么

要调用外部例程，请参阅例程调用

编写例程

开始之前

就编写例程而言，三种例程（过程、UDF 以及方法）具有许多共同点。例如，这三种例程采用其中一些相同的参数样式，支持通过各种客户机接口（嵌入式 SQL、CLI 以及 JDBC）来使用 SQL，以及都可以调用其他例程。最后，下列步骤表示用于编写例程的单一方法。

例程类型具有一些特定的例程功能。例如，结果集特定于存储过程，而暂存区特定于 UDF 和方法。如果您遇到不适用于您所开发的例程类型的步骤，请转至该步骤后面的步骤。

在编写例程之前，您必须决定下列事项：

- 所需的例程类型。
- 将用来编写例程的编程语言。
- 例程中需要 SQL 语句时要使用的接口。

另请参阅有关安全性、库和类管理以及性能注意事项的主题。

过程

要创建例程主体，您必须执行下列操作：

1. *只适用于外部例程*。接受来自调用应用程序或例程的输入参数，并声明输出参数。例程如何接受参数依赖于用来创建例程的参数样式。每种参数样式都定义可传递至例程主体的参数集以及参数的传递顺序。

例如，下列是 PARAMETER STYLE SQL 中使用 C（使用 sqludf.h）编写的 UDF 主体的特征符:

```
SQL_API_RC SQL_API_FN product ( SQLUDF_DOUBLE *in1,
                                SQLUDF_DOUBLE *in2,
                                SQLUDF_DOUBLE *outProduct,
                                SQLUDF_NULLIND *in1NullInd,
                                SQLUDF_NULLIND *in2NullInd,
                                SQLUDF_NULLIND *productNullInd,
                                SQLUDF_TRAIL_ARGS )
```

2. 添加例程要执行的逻辑。您可以在例程主体中采用的某些功能如下所示:
   - 调用其他例程（嵌套）或调用当前例程（递归）。
   - 在定义为具有 SQL（CONTAINS SQL、READS SQL 或 MODIFIES SQL）的例程中，例程可以发出 SQL 语句。例程的注册方式控制可以调用的语句类型。
   - 在外部 UDF 和方法中，使用暂存区来保存两个调用之间的状态。
   - 在 SQL 过程中，使用条件处理程序来确定出现指定条件时 SQL 过程的行为。您可以根据 SQLSTATE 来定义条件。
3. *只适用于存储过程*。返回一个或多个结果集。除与调用应用程序交换的个别参数之外，存储过程还能够返回多个结果集。只有 SQL 例程和 CLI、ODBC、JDBC 以及 SQLJ 例程和客户机才能接受结果集。

结果

除了编写例程之外，您还需要注册该例程，然后才能调用。这是使用与您所开发的例程类型匹配的 CREATE 语句来完成。通常，编写和注册例程的顺序无关紧要。但是，如果例程发出对自身进行引用的 SQL，那么必须先注册例程，然后构建例程。在此情况下，必须已注册例程，才能成功绑定。

调试例程

在生产服务器上部署例程之前，您必须在测试服务器上对其进行充分测试和调试。

关于此任务

这对于需要注册为 NOT FENCED 例程的例程而言尤其重要，这是因为，它们可以对数据库管理器的内存、数据库和数据库控制结构进行不受限制的访问。FENCED THREADSAFE 例程与其他例程共享内存，因此也需要密切注意。

常见例程问题的核对表

要确保例程正确地执行，请检查下列事项:

- 是否已正确地注册该例程。CREATE 语句中提供的参数必须与例程体所处理的自变量匹配。在牢记这一点的情况下，检查下列具体事项:
  - 例程体所使用的自变量的数据类型适合于 CREATE 语句中定义的参数类型。
  - 例程写入输出变量的字节数不超出 CREATE 语句为相应结果定义的字节数。
  - 如果使用相应的 CREATE 选项来注册例程，那么存在 SCRATCHPAD、FINAL CALL 和 DBINFO 的例程自变量。
  - 对于外部例程而言，CREATE 语句中 EXTERNAL NAME 子句的值必须与例程库和入口点匹配（随操作系统不同，是否区分大小写也有所变化）。

- 对于 C++ 例程，C++ 编译器将对入口点名称应用类型声明。您或者应该在 EXTERNAL NAME 子句中指定已进行类型装饰的名称，或者应该在用户代码中将入口点定义为 `extern "C"`。
- 调用期间指定的例程名必须与该例程的注册名（由 CREATE 语句定义）匹配。缺省情况下，例程标识将转换为大写。这并不适用于定界标识，这些标识不会转换为大写，因此区分大小写。

  必须将例程放入 CREATE 语句所指定的目录路径，未指定路径时，DB2 数据库系统在缺省情况下将在此路径中查找该例程。对于 UDF、方法和受防护过程，这是 `sqllib/function` (UNIX) 或 `sqllib\function` (Windows)。对于未受防护过程而言，这是 `sqllib/function/unfenced` (UNIX) 或 `sqllib\function\unfenced` (Windows)。

- 已使用正确的调用序列、预编译（对于嵌入式 SQL）、编译和链接选项来构建例程。
- 已将应用程序与数据库绑定（使用 CLI、ODBC 或 JDBC 编写的应用程序除外）。如果例程包含 SQL 并且未使用任何这些接口，那么也需要对其进行绑定。
- 例程准确地将任何错误信息返回到客户机应用程序。
- 如果对例程定义了 FINAL CALL，那么将考虑所有适用的调用类型。
- 释放例程所使用的系统资源。
- 如果您尝试调用例程，但接收到指示您的特权不足以执行此操作的错误 (SQLCODE -551, SQLSTATE 42501)，这可能是因为您对该例程不具有 EXECUTE 特权。具有 SECADM 或 ACCESSCTRL 权限的用户或者任何对该例程具有 EXECUTE WITH GRANT OPTION 的用户可以将此特权授予例程的任何调用者。有关授权和例程的相关主题提供了有关如何有效管理对此特权的使用的详细信息。

例程调试技术

要调试例程，请使用下列技术:

- 开发中心提供了用于调试包含 SQL 主体的过程和 Java 过程的众多调试工具。
- 不可能将诊断数据从例程写至屏幕。如果您打算将诊断数据写入文件，请确保写入可全局访问的目录，例如 `\tmp`。不要写入由数据库管理器或数据库使用的目录。

  对于过程而言，一种安全的替代方法是，将诊断数据写入 SQL 表。必须使用 MODIFIES SQL DATA 子句来注册您所测试的过程，这样才能写入 SQL 表。如果您需要现有过程将数据写入 SQL 表或者不再将数据写入 SQL 表，那么必须删除该过程，然后在指定或不指定 MODIFIES SQL DATA 子句的情况下重新注册该过程。在删除并重新注册过程之前，请了解其从属项。
- 您可以通过编写直接调用例程入口点的简单应用程序来调试该例程。有关使用所提供的调试器的信息，请查阅编译器文档。

外部例程库和类管理

要成功开发和调用外部例程，必须正确部署和管理外部例程库和类文件。

如果最初创建外部例程以及部署库和类文件时小心谨慎，那么可以将外部例程库和类文件的管理工作量减至最少。

外部例程管理的主要注意事项如下所示:

- 部署外部例程库和类文件
- 保护外部例程库和类文件的安全
- 解析外部例程库和类
- 修改外部例程库和类文件
- 备份和复原外部例程库和类文件
- 验证所有例程库是否在 sqllib/function 目录中并且它们是否在正确的库中。选择包含例程库最终版本的成员。该库与执行 **db2iupdt** 命令的最后一个成员的库是同一个库。

系统管理员、数据库管理员以及数据库应用程序开发者都应该承担责任，确保在执行例程开发和数据库管理任务期间，外部例程库和类文件是安全的且已正确保留。

部署外部例程库和类

部署外部例程库和类是指在根据源代码构建外部例程库和类后，将其复制到数据库服务器。

必须将外部例程库、类或组合件文件复制到数据库服务器上的 DB2 数据库系统的函数目录或其子目录。此位置是建议的外部例程部署位置。要了解有关函数目录的更多信息，请参阅下列任一 SQL 语句的 EXTERNAL 子句的描述: CREATE PROCEDURE 或 CREATE FUNCTION。

您可以将外部例程类、库或组合件复制到服务器上的其他目录位置（取决于用来实现例程的 API 和编程语言），然而，通常不推荐这样做。如果这样做，那么必须专门记录标准路径名并确保此值是与 EXTERNAL NAME 子句搭配使用，才能成功调用例程。

可以使用各种常用的文件传输工具将库和类文件复制到数据库服务器文件系统。可以将 Java 例程从安装了 DB2 客户机的计算机复制到 DB2 数据库服务器（可以使用专门针对此用途而设计的特殊内置过程进行复制）。请参阅有关 Java 例程的主题，以了解更多详细信息。

执行适合于例程类型的 SQL 语言 CREATE 语句（CREATE PROCEDURE 或 CREATE FUNCTION）时，请确保指定相应的子句并特别注意 EXTERNAL NAME 子句。

- 使用所选 API 或编程语言的适当值来指定 LANGUAGE 子句。示例包括: CLR、C 以及 JAVA。
- 使用支持的参数样式（在例程代码中实现）的名称来指定 PARAMETER STYLE 子句。
- 使用库、类或组合件文件（将通过下列其中一个值与例程相关联）的名称来指定 EXTERNAL 子句:
  - 例程库、类或组合件文件的标准路径名。
  - 例程库、类或组合件文件的相对路径名（相对于函数目录）。

缺省情况下，DB2 数据库系统将在函数目录中按名称查找库、类或组合件文件，除非在 EXTERNAL 子句中指定库、类或组合件文件的标准或相对路径名。

外部例程库或类文件的安全性

外部例程库是存储在数据库服务器的文件系统上，而 DB2 数据库管理器不会以任何方式对外部例程库进行备份或保护。要使例程能够继续成功被调用，与例程相关联的库必须继续存在于 CREATE 语句（用来创建例程）的 EXTERNAL 子句所指定的位置。

在创建例程后，请不要移动或删除例程库；否则，会导致例程调用失败。

要防止意外或蓄意删除或替换例程库，您必须限制对数据库服务器上包含例程库的目录的访问权，并限制对例程库文件的访问权。要进行限制，请使用操作系统命令来设置目录和文件许可权。

解析外部例程库和类

DB2 外部例程库解析在 DB2 实例级别执行。这意味着，在包含多个 DB2 数据库的 DB2 实例中，可以在一个数据库中创建外部例程并让那些外部例程使用已用于另一数据库中的例程的外部例程库。

实例级外部例程解析通过允许多个例程定义与单一库相关联来支持代码复用。未以此方式复用外部例程库，而是让数据库服务器的文件系统包含外部例程库的副本时，库名可能会发生冲突。当单一实例中存在多个数据库，并且每个数据库中的例程与它们自己的库副本和例程主体类相关联时，尤其会发生这种情况。如果一个数据库中的例程所使用的库或类的名称与同一实例中另一数据库中的例程所使用的库或类的名称完全相同，那么将发生冲突。

为了最大程度地降低发生这种情况的可能性，建议在实例级函数目录（`sqllib/function` 目录）中存储例程库的单一副本，并让每个数据库中所有例程定义的 EXTERNAL 子句引用唯一的库。

如果必须创建两个功能不同的同名例程库，那么执行两个附加步骤以便最大程度地降低库名冲突可能性至关重要。

对于 C、C++、COBOL 和 ADO.NET 例程:

可以通过执行下列操作来最大程度地减少或解决库名冲突:

1. 将包含例程主体的库存储在每个数据库的不同目录中。
2. 创建例程时，通过 EXTERNAL NAME 子句值来指定给定的库的完整路径（而不是指定相对路径）。

对于 Java 例程:

无法通过将相应的类文件移入不同的目录来解决类名冲突，这是因为，**CLASSPATH** 环境变量将应用于整个实例。在 **CLASSPATH** 中遇到的第一个类就是所使用的类。因此，如果有两个不同的 Java 例程并且它们引用同名的类，那么其中一个例程将使用不正确的类。有两种可能的解决方案：将受影响类重命名，或者为每个数据库创建不同的实例。

修改外部例程库和类文件

在部署现有的外部例程并在生产数据库系统环境中使用该例程之后，可能需要对其逻辑进行修改。您可以对现有例程进行修改，但重要的是仔细地进行修改，以便定义清晰的接管时间点执行更新以及最大程度地降低中断任何并发例程调用的风险。

如果需要更新外部例程库，请不要重新编译该例程并将其重新链接到数据库管理器运行期间使用的当前例程的目标文件（例如 `sqllib/function/foo.a`）。如果当前例程调

用正在访问高速缓存的例程进程版本，并且底层库被替换，那么可能会导致例程调用失败。如果有必要在不停止然后重新启动 DB2 数据库管理器的情况下更改例程的主体，请完成下列步骤:

1. 创建使用另一个库或类文件名的新外部例程库。
2. 如果它是嵌入式 SQL 例程，请使用 **BIND** 命令对例程程序包与数据库进行绑定。
3. 使用 ALTER ROUTINE 语句来更改例程定义，以使 EXTERNAL NAME 子句引用更新后的例程库或类。如果要更新的例程主体由多个数据库中编目的例程使用，那么必须对每个受影响的数据库执行本节描述的操作。
4. 要更新已构建到 JAR 文件中的 Java 例程，必须发出 CALL SQLJ.REFRESH_CLASSES() 语句以强制 DB2 数据库管理器装入新类。如果更新 Java 例程类之后未发出 CALL SQLJ.REFRESH_CLASSES() 语句，那么 DB2 数据库系统将继续使用先前的类版本。DB2 数据库系统将在 COMMIT 或 ROLLBACK 发生时刷新类。

一旦例程定义被更新，对该例程进行的所有后续调用都将装入并运行新的外部例程库或类。

备份和复原外部例程库和类文件

执行数据库备份时，不会将外部例程库与其他数据库对象一起备份。类似地，复原数据库时，不会复原外部例程库。

如果数据库备份和复原的目的是为了重新部署数据库，那么必须以此方式将外部例程库文件从原始数据库服务器文件系统复制到目标数据库服务器文件系统，以保留外部例程库的相对路径名。

外部例程库管理和性能

外部例程库管理会影响例程性能，这是因为，DB2 数据库管理器动态地对外部例程库进行高速缓存，以便提高使用例程的性能。

为了获得最佳的外部例程性能，请考虑下列各项:

- 保持每个库中的例程数尽可能地少。使用众多较小的外部例程库比使用几个大库更好。
- 将通常一起调用的例程的例程函数一起分组到源代码中。将此代码编译到外部例程库中时，经常调用的各个例程的入口点之间的距离较近，这使数据库管理器能够提高更好的高速缓存支持。改善高速缓存支持的原因是，装入一次单一外部例程库并调用该库中的多个外部例程函数有助于提高效率。

  对于使用 C 或 C++ 编程语言实现的外部例程而言，装入库的成本只对 C 例程中始终使用的库发生一次。调用一次例程之后，该进程中同一线程的所有后续调用都不需要重新装入该例程的库。

第 3 章 .NET 公共语言运行时 (CLR) 例程

在 DB2 数据库系统中，公共语言运行时 (CLR) 例程是一个通过执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句（将 .NET 组合件作为其外部代码主体来引用）来创建的外部例程。

下列术语在 CLR 例程的上下文中很重要:

.NET Framework
一个 Microsoft 应用程序开发环境，包含 CLR 和 .NET Framework 类库（设计为提供用于开发和集成代码段的一致编程环境）。

公共语言运行时 (CLR)
所有 .NET Framework 应用程序的运行时解释器。

中间语言 (IL)
由 .NET Framework CLR 解释的已编译字节码的类型。所有 .NET 兼容语言的源代码都编译为 IL 字节码。

组合件 包含 IL 字节码的文件。这可以是库或可执行文件。

您可以使用任何可编译为 IL 组合件的语言来实现 CLR 例程。这些语言包括但不限于 Managed C++、C#、Visual Basic 以及 J#。

在开发 CLR 例程前，必须了解例程基本原理以及特定于 CLR 例程的特有功能和特征。要了解例程和 CLR 例程的更多信息，请参阅:

- 第 5 页的『使用例程的益处』
- 第 41 页的『.NET CLR 例程中的 SQL 数据类型表示』
- 第 42 页的『.NET CLR 例程中的参数』
- 第 44 页的『从 .NET CLR 过程中返回结果集』
- 第 46 页的『.NET CLR 例程限制』
- 第 55 页的『与 .NET CLR 例程相关的错误』

开发 CLR 例程很容易。有关如何开发 CLR 例程的逐步指示信息以及完整示例，请参阅:

- 第 48 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 57 页的『C# .NET CLR 过程示例』
- 第 83 页的『C# .NET CLR 函数示例』

对使用 .NET CLR 语言进行外部例程开发的支持

要使用 .NET CLR 语言开发以及成功运行外部例程，您将需要使用支持的操作系统、DB2 数据库服务器和客户机版本以及开发软件。

可以使用可由 Microsoft .NET Framework 编译成 IL 组合件的语言来实现 .NET CLR 外部例程。这些语言包括但不限于 Managed C++、C#、Visual Basic 以及 J#。

您可以在下列操作系统上开发 .NET CLR 例程:

- Windows 2000
- Windows XP（32 位和 64 位版本）
- Windows Server 2003（32 位和 64 位版本）
- Windows Server 2008（32 位和 64 位版本）

必须针对 .NET CLR 例程开发安装 V9 或更高版本的数据服务器客户机。数据库服务器必须正在运行 DB2 V9 或更高版本的数据库产品。

必须将 Microsoft .NET Framework 软件的支持版本安装至 DB2 数据库服务器所在的计算机。Microsoft .NET Framework 是单独提供的，或作为 Microsoft .NET Framework 软件开发包的一部分提供。

用于开发 .NET CLR 例程的工具

工具可以使开发 .NET CLR 例程以与 DB2 数据库交互的任务更快速、更容易。

可以在 Microsoft Visual Studio .NET 中使用下列软件提供的图形工具来开发 .NET CLR 例程:

- IBM Database Add-Ins for Microsoft Visual Studio

也可以使用随 DB2 数据库系统提供的下列命令行界面，在 DB2 数据库服务器上开发 .NET CLR 例程:

- DB2 命令行处理器 (DB2 CLP)
- DB2 命令窗口

设计 .NET CLR 例程

设计 .NET CLR 例程时，应该考虑常规外部例程设计注意事项以及 .NET CLR 特定设计注意事项。

.NET 应用程序开发的知识和经验以及对外部例程的一般了解。下列主题可以向您提供一些必备信息。

有关外部例程的功能和使用的更多信息，请参阅:

- 第 6 页的『外部例程实现』

有关 .NET CLR 例程的特征的更多信息，请参阅:

- 第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』

掌握必备知识之后，设计嵌入式 SQL 例程的主要注意事项是了解 .NET CLR 例程的特有功能和特征:

- 包含组合件，支持在 .NET CLR 例程中执行 SQL 语句 (IBM.Data.DB2)
- .NET CLR 例程中的受支持 SQL 数据类型
- .NET CLR 例程的参数
- 返回 .NET CLR 例程的结果集
- .NET CLR 例程的安全性和执行控制方式设置
- .NET CLR 例程限制

- 从 .NET CLR 过程中返回结果集

在了解 .NET CLR 特征后，请参阅第 47 页的『创建 .NET CLR 例程』。

.NET CLR 例程中的 SQL 数据类型表示

.NET CLR 例程可以引用作为例程参数的 SQL 数据类型值、用作 SQL 语句执行一部分的参数值以及用作变量的参数值，然而必须使用相应的 IBM SQL 数据类型值、IBM Data Server Provider for .NET 数据类型值以及 .NET Framework 数据类型值来确保在访问或检索这些值时不会截断或丢失数据。

对于 CREATE PROCEDURE 或 CREATE FUNCTION 语句中用来创建 .NET CLR 例程的例程参数规范，会使用 DB2 SQL 数据类型值。可以为例程参数指定大多数 SQL 数据类型，然而存在一些例外。

必须使用 IBM Data Server Provider for .NET 对象，才能指定要用作需执行的 SQL 语句一部分的参数值。使用 `DB2Parameter` 对象来表示要添加至 `DB2Command` 对象（表示 SQL 语句）的参数。指定参数的数据类型值时，必须使用 `IBM.Data.DB2Types` 名称空间中所提供的 IBM Data Server Provider for .NET 数据类型值。`IBM.Data.DB2Types` 名称空间提供类和结构来表示每种支持的 DB2 SQL 数据类型。

对于可能临时存放 SQL 数据类型值的参数和局部变量，必须使用如 IBM.Data.DB2Types 名称空间中所定义的相应 IBM Data Server Provider for .NET 数据类型。

注: 会将 `dbinfo` 结构作为参数传递至 CLR 函数和过程。也会将 CLR UDF 的暂存区和调用类型作为参数传递至 CLR 例程。有关适合于这些参数的 CLR 数据类型的信息，请参阅相关主题:

- CLR 例程中的参数

下表显示 DB2Type 数据类型、DB2 数据类型、Informix® 数据类型、Microsoft .NET Framework 类型以及 DB2Types 类和结构之间的映射。

类别	DB2Types 类和结构	DB2Type 数据类型	DB2 数据类型	Informix 数据类型	.NET 数据类型
数字	DB2Int16	SmallInt	SMALLINT	BOOLEAN 和 SMALLINT	Int16
数字	DB2Int32	Integer	INT	INTEGER、INT 和 SERIAL	Int32
数字	DB2Int64	BigInt	BIGINT	BIGINT、BIGSERIAL、INT8 和 SERIAL8	Int64
数字	DB2Real 和 DB2Real370	Real	REAL	REAL 和 SMALLFLOAT	Single
数字	DB2Double	Double	DOUBLE PRECISION	DECIMAL（≤31）和 DOUBLE PRECISION	Double
数字	DB2Double	Float	FLOAT	DECIMAL（32）和 FLOAT	Double

1. DB2 .NET 公共语言运行时例程不支持将这些数据类型用作参数。

2. 类型为 DB2Type.Xml 的 DB2ParameterClass.ParameterName 属性可以接受下列类型的变量: String、byte[]、DB2Xml 以及 XmlReader。

3. 这些数据类型只适用于 DB2 for z/OS。

4. 仅 DB2 for z/OS V9 及更高版本以及 DB2 for Linux, UNIX and Windows V9.5 及更高发行版支持此数据类型。

类别	DB2Types 类和结构	DB2Type 数据类型	DB2 数据类型	Informix 数据类型	.NET 数据类型
数字	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
数字	DB2DecimalFloat	DecimalFloat	DECFLOAT（16|34）[1、4]		Decimal
数字	DB2Decimal	Numeric	DECIMAL	DECIMAL（≤31）和 NUMERIC	Decimal
日期/时间	DB2Date	Date	DATE	DATETIME（日期精度）	Datetime
日期/时间	DB2Time	Time	TIME	DATETIME（时间精度）	TimeSpan
日期/时间	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME（时间和日期精度）	DateTime
XML	DB2Xml	Xml[2]	XML		Byte[]
字符数据	DB2String	Char	CHAR	CHAR	String
字符数据	DB2String	VarChar	VARCHAR	VARCHAR	String
字符数据	DB2String	LongVarChar[1]	LONG VARCHAR	LVARCHAR	String
二进制数据	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
二进制数据	DB2Binary	Binary[3]	BINARY		Byte[]
二进制数据	DB2Binary	VarBinary[3]	VARBINARY		Byte[]
二进制数据	DB2Binary	LongVarBinary[1]	LONG VARCHAR FOR BIT DATA		Byte[]
图形数据	DB2String	Graphic	GRAPHIC		String
图形数据	DB2String	VarGraphic	VARGRAPHIC		String
图形数据	DB2String	LongVarGraphic[1]	LONG VARGRAPHIC		String
LOB 数据	DB2Clob	Clob	CLOB	CLOB 和 TEXT	String
LOB 数据	DB2Blob	Blob	BLOB	BLOB 和 BYTE	Byte[]
LOB 数据	DB2Clob	DbClob	DBCLOB		String
行标识	DB2RowId	RowId	ROWID		Byte[]

.NET CLR 例程中的参数

.NET CLR 例程中的参数声明必须符合其中一种支持参数样式的要求，且必须遵循用于例程的特定 .NET 语言的参数关键字要求。

如果例程将使用暂存区或 dbinfo 结构，或者将具有 PROGRAM TYPE MAIN 参数接口，那么需要考虑其他详细信息。本主题提出所有 CLR 参数注意事项。

CLR 例程支持的参数样式

在创建例程时，必须在例程的 CREATE 语句的 EXTERNAL 子句中指定例程的参数样式。必须在外部 CLR 例程代码的实现中精确地反映此参数样式。CLR 例程支持下列 DB2 参数样式:

- SQL（受过程和函数支持）
- GENERAL（只受过程支持）
- GENERAL WITH NULLS（只受过程支持）
- DB2SQL（受过程和函数支持）

有关这些参数样式的更多信息，请参阅:

- 第 20 页的『外部例程的参数样式』

CLR 例程参数 null 指示符

如果为 CLR 例程选择的参数样式要求为参数指定 null 指示符，那么当参数样式调用 null 指示符的向量时，会将 null 指示符作为 System.Int16 类型值传递至 CLR 例程，或在 System.Int16[] 值中传递。

如果参数样式指定必须将 null 指示符作为单值参数传递至例程（参数样式 SQL 要求这样做），那么每个参数需要一个 System.Int16 null 指示符。

在 .NET 语言中，必须在单值参数前面加上一个关键字，以指示是通过值还是通过引用来传递参数。必须将用于例程参数的同一关键字用于相关联的 null 指示符参数。下节更详细地讨论了用来指示是通过值还是通过引用来传递自变量的关键字。

有关参数样式 SQL 及其他支持的参数样式的更多信息，请参阅:

- 第 20 页的『外部例程的参数样式』

通过值或通过引用来传递 CLR 例程参数

编译为中间语言 (IL) 字节码的 .NET 语言例程要求在参数前面加上关键字，以指示参数的特定属性，例如是通过值还是通过引用来传递参数，或者参数是仅输入参数还是仅输出参数。

参数关键字特定于 .NET 语言。例如，要使用 C# 通过引用来传递参数，参数关键字是 ref，而使用 Visual Basic 时，通过引用参数是由 byRef 关键字指示。必须使用关键字来指示例程的 CREATE 语句中所指定的 SQL 参数使用 （IN、OUT 或 INOUT）。

将参数关键字应用于 DB2 例程中的 .NET 语言例程参数时，会适用下列规则:

- 使用 C# 声明 IN 类型参数时*不能* 使用参数关键字，使用 Visual Basic 声明这些类型参数时必须使用 byVal 关键字。
- 必须使用语言特定关键字（指示通过引用传递参数）来声明 INOUT 类型参数。使用 C#，相应的关键字是 ref。使用 Visual Basic，相应的关键字是 byRef。
- 必须使用语言特定关键字（指示参数是仅输出参数）来声明 OUT 类型参数。使用 C#，使用 out 关键字。在 Visual Basic 中，必须使用 byRef 关键字来声明参数。在例程返回至调用者时，必须对仅输出参数赋值。如果例程未对仅输出参数赋值，那么在编译 .NET 例程时，将抛出错误。

下列是返回单一输出参数 language 的例程的 C# 参数样式 SQL 过程原型形式。

```
public static void Counter  (out String language,
                             out Int16  languageNullInd,
                             ref String sqlState,
                                 String funcName,
                                 String funcSpecName,
                             ref String sqlMsgString,
                                 Byte[] scratchPad,
                                 Int32  callType);
```

很明显，是由于额外的 null 指示符参数 languageNullInd（与输出参数 language 相关联）、用于传递 SQLSTATE 的参数、例程名称、例程特定名称以及可选的用户定义 SQL 错误消息，而实现了参数样式 SQL。为参数指定了参数关键字，如下所示:

- 使用 C# 时，仅输入参数不需要参数关键字。
- 使用 C# 时，“out”关键字指示变量是仅输出参数，且指示调用者尚未初始化参数值。
- 使用 C# 时，“ref”关键字指示调用者已初始化参数，且指示例程可选择性地修改此值。

请参阅关于参数传递的 .NET 语言特定文档，以了解有关该语言中的参数关键字的信息。

注: DB2 数据库系统控制所有参数的内存分配，以及维护对传入或传出例程的所有参数的 CLR 引用。

过程结果集不需要参数标记

在过程的过程声明中，将返回给调用者的结果集不需要参数标记。任何未从 CLR 存储过程内部关闭的游标语句将作为结果集传递回至其调用者。

有关 CLR 例程中结果集的更多信息，请参阅:

- 『从 .NET CLR 过程中返回结果集』

作为 CLR 参数的 Dbinfo 结构

CLR 例程通过使用 IL `dbinfo` 类来支持用于将其他数据库信息参数传入和传出例程的 `dbinfo` 结构。此类包含所有可在 C 语言 `sqludf_dbinfo` 结构中找到的元素（与字符串相关联的长度字段除外）。可以使用每个字符串的 .NET 语言 `Length` 属性来找到特定字符串的长度。

要访问 `dbinfo` 类，只需在包含例程的文件中包括 `IBM.Data.DB2` 组合件，然后在例程特征符中由使用的参数样式所指定的位置，添加类型为 `sqludf_dbinfo` 的参数。

作为 CLR 参数的 UDF 暂存区

如果为用户定义的函数请求暂存区，那么会将它传递至作为具有指定大小的 `System.Byte[]` 参数的例程。

CLR UDF 调用类型或最终调用参数

对于已请求最终调用参数的用户定义的函数或对于表函数，会将调用类型参数作为 `System.Int32` 数据类型传递至例程。

CLR 过程支持的 PROGRAM TYPE MAIN

.NET CLR 过程支持程序类型 MAIN。定义为使用程序类型 MAIN 的过程必须具有下列特征符:

```
void functionname(Int32 NumParams, Object[] Params)
```

从 .NET CLR 过程中返回结果集

您可以开发将结果集返回给调用例程或应用程序的 CLR 过程。无法从 CLR 函数 (UDF) 返回结果集。

开始之前

结果集的 .NET 表示是可从 DB2Command 对象的各种执行调用之一返回的 DB2DataReader 对象。如果 DB2DataReader 对象的 Close() 方法未在过程返回之前进行显式地调用，那么可以返回该对象。将结果集返回给调用者的顺序，与 DB2DataReader 对象的实例化顺序相同。函数定义中不需要其他参数，即可返回结果集。

了解如何创建 CLR 例程将有助于您执行以下过程（用于从 CLR 过程返回结果）中的步骤。

- 第 48 页的『从 DB2 命令窗口创建 .NET CLR 例程』

过程

要从 CLR 过程中返回结果集，请执行下列操作:

1. 在 CLR 例程的 CREATE PROCEDURE 语句中，除指定任何其他相应的语句之外，您还必须指定 DYNAMIC RESULT SETS 子句（其值等于过程将要返回的结果集数）。
2. 在过程声明中，将返回给调用者的结果集不需要参数标记。
3. 在 CLR 例程的 .NET 语言实现中，创建 DB2Connection 对象、DB2Command 对象以及 DB2Transaction 对象。DB2Transaction 对象负责回滚和落实数据库事务。
4. 将 DB2Command 对象的事务属性初始化为 DB2Transaction 对象。
5. 将字符串查询指定给 DB2Command 对象的 CommandText 属性以定义要返回的结果集。
6. 实例化 DB2DataReader 并对其指定 DB2Command 对象方法 ExecuteReader 的调用结果。会将查询的结果集包含在 DB2DataReader 对象中。
7. 在过程返回至调用者之前的任何时候，请不要执行 DB2DataReader 对象的 Close() 方法。会将仍处于打开状态的 DB2DataReader 对象作为结果集返回给调用者。

   如果在过程返回时有多个 DB2DataReader 保持处于打开状态，那么会将 DB2DataReaders 以其创建顺序返回给调用者。只将 CREATE PROCEDURE 语句中所指定数目的结果集返回给调用者。
8. 编译 .NET CLR 语言过程，并将组合件安装至 CREATE PROCEDURE 语句中的 EXTERNAL 子句所指定的位置。针对 CLR 过程执行 CREATE PROCEDURE 语句（如果尚未这样做）。
9. 在将 CLR 过程组合件安装至相应位置且成功执行 CREATE PROCEDURE 语句后，您可以使用 CALL 语句来调用该过程以查看返回给调用者的结果集。

CLR 例程的安全性和执行方式

作为数据库管理员或应用程序开发者，您可能需要保护与 DB2 外部例程相关的组合件，以限制运行时对例程所执行的操作，从而避免令人厌烦的篡改。DB2 .NET 公共语言运行时 (CLR) 例程支持指定执行控制方式，以标识运行时允许例程执行的操作类型。在运行时，DB2 数据库系统可以检测例程尝试执行的操作是否超出对例程指定的执行控制方式的作用域，这有助于确定组合件是否遭破坏。

要设置 CLR 例程的执行控制方式，请在例程的 CREATE 语句中指定可选的 EXECUTION CONTROL 子句。有效方式如下:

- SAFE
- FILEREAD

- FILEWRITE
- NETWORK
- UNSAFE

要在现有 CLR 例程中修改执行控制方式，请执行 ALTER PROCEDURE 或 ALTER FUNCTION 语句。

如果未对 CLR 例程指定 EXECUTION CONTROL 子句，那么缺省情况下，将使用限制性最强的执行控制方式来运行 CLR 例程: SAFE。使用此执行控制方式来创建的例程只能访问由数据库管理器控制的资源。限制性较弱的执行控制方式允许例程访问文件（FILEREAD 或 FILEWRITE），或执行网络操作，例如访问 Web 页面 (NETWORK)。执行控制方式 UNSAFE 指定不对例程的行为实施任何限制。以 UNSAFE 执行控制方式定义的例程可以执行二进制代码。

这些方式表示可允许操作的层次结构，并且高级方式包括层次结构中其下所允许的操作。例如，执行控制方式 NETWORK 允许例程访问因特网上的 Web 页面、读/写文件以及访问由数据库管理器控制的资源。建议使用限制性尽可能强的执行控制方式，且避免使用 UNSAFE 方式。

如果 DB2 数据库系统在运行时检测到 CLR 例程正尝试在其执行控制方式作用域外部执行操作，那么 DB2 数据库系统将返回错误 (SQLSTATE 38501)。

只能为 LANGUAGE CLR 例程指定 EXECUTION CONTROL 子句。EXECUTION CONTROL 子句的适用性作用域限制于 .NET CLR 例程本身，且不扩展至它可能调用的任何其他例程。

请参阅相应例程类型的 CREATE 语句的语法，以取得支持的执行控制方式的完整描述。

.NET CLR 例程限制

适用于所有外部例程或特定例程类（过程或 UDF）的一般实现限制也适用与 CLR 例程。另外，还有一些 CLR 例程所特有的限制。这些限制列示如下。

不支持带有 LANGUAGE CLR 子句的 CREATE METHOD 语句

不能为引用了 CLR 组合件的 DB2 数据库结构化类型创建外部方法。不支持使用指定了值为 CLR 的 LANGUAGE 子句的 CREATE METHOD 语句。

不能将 CLR 过程作为 NOT FENCED 过程实现

CLR 过程不能作为未受防护过程运行。用于创建 CLR 过程的 CREATE PROCEDURE 语句不能指定 NOT FENCED 子句。

EXECUTION CONTROL 子句会限制例程中包含的逻辑

EXECUTION CONTROL 子句和相关联的值确定了可以在 .NET CLR 例程中执行的逻辑和操作的类型。缺省情况下，EXECUTION CONTROL 子句值设置为 SAFE。对于读取文件、写入文件或访问因特网的例程逻辑而言，必须对 EXECUTION CONTROL 子句指定限制性没那么强的非缺省值。

在 CLR 例程中，最大十进制精度为 29，最大十进制小数位数为 28

在 DB2 数据库中，DECIMAL 数据类型表示为精度 32 位且小数位数 28 位。.NET CLR System.Decimal 数据类型限制为精度 29 位且小数位数 28 位。因此，DB2 外部 CLR 例程不能将大于 (2^96)-1（可以使用 29 位精度和 28 位小数表示的最大值）的值赋予 System.Decimal 数据类型。如果发生这样的赋值，那么 DB2 数据库系统将发出运行时错误 (SQLSTATE 22003, SQLCODE -413)。执行用于创建例程的 CREATE 语句时，如果将 DECIMAL 数据类型参数定义为小数位数大于 28，那么 DB2 数据库系统将发出错误 (SQLSTATE 42613, SQLCODE -628)。

如果您要求例程处理最大精度和小数位数受 DB2 数据库系统支持的十进制值，那么可以使用 Java 之类的另一编程语言来实现外部例程。

在 CLR 例程中不受支持的数据类型

在 CLR 例程中，不支持下列 DB2 SQL 数据类型:

- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- LONG GRAPHIC
- ROWID

在 64 位实例上运行 32 位 CLR 例程

CLR 例程不能在 64 位实例上运行，这是因为，目前无法将 .NET Framework 安装在 64 位操作系统上。

不支持使用 .NET CLR 来实现安全性插件

不支持使用 .NET CLR 来编译和链接安全性插件库的源代码。

创建 .NET CLR 例程

创建 .NET CLR 例程包括执行用于在 DB2 数据库服务器中定义该例程的 CREATE 语句以及开发对应例程定义的例程实现。

开始之前

- 复审第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』。
- 确保您可以访问 DB2 版本 9 服务器（其中包括实例和数据库）。
- 确保操作系统处于 DB2 数据库产品支持的版本级别。
- 确保 Microsoft .NET 开发软件为支持 .NET CLR 例程开发的版本级别。请参阅第 39 页的『对使用 .NET CLR 语言进行外部例程开发的支持』。
- 有权执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句。

有关与 CLR 例程相关联的限制列表，请参阅:

- 第 46 页的『.NET CLR 例程限制』

关于此任务

可用来创建 .NET CLR 例程的方法如下所示:

- 使用 IBM Database Add-Ins for Microsoft Visual Studio 随附的图形工具
- 使用 DB2 命令窗口

通常，使用 IBM Database Add-Ins for Microsoft Visual Studio 创建 .NET CLR 例程最简单。如果无法使用此软件，那么 DB2 命令窗口可通过命令行界面提供类似的支持。

从下列其中一个界面创建 .NET CLR 例程:

过程

- Visual Studio .NET（如果也安装了 IBM Database Add-Ins for Microsoft Visual Studio）。在安装此 Add-In 后，集成至 Visual Studio .NET 的图形工具可用于创建 .NET CLR 可以在 DB2 数据库服务器中工作的 .NET CLR 例程。
- DB2 命令窗口

下一步做什么

要从 DB2 命令窗口创建 .NET CLR 例程，请参阅:

- 『从 DB2 命令窗口创建 .NET CLR 例程』

从 DB2 命令窗口创建 .NET CLR 例程

使用创建任何外部例程的方法来创建对中间语言组合件进行引用的过程和函数。

开始之前

- 了解 CLR 例程实现。要了解 CLR 例程的常规信息以及 CLR 功能的信息，请参阅:
  - 第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』
- 数据库服务器必须正在运行支持 Microsoft .NET Framework 的 Windows 操作系统。
- 必需将 Microsoft .NET Framework 软件的支持版本安装到该服务器上。.NET Framework 是单独提供或作为 Microsoft .NET Framework 软件开发包的一部分提供。
- 必须安装支持的 DB2 数据库产品或 IBM 数据服务器客户机。请参阅 DB2 数据库产品的安装要求。
- 有权对外部例程执行 CREATE 语句。有关执行 CREATE PROCEDURE 语句或 CREATE FUNCTION 语句所需的特权，请参阅相应语句的详细信息。

限制

有关与 CLR 例程相关联的限制列表，请参阅:

- 第 46 页的『.NET CLR 例程限制』

关于此任务

在下列情况下，您不妨选择使用 .NET 语言来实现外部例程:

- 您需要在例程中封装复杂逻辑，以访问数据库或在数据库外部执行操作。
- 您要求从下列任何项调用封装的逻辑: 多个应用程序、CLP、另一个例程（过程、函数 (UDF) 或方法）或触发器。
- 您最擅长使用 .NET 语言来编写此逻辑。

过程

1. 使用 CLR 所支持的任何语言来编写例程逻辑。
   - 有关 .NET CLR 例程和 .NET CLR 例程功能的常规信息，请参阅“开始之前”部分中所引用的主题。
   - 如果您的例程将执行 SQL，请使用或导入 `IBM.Data.DB2` 组合件。
   - 正确地使用映射至 DB2 SQL 数据类型的数据类型来声明主变量和参数。有关 DB2 和 .NET 数据类型之间的数据类型映射的信息，请参阅:
     – 第 41 页的『.NET CLR 例程中的 SQL 数据类型表示』
   - 必须使用 DB2 所支持的其中一种参数样式，且根据 .NET CLR 例程的参数要求来声明参数和参数 null 指示符。此外，会将 UDF 的暂存区以及 DBINFO 类作为参数传递至 CLR 例程。有关参数和原型声明的更多信息，请参阅:
     – 第 42 页的『.NET CLR 例程中的参数』
   - 如果例程是过程且您需要将结果集返回给例程调用者，那么您不需要该结果集的任何参数。有关返回 CLR 例程的结果集的更多信息，请参阅:
     – 第 44 页的『从 .NET CLR 过程中返回结果集』
   - 根据需要设置例程返回值。CLR 标量函数要求在返回前设置返回值。CLR 表函数要求指定返回码作为表函数每个调用的输出参数。CLR 过程不会返回任何返回值。
2. 将代码构建为由 CLR 执行的中间语言 (IL) 组合件。 有关如何构建访问 DB2 数据库的 CLR .NET 例程的信息，请参阅以下主题:
   - *开发 ADO.NET 和 OLE DB 应用程序* 中的『构建公共语言运行时 (CLR) .NET 例程』
3. 将组合件复制到数据库服务器上的 DB2 *function* 目录。建议将 DB2 例程的相关联组合件或库存储至函数目录。要了解有关函数目录的更多信息，请参阅下列任一语句的 EXTERNAL 子句: CREATE PROCEDURE 或 CREATE FUNCTION。

   如果需要，您可以将组合件复制到服务器上的另一个目录，但是为了成功调用例程，您必须记下组合件的标准路径名，因为下一步需要该标准路径名。
4. 动态或静态地执行适合于例程类型的 SQL 语言 CREATE 语句: CREATE PROCEDURE 或 CREATE FUNCTION。
   - 使用值 `CLR` 来指定 `LANGUAGE` 子句。
   - 使用支持的参数样式（在例程代码中实现）的名称来指定 `PARAMETER STYLE` 子句。
   - 使用组合件（将通过下列其中一个值与例程相关联）的名称来指定 `EXTERNAL` 子句:
     – 例程组合件的标准路径名。
     – 例程组合件的相对路径名（相对于函数目录）。

     缺省情况下，DB2 数据库系统将在函数目录中按名称查找组合件，除非在 EXTERNAL 子句中指定了库的标准或相对路径名。

     当执行 CREATE 语句时，如果 DB2 数据库系统找不到在 EXTERNAL 子句中指定的组合件，那么将接收到错误 (SQLCODE -20282)，原因码为 1。
   - 使用整数值（等价于例程可能返回的最大结果集数）来指定 DYNAMIC RESULT SETS 子句。

- 不能对 CLR 过程指定 NOT FENCED 子句。缺省情况下，CLR 过程作为 FENCED 过程来执行。

构建 .NET CLR 例程代码

在编写 .NET CLR 例程实现代码后，必须先构建，然后才能部署例程组合件并调用例程。构建 .NET CLR 例程所需的步骤类似于构建任何外部例程所需的步骤，然而存在某些差别。

过程

可以使用三种方法来构建 .NET CLR 例程:

- 使用 IBM Database Add-Ins for Microsoft Visual Studio 随附的图形工具
- 使用 DB2 样本批处理文件
- 从 DB2 命令窗口输入命令

例程的 DB2 样本构建脚本和批处理文件设计用于通过缺省的支持编译器，针对特定操作系统构建 DB2 样本例程（过程和用户定义的函数）以及用户创建的例程。
C# 和 Visual Basic 分别具有一组 DB2 样本构建脚本和批处理文件。通常，使用图形工具或构建脚本（可根据需要轻松修改）来构建 .NET CLR 例程非常容易，但是，如果也知道如何从 DB2 命令窗口构建例程，往往很有帮助。

使用样本构建脚本来构建 .NET 公共语言运行时 (CLR) 例程代码

构建 .NET 公共语言运行时 (CLR) 例程源代码是创建 .NET CLR 例程的子任务。可以使用 DB2 样本批处理文件来简单快捷地完成此任务。

可以将样本构建脚本用于带有或不带 SQL 语句的源代码。构建脚本处理下列操作: 将构建的组合件编译、链接以及部署至 `function` 目录。

作为替代选择，您可以简化构建 .NET CLR 例程代码的任务；要完成简化，请使用 Visual Studio .NET 或使用 DB2 样本构建脚本来手动执行这些步骤。请参阅:

- 使用 Visual Studio .NET 来构建 .NET 公共语言运行时 (CLR) 例程
- 使用 DB2 命令窗口来构建 .NET 公共语言运行时 (CLR) 例程

用于构建 C# 和 Visual Basic .NET CLR 例程的编程语言特定样本构建脚本称为 **bldrtn**。这些脚本及其用于构建的样本程序都位于 DB2 目录，如下所示:

- 对于 C: `sqllib/samples/cs/`
- 对于 C++: `sqllib/samples/vb/`

可以使用 **bldrtn** 脚本来构建包含过程和用户定义的函数的源代码文件。脚本完成下列操作:

- 建立与用户指定数据库的连接
- 编译和链接源代码以生成带有 .DLL 文件后缀的组合件
- 将组合件复制到数据库服务器上的 DB2 函数目录

bldrtn 脚本接受两个自变量:

- 不带任何文件后缀的源代码文件名
- 将与其建立连接的数据库的名称

数据库参数是可选的。如果未提供任何数据库名称，那么程序将使用缺省样本数据库。因为必须在数据库所在的相同实例上构建例程，所以用户标识和密码不需要任何自变量。

先决条件

- 必须满足必需的 .NET CLR 例程操作系统以及开发软件先决条件。请参阅: "对 .NET CLR 例程开发的支持"。
- 包含一个或多个例程实现的源代码文件。
- 当前 DB2 实例中将创建例程的数据库的名称。

过程

要构建包含一个或多个例程代码实现的源代码文件，请执行以下步骤。

1. 打开 DB2 命令窗口。
2. 将源代码文件复制到 **bldrtn** 脚本文件所在的相同目录。
3. 如果将在样本数据库中创建例程，请输入构建脚本名称，后跟不带 .cs 或 .vb 文件扩展名的源代码文件名:

   ```
   bldrtn file-name
   ```

   如果将在另一个数据库中创建例程，请输入构建脚本名称、不带任何文件扩展名的源代码文件名以及数据库名:

   ```
   bldrtn file-name database-name
   ```

   脚本会编译和链接源代码并产生组合件。然后，脚本会将组合件复制到数据库服务器上的函数目录
4. 如果这不是您首次构建包含例程实现的源代码文件，请停止然后重新启动数据库以确保 DB2 数据库系统使用的是新版本的共享库。要完成此任务，请在命令行中输入 **db2stop**，接着输入 **db2start**。

在成功构建例程共享库并将其部署至数据库服务器上的函数目录后，您应该完成与创建 C 和 C++ 例程的任务相关联的步骤。

创建 .NET CLR 例程的过程包括针对源代码文件中实现的每个例程执行 CREATE 语句的步骤。在完成创建例程后，您可以调用例程。

从 DB2 命令窗口构建 .NET 公共语言运行时 (CLR) 例程代码

构建 .NET CLR 例程源代码是创建 .NET CLR 例程的子任务。可以手动从 DB2 命令窗口完成此任务。可以遵循相同的过程，而不论例程代码中是否存在 SQL 语句。任务步骤包括将使用 .NET CLR 支持编程语言编写的源代码，编译至文件后缀为 .DLL 的组合件。

开始之前

作为替代选择，您可以简化构建 .NET CLR 例程代码的任务；要完成简化，请使用 Visual Studio .NET 或使用 DB2 样本构建脚本。请参阅:

- 使用 Visual Studio .NET 来构建 .NET 公共语言运行时 (CLR) 例程
- 使用样本构建脚本来构建 .NET 公共语言运行时 (CLR) 例程
- 已满足必需的操作系统和 .NET CLR 例程开发软件先决条件。

- 使用支持的 .NET CLR 编程语言编写的源代码（包含一个或多个 .NET CLR 例程实现）。
- 当前 DB2 实例中将创建例程的数据库的名称。
- 构建 .NET CLR 例程时所需的操作特定编译和链接选项。

过程

要构建包含一个或多个 .NET CLR 例程代码实现的源代码文件:

1. 打开 DB2 命令窗口。
2. 浏览至包含源代码文件的目录。
3. 将建立与数据库（将在其中创建例程）的连接。
4. 编译源代码文件。
5. 链接源代码文件以生成共享库。这需要使用一些特定于 DB2 数据库系统的编译和链接选项。
6. 将文件后缀为 .DLL 的组合件文件复制到数据库服务器上的 DB2 函数目录。
7. 如果这不是您首次构建包含例程实现的源代码文件，请停止然后重新启动数据库以确保 DB2 数据库系统使用的是新版本的共享库。 可通过先发出 **db2stop** 命令然后发出 **db2start** 命令来完成此操作。

结果

在成功构建和部署例程库后，您应该完成与创建 .NET CLR 例程的任务相关联的步骤。创建 .NET CLR 例程的过程包括针对源代码文件中实现的每个例程执行 CREATE 语句的步骤。也必须先完成此步骤，才能调用例程。

示例

下列示例演示如何重新构建 .NET CLR 源代码文件。会显示 Visual Basic 代码文件 myVBfile.vb（包含例程实现）以及 C# 代码文件 myCSfile.cs 的步骤。在 Windows 2000 操作系统上构建了这些例程（使用 Microsoft .NET Framework 1.1 来生成 64 位组合件）。

1. 打开 DB2 命令窗口。
2. 浏览至包含源代码文件的目录。
3. 将建立与数据库（将在其中创建例程）的连接。

   ```
   db2 connect to database-name
   ```

4. 使用建议的编译和链接选项来编译源代码文件（其中，$DB2PATH 是 DB2 实例的安装路径。在运行命令之前，请替换此值）:

   ```
   C# example
   ==================
   csc /out:myCSfile.dll /target:library
       /reference:$DB2PATH%\bin\netf11\IBM.Data.DB2.dll myCSfile.cs

   Visual Basic example
   ====================
   vbc /target:library /libpath:$DB2PATH\bin\netf11
       /reference:$DB2PATH\bin\netf11\IBM.Data.DB2.dll
       /reference:System.dll
       /reference:System.Data.dll myVBfile.vb
   ```

如果发生任何错误，那么编译器将生成输出。此步骤会生成名为 myfile.exp 的导出文件。

5. 将共享库复制到数据库服务器上的 DB2 函数目录。

```
C# example
====================
rm -f ~HOME/sqllib/function/myCSfile.DLL
cp myCSfile $HOME/sqllib/function/myCSfile.DLL

Visual Basic example
====================
rm -f ~HOME/sqllib/function/myVBfile.DLL
cp myVBfile $HOME/sqllib/function/myVBfile.DLL
```

此步骤确保例程库位于 DB2 将在其中查找例程库的缺省目录。请参阅有关创建 .NET CLR 例程的主题，以了解有关如何部署例程库的更多信息。

6. 停止然后重新启动数据库，因为此步骤会重新构建先前构建的例程源代码文件。

```
db2stop
db2start
```

通常，可以使用操作特定样本构建脚本非常轻松地构建 .NET CLR 例程，这些脚本也可用作如何从命令行构建例程的参考。

CLR .NET 例程的编译和链接选项

在 Windows 上使用 Microsoft Visual Basic .NET 编译器或 Microsoft C# 编译器来构建公共语言运行时 (CLR) .NET 例程时，使用 DB2 中提供的编译和链接选项，如 samples\.NET\cs\bldrtn.bat 和 samples\.NET\vb\bldrtn.bat 批处理文件中所示。

使用 Microsoft C# 编译器时 bldrtn 的编译和链接选项:

使用 Microsoft C# 编译器时的编译和链接选项:

csc Microsoft C# 编译器。

/out:%1.dll /target:library
将动态链接库作为存储过程组合件 DLL 进行输出。

/debug 使用调试器。

/lib: "%DB2PATH%"\bin\netf20
对 .NET Framework V2.0 使用库路径。

应用程序支持多个 .NET Framework 版本: V2.0、V3.0 和 V3.5。每个版本都有一个动态链接库。对于 .NET Framework V1.1，请使用 "%DB2PATH%"\bin\netf11 子目录。对于 .NET Framework V2.0、V3.0 和 V3.5，请使用 "%DB2PATH%"\bin\netf20 子目录。

/reference:IBM.Data.DB2.dll
对 IBM Data Server Provider for .NET 使用 DB2 动态链接库

请参阅编译器文档，以了解其他编译器选项。

使用 Microsoft Visual Basic .NET 编译器时 bldrtn 的编译和链接选项:

vbc Microsoft Visual Basic .NET 编译器。

/out:%1.dll /target:library
将动态链接库作为存储过程组合件 DLL 进行输出。

/debug 使用调试器。

/libpath:"%DB2PATH%"\bin\netf20
对 .NET Framework V2.0 使用库路径。

应用程序支持多个 .NET Framework 版本: V2.0、V3.0 和 V3.5。每个版本都有一个动态链接库。对于 .NET Framework V1.1，请使用 "%DB2PATH%"\bin\netf11 子目录。对于 .NET Framework V2.0、V3.0 和 V3.5，请使用 "%DB2PATH%"\bin\netf20 子目录。

/reference:IBM.Data.DB2.dll
对 IBM Data Server Provider for .NET 使用 DB2 动态链接库。

/reference:System.dll
引用 Microsoft Windows 系统动态链接库。

/reference:System.Data.dll
引用 Microsoft Windows 系统数据动态链接库。

请参阅编译器文档，以了解其他编译器选项。

调试 .NET CLR 例程

如果无法创建例程或调用例程，或者在调用时例程的性能或性能不符合预期，那么可能需要调试 .NET CLR 例程。

关于此任务

调试 .NET CLR 例程时，请考虑下列事项:

过程

- 验证是否使用支持的操作系统来开发 .NET CLR 例程。
- 验证是否使用支持的 DB2 数据库服务器和 DB2 客户机来开发 .NET CLR 例程。
- 验证是否使用支持的 Microsoft .NET Framework 开发软件。
- 如果创建例程失败，请执行下列操作:
  - 验证用户是否具有必需的权限和特权来执行 CREATE PROCEDURE 或 CREATE FUNCTION 语句。
- 如果调用例程失败，请执行下列操作:
  - 验证用户是否有权执行例程。如果发生错误（SQLCODE -551，SQLSTATE 42501），那么这可能是因为调用者对例程不具有 EXECUTE 特权。任何具有 SECADM 权限或 ACCESSCTRL 权限的用户，或任何对例程具有 EXECUTE WITH GRANT OPTION 特权的用户都可以授予此特权。
  - 验证在例程的 CREATE 语句中使用的例程参数特征符，是否匹配例程实现中的例程参数特征符。
  - 验证例程实现中使用的数据类型，是否与 CREATE 语句的例程参数特征符中指定的数据类型兼容。
  - 验证在例程实现中，用来指示方法（必须依据该方法传递参数: 通过值或通过引用）的 .NET CLR 语言特定关键字是否有效。

- 验证在 CREATE PROCEDURE 或 CREATE FUNCTION 语句的 EXTERNAL 子句中指定的值，是否匹配包含例程实现的 .NET CLR 组合件，在安装 DB2 数据库服务器的计算机文件系统上的位置。
- 如果例程是函数，请验证是否已在例程实现中正确地编程所有适用的调用类型。如果在例程中定义了 FINAL CALL 子句，那么这特别重要。

• 如果例程的行为不符合预期，请执行下列操作:
- 修改例程以将诊断信息输出至位于全局可访问目录中的文件。无法从 .NET CLR 例程将诊断信息输出至屏幕。请不要将输出定向至 DB2 数据库管理器或 DB2 数据库所使用目录中的文件。
- 要在本地调试例程，请编写简单 .NET 应用程序以直接调用例程入口点。有关如何使用 Microsoft Visual Studio .NET 中的调试功能的信息，请参阅 Microsoft Visual Studio .NET 编译器文档。

结果

有关与 .NET CLR 例程创建和调用相关的常见错误的更多信息，请参阅:

• 『与 .NET CLR 例程相关的错误』

与 .NET CLR 例程相关的错误

虽然外部例程共享一个通用实现，但是可能会发生一些特定于 CLR 例程的 DB2 数据库系统错误。

此参考按 SQLCODE 或行为列出最常见的 .NET CLR 相关错误以及一些调试建议。与例程相关的 DB2 数据库系统错误可按如下所示进行分类:

例程创建时间错误
执行例程的 CREATE 语句时发生的错误。

例程运行时错误
例程调用或执行期间发生的错误。

不管 DB2 数据库系统抛出 DB2 例程相关错误的时间为何，错误消息文本都会详细说明错误原因以及用户为解决问题而应该执行的操作。可以在 **db2diag** 诊断日志文件中找到其他例程错误场景信息。

CLR 例程创建时间错误

SQLCODE -451, SQLSTATE 42815
如果尝试执行 CREATE TYPE 语句，但该语句包含使用值 CLR 来指定 LANGUAGE 子句的外部方法声明，那么会抛出此错误。目前，无法为引用 CLR 组合件的结构化类型创建 DB2 外部方法。更改 LANGUAGE 子句以便它为方法指定支持的语言，然后使用该备用语言来实现方法。

SQLCODE -449, SQLSTATE 42878
CLR 例程的 CREATE 语句在 EXTERNAL NAME 子句中包含的库或函数标识未经有效地格式化。对于语言 CLR，EXTERNAL 子句值必须专门采用以下格式: '<a>:<b>!<c>'，如下所示:

- <a> 是类所在的 CLR 组合件文件。
- <b> 是要调用的方法所在的类。
- <c> 是要调用的方法。

在单引号、对象标识和分隔字符之间不允许使用前导或尾部空白字符（例如，' <a> ! <b> ' 无效）。但是，如果平台允许，那么路径名和文件名可以包含空格。对于所有文件名，可以使用短格式文件名（示例: math.dll）或标准路径名（示例: d:\udfs\math.dll）来指定文件。如果使用短格式文件名、平台是 UNIX 或例程是 LANGUAGE CLR 例程，那么该文件必须位于函数目录中。如果平台是 Windows 且例程不是 LANGUAGE CLR 例程，那么该文件必须位于系统 PATH 中。应该始终在文件名中包含文件扩展名，示例: .a（在 UNIX 上）和 .dll（在 Windows 上）。

CLR 例程运行时错误

SQLCODE -20282, SQLSTATE 42724，原因码 1

找不到例程 CREATE 语句中 EXTERNAL 子句所指定的外部组合件。

- 检查 EXTERNAL 子句是否指定正确的例程组合件名称，以及组合件是否位于指定的位置。如果 EXTERNAL 子句未指定期望组合件的标准路径名，那么 DB2 数据库系统假定所提供的路径名是组合件的相对路径名（相对于 DB2 数据库系统函数目录）。

SQLCODE -20282, SQLSTATE 42724，原因码 2

在例程 CREATE 语句的 EXTERNAL 子句所指定的位置中找到组合件，但是在组合件中找不到与 EXTERNAL 子句中所指定的类相匹配的类。

- 检查 EXTERNAL 子句中所指定的组合件名称是否是例程的正确组合件，以及它是否存在于指定的位置。
- 检查 EXTERNAL 子句中所指定的类名是否是正确的类名，以及它是否存在于指定的组合件。

SQLCODE -20282, SQLSTATE 42724，原因码 3

在例程 CREATE 语句的 EXTERNAL 子句所指定的位置中找到组合件，该组合件具有正确匹配的类定义，但是例程方法特征符与例程 CREATE 语句中所指定的例程特征符不匹配。

- 检查 EXTERNAL 子句中所指定的组合件名称是否是例程的正确组合件，以及它是否存在于指定的位置。
- 检查 EXTERNAL 子句中所指定的类名是否是正确的类名，以及它是否存在于指定的组合件。
- 检查参数样式实现是否与例程 CREATE 语句中所指定的参数样式匹配。
- 检查参数实现的顺序是否与例程 CREATE 语句中的参数声明顺序匹配，以及它是否遵循参数样式的额外参数要求。
- 检查是否已正确地将 SQL 参数数据类型映射至 CLR .NET 支持数据类型。

SQLCODE -4301, SQLSTATE 58004，原因码 5 或 6

尝试启动 .NET 解释器或与其通信时发生错误。DB2 数据库系统无法装入从属 .NET 库 [原因码 5]，或调用 .NET 解释器失败 [原因码 6]。

- 确保 DB2 实例已正确配置为运行 .NET 过程或函数（mscoree.dll 必须存在于系统 PATH）。确保 db2clr.dll 存在于 sqllib/bin 目录，以及已将 IBM.Data.DB2 安装至全局组合件高速缓存。如果这些都不存在，请确保已将 .NET Framework V1.1 或更高版本安装至数据库服务器，且该数据库服务器正在运行 DB2 版本 8.2 或更高发行版。

SQLCODE -4302, SQLSTATE 38501

执行例程时、准备执行例程时或在执行例程后发生未处理的异常。导致此异常的原因可能是例程逻辑编程错误未处理，或发生内部处理错误。对于此类型的错误，会将 .NET 堆栈回溯（指示发生未处理异常的位置）写入 db2diag 日志文件。

如果例程尝试执行的操作，超出例程的指定执行方式所允许的操作范围，那么也会发生此错误。在此情况下，将在 db2diag 日志文件中记录一个条目，用以专门指示发生异常的原因是执行控制违例。此外，还将包含异常堆栈回溯（指示发生违例的位置）。

确定例程的组合件是否受损或最近是否进行了修改。如果对例程进行的修改是有效的，那么可能会发生此问题，因为例程的 EXECUTION CONTROL 方式已不再设为适合于所更改逻辑的方式。如果您确信组合件未遭篡改，那么可以根据情况使用 ALTER PROCEDURE 或 ALTER FUNCTION 语句来修改例程的执行方式。请参阅下列主题，以了解更多信息:

- 第 45 页的『CLR 例程的安全性和执行方式』

.NET CLR 例程示例

开发 .NET CLR 例程时，参考示例以获取 CREATE 语句的含义和 .NET CLR 例程代码的形式会有所帮助。

关于此任务

下列主题包含 .NET CLR 过程和函数（其中包括标量函数和表函数）的示例:

.NET CLR 过程

- Visual Basic .NET CLR 过程示例
- C# .NET CLR 过程示例

.NET CLR 函数

- Visual Basic .NET CLR 函数示例
- C# .NET CLR 函数示例

C# .NET CLR 过程示例

在了解过程（又称为存储过程）的基本原理，以及 .NET 公共语言运行时例程的基础知识之后，您可以开始在应用程序中使用 CLR 过程。

开始之前

在使用 CLR 过程示例之前，您不妨阅读下列概念主题:

- 第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 48 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 构建公共语言运行时 (CLR) .NET 例程

关于此任务

本主题包含使用 C# 来实现的 CLR 过程示例；本示例说明支持的参数样式、传递参数（其中包括 dbinfo 结构）以及如何返回结果集等。要获取使用 C# 编写的 CLR UDF 示例，请参阅:

- 第 83 页的『C# .NET CLR 函数示例』

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

过程

在创建您自己的 C# CLR 过程时，请使用下列示例作为参考:

- 58
- 58
- 59
- 61
- 62
- 62
- 63

示例

C# 外部代码文件

下列示例显示各种 C# 过程实现。每个示例都包含两个部分: CREATE PROCEDURE 语句以及过程的外部 C# 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 C# 源文件称为 gwenProc.cs 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

在此文件顶部指示文件包含。如果此文件中的任何过程包含 SQL，那么将需要 IBM.Data.DB2 包含。此文件中有一个名称空间声明，以及一个包含过程的类 empOps。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。

必须记下此文件的名称、名称空间以及包含给定过程实现的类的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 可以找到 CLR 过程的组合件和类。

示例 1: C# 参数样式 GENERAL 过程

本示例显示下列内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL 过程的 C# 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据职员薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE setEmpBonusGEN(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
DYNAMIC RESULT SETS 0
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGEN' ;
```

```
public static void SetEmpBonusGEN(    String empID,
                                   ref Decimal bonus,
                                   out String empName)
{
   // Declare local variables
   Decimal salary = 0;

   DB2Command myCommand = DB2Context.GetCommand();
   myCommand.CommandText =
                    "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                  + "FROM EMPLOYEE "
                  + "WHERE EMPNO = '" + empID + '";

   DB2DataReader reader = myCommand.ExecuteReader();

   if (reader.Read())  // If employee record is found
   {
      // Get the employee's full name and salary
      empName = reader.GetString(0) + " " +
                reader.GetString(1) + ". " +
                reader.GetString(2);

      salary = reader.GetDecimal(3);

      if (bonus == 0)
      {
         if (salary > 75000)
         {
            bonus = salary * (Decimal)0.025;
         }
         else
         {
            bonus = salary * (Decimal)0.05;
         }
      }
   }
   else  // Employee not found
   {
      empName = "";  // Set output parameter
   }

   reader.Close();
}
```

示例 2: C# 参数样式 GENERAL WITH NULLS 过程

本示例显示下列内容:

- 参数样式 GENERAL WITH NULLS 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL WITH NULLS 过程的 C# 代码

此过程接受职员 ID 和当前奖金金额作为输入。如果输入参数不为 NULL，那么将检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到职员数据，那么将返回一个 NULL 字符串和一个整数。

```
CREATE PROCEDURE SetEmpbonusGENNULL(IN empID CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusGENNULL
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
EXECUTION CONTROL SAFE
FENCED
THREADSAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'
;
```

```
 public static void SetEmpBonusGENNULL(    String empID,
                                       ref Decimal bonus,
                                       out String empName,
                                           Int16[] NullInds)
 {
    Decimal salary = 0;
    if (NullInds[0] == -1) // Check if the input is null
    {
      NullInds[1] = -1;    // Return a NULL bonus value
      empName = "";        // Set output value
      NullInds[2] = -1;    // Return a NULL empName value
    }
    else
    {
       DB2Command myCommand = DB2Context.GetCommand();
        myCommand.CommandText =
                      "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY "
                    + "FROM EMPLOYEE "
                    + "WHERE EMPNO = '" + empID + "'";
       DB2DataReader reader = myCommand.ExecuteReader();

       if (reader.Read())  // If employee record is found
       {
          // Get the employee's full name and salary
          empName = reader.GetString(0) + " "
          +
                    reader.GetString(1) + ". " +
                    reader.GetString(2);
          salary = reader.GetDecimal(3);

          if (bonus == 0)
          {
            if (salary > 75000)
            {
               bonus = salary * (Decimal)0.025;
               NullInds[1] = 0; // Return a non-NULL value
            }
            else
            {
               bonus = salary * (Decimal)0.05;
               NullInds[1] = 0; // Return a non-NULL value
            }
          }
       }
       else  // Employee not found
       {
          empName = "*sdq;;         // Set output parameter
```

```
            NullInds[2] = -1;     // Return a NULL value
         }

         reader.Close();
      }
   }
```

示例 3: C# 参数样式 SQL 过程

本示例显示下列内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 参数样式 SQL 过程的 C# 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE SetEmpbonusSQL(IN empID CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpbonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!SetEmpBonusSQL' ;
```

```
 public static void SetEmpBonusSQL(    String empID,
                                    ref Decimal bonus,
                                    out String empName,
                                        Int16  empIDNullInd,
                                    ref Int16  bonusNullInd,
                                    out Int16  empNameNullInd,
                                    ref string sqlStateate,
                                        string funcName,
                                        string specName,
                                    ref string sqlMessageText)
 {
    // Declare local host variables
    Decimal salary eq; 0;

    if (empIDNullInd == -1) // Check if the input is null
    {
       bonusNullInd = -1;   // Return a NULL bonus value
       empName = "";
       empNameNullInd = -1; // Return a NULL empName value
    }
    else
    {
       DB2Command myCommand = DB2Context.GetCommand();
       myCommand.CommandText =
                        "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY
                        "
                      + "FROM EMPLOYEE "
                      + "WHERE EMPNO = '" + empID + "'";

       DB2DataReader reader = myCommand.ExecuteReader();

       if (reader.Read())  // If employee record is found
       {
          // Get the employee's full name and salary
          empName = reader.GetString(0) + " "
          +
```

```
            reader.GetString(1) + ". " +
            reader.GetString(2);
            empNameNullInd = 0;
            salary = reader.GetDecimal(3);

            if (bonus == 0)
            {
               if (salary > 75000)
               {
                  bonus = salary * (Decimal)0.025;
                  bonusNullInd = 0;  // Return a non-NULL value
               }
               else
               {
                  bonus = salary * (Decimal)0.05;
                  bonusNullInd = 0;  // Return a non-NULL value
               }
            }
         }
         else  // Employee not found
         {
            empName = "";  // Set output parameter
            empNameNullInd = -1;  // Return a NULL value
         }

         reader.Close();
      }
   }
```

示例 4: 返回结果集的 C# 参数样式 GENERAL 过程

本示例显示下列内容:

- 返回结果集的外部 C# 过程的 CREATE PROCEDURE 语句
- 返回结果集的参数样式 GENERAL 过程的 C# 代码

此过程接受表名作为参数。它返回一个包含输入参数所指定的全部表行的结果集。通过为过程返回时打开的给定查询结果集保留一个 DB2DataReader 来实现此目的。具体地说，如果未执行 reader.Close()，那么将返回结果集。

```
CREATE PROCEDURE ReturnResultSet(IN tableName
                                 VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnResultSet' ;
```

```
public static void ReturnResultSet(string tableName)
{
   DB2Command myCommand = DB2Context.GetCommand();

   // Set the SQL statement to be executed and execute it.
   myCommand.CommandText = "SELECT * FROM " + tableName;
   DB2DataReader reader = myCommand.ExecuteReader();

   // The DB2DataReader contains the result of the query.
   // This result set can be returned with the procedure,
   // by simply NOT closing the DB2DataReader.
   // Specifically, do NOT execute reader.Close();
}
```

示例 5: 访问 dbinfo 结构的 C# 参数样式 SQL 过程

本示例显示下列内容:

- 访问 dbinfo 结构的过程的 CREATE PROCEDURE 语句

- 访问 dbinfo 结构的参数样式 SQL 过程的 C# 代码

必须在 CREATE PROCEDURE 语句中指定 DBINFO 子句，才能访问 dbinfo 结构。CREATE PROCEDURE 语句中的 dbinfo 结构不需要参数，但是，必须在外部例程代码中为此结构创建参数。此过程只返回 dbinfo 结构中 dbname 字段的当前数据库名值。

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE SQL
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
DBINFO
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!ReturnDbName'
;
```

```
 public static void ReturnDbName(out string dbName,
                                  out Int16  dbNameNullInd,
                                  ref string sqlState,
                                      string funcName,
                                      string specName,
                                  ref string sqlMessageText,
                                      sqludf_dbinfo dbinfo)
 {
    // Retrieve the current database name from the
    // dbinfo structure and return it.
    // ** Note! ** dbinfo field names are case sensitive
    dbName = dbinfo.dbname;
    dbNameNullInd = 0;  // Return a non-null value;

    // If you want to return a user-defined error in
    // the SQLCA you can specify a 5 digit user-defined
    // sqlStateate and an error message string text.
    // For example:
    //
    //   sqlStateate = "ABCDE";
    //   sqlMessageText = "A user-defined error has occurred"
    //
    //  DB2 returns the above values to the client in the
    //  SQLCA structure.  The values are used to generate a
    //  standard DB2 sqlStateate error.
 }
```

示例 6: 使用 PROGRAM TYPE MAIN 样式的 C# 过程

本示例显示下列内容:

- 使用主程序样式的过程的 CREATE PROCEDURE 语句
- 使用 MAIN 程序样式的 C# 参数样式 GENERAL WITH NULLS 代码

必须在带有值 MAIN 的 CREATE PROCEDURE 语句中指定 PROGRAM TYPE 子句，才能使用主程序样式来实现例程。在 CREATE PROCEDURE 语句中指定参数，但是在代码实现中，会通过 argc 整数参数和 argv 数组参数，将参数传递至例程。

```
CREATE PROCEDURE MainStyle( IN empID CHAR(6),
                            INOUT bonus Decimal(9,2),
                            OUT empName VARCHAR(60))
SPECIFIC MainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
```

```
MODIFIES SQL DATA
FENCED
THREADSAFE
EXECUTION CONTROL SAFE
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!main' ;

 public static void main(Int32 argc, Object[]
 argv)
 {
   String empID = (String)argv[0];   // argv[0] has nullInd:argv[3]
   Decimal bonus = (Decimal)argv[1]; // argv[1] has nullInd:argv[4]
                                     // argv[2] has nullInd:argv[5]
   Decimal salary = 0;
   Int16[] NullInds = (Int16[])argv[3];

   if ((NullInds[0]) == (Int16)(-1)) // Check if empID is null
   {
      NullInds[1] = (Int16)(-1);  // Return a NULL bonus value
      argv[1] = (String)"";       // Set output parameter empName
      NullInds[2] = (Int16)(-1);  // Return a NULL empName value
      Return;
   }
   else
   {
      DB2Command myCommand = DB2Context.GetCommand();
      myCommand.CommandText =
                     "SELECT FIRSTNME, MIDINIT, LASTNAME, salary "
                   + "FROM EMPLOYEE "
                   + "WHERE EMPNO = '" + empID + "'";

      DB2DataReader reader = myCommand.ExecuteReader();

      if (reader.Read())  // If employee record is found
      {
        // Get the employee's full name and salary
        argv[2] = (String) (reader.GetString(0) + " " +
                            reader.GetString(1) + ".
                            " +
                            reader.GetString(2));
        NullInds[2] = (Int16)0;
        salary = reader.GetDecimal(3);

        if (bonus == 0)
        {
           if (salary > 75000)
           {
              argv[1] = (Decimal)(salary * (Decimal)0.025);
              NullInds[1] = (Int16)(0);  // Return a non-NULL value
           }
           else
           {
              argv[1] = (Decimal)(salary * (Decimal)0.05);
              NullInds[1] = (Int16)(0);  // Return a non-NULL value
           }
        }
      }
      else  // Employee not found
      {
         argv[2] = (String)("");         // Set output parameter
         NullInds[2] = (Int16)(-1);      // Return a NULL value
      }

      reader.Close();
    }
 }
```

Visual Basic .NET CLR 函数示例

在了解用户定义的函数 (UDF) 的基本原理以及 CLR 例程的基础知识之后，您可以开始在应用程序和数据库环境中利用 CLR UDF。本主题包含一些帮您入门的 CLR UDF 示例。

开始之前

在使用 CLR UDF 示例之前，您不妨阅读下列概念主题:

- 第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 48 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 11 页的『外部标量函数』
- 构建公共语言运行时 (CLR) .NET 例程

关于此任务

要获取使用 Visual Basic 编写的 CLR 过程示例，请参阅:

- 第 69 页的『Visual Basic .NET CLR 过程示例』

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

过程

在创建您自己的 Visual Basic CLR UDF 时，请使用下列示例作为参考:

- 65
- 65
- 68

示例

Visual Basic 外部代码文件

下列示例显示各种 Visual Basic UDF 实现。为每个 UDF 提供了 CREATE FUNCTION 语句及对应的 Visual Basic 源代码（可根据其构建相关联的组合件）。包含下列示例中所使用函数声明的 Visual Basic 源文件称为 gwenVbUDF.cs 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   ...
   ' Class definitions that contain UDF declarations
   ' and any supporting class definitions
   ...

End Namespace
```

必须将函数声明包含在 Visual Basic 文件的类中。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。如果函数包含 SQL，那么将需要 IBM.Data.DB2. 包含。

示例 1: Visual Basic 参数样式 SQL 表函数

本示例显示下列内容:

- 参数样式 SQL 表函数的 CREATE FUNCTION 语句
- 参数样式 SQL 表函数的 Visual Basic 代码

此表函数会返回一个包含根据数据数组建立的职员数据行的表。本示例有两个相关联的类。类 person 表示职员，而类 empOps 包含使用类 person 的例程表 UDF。会根据输入参数的值来更新职员薪水信息。第一次调用表函数时，会在此表函数本身中创建本示例中的数据数组。此外，还可通过从文件系统上的文本文件读取数据来创建了此类数组。数组数据值将写入暂存区，以便可以在表函数的后续调用中访问此数据。

在每次调用表函数时，会从数组中读取一个记录，并在函数所返回的表中生成一行。通过将表函数的输出参数设为期望的行值，在表中生成行。在最终调用表函数之后，会返回所生成行的表。

```
CREATE FUNCTION TableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenVbUDF.dll:bizLogic.empOps!TableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
SCRATCHPAD 10
FINAL CALL
DISALLOW PARALLEL
NO DBINFO
EXECUTION CONTROL SAFE
```

```
Class Person
' The class Person is a supporting class for
' the table function UDF, tableUDF, below.

  Private name As String
  Private position As String
  Private salary As Int32

  Public Sub New(ByVal newName As String, _
                 ByVal newPosition As String, _
                 ByVal newSalary As Int32)

    name = newName
    position = newPosition
    salary = newSalary
  End Sub

  Public Property GetName() As String
    Get
      Return name
    End Get

    Set (ByVal value As String)
      name = value
    End Set
  End Property

  Public Property GetPosition() As String
    Get
      Return position
    End Get
```

```
    Set (ByVal value As String)
      position = value
    End Set
  End Property

  Public Property GetSalary() As Int32
    Get
      Return salary
    End Get

    Set (ByVal value As Int32)
      salary = value
    End Set
  End Property

End Class

 Class empOps

   Public Shared Sub TableUDF(byVal factor as Double, _
                              byRef name As String, _
                              byRef position As String, _
                              byRef salary As Double, _
                              byVal factorNullInd As Int16, _
                              byRef nameNullInd As Int16, _
                              byRef positionNullInd As Int16, _
                              byRef salaryNullInd As Int16, _
                              byRef sqlState As String, _
                              byVal funcName As String, _
                              byVal specName As String, _
                              byRef sqlMessageText As String, _
                              byVal scratchPad As Byte(), _
                              byVal callType As Int32)

     Dim intRow As Int16

     intRow = 0

     ' Create an array of Person type information
     Dim staff(2) As Person
     staff(0) = New Person("Gwen", "Developer", 10000)
     staff(1) = New Person("Andrew", "Developer", 20000)
     staff(2) = New Person("Liu", "Team Leader", 30000)

     ' Initialize output parameter values and NULL indicators
     salary = 0
     name = position = ""
     nameNullInd = positionNullInd = salaryNullInd = -1

     Select callType
        Case -2   ' Case SQLUDF_TF_FIRST:
        Case -1   ' Case SQLUDF_TF_OPEN:
          intRow = 1
          scratchPad(0) = intRow  ' Write to scratchpad
        Case 0    ' Case SQLUDF_TF_FETCH:
          intRow = scratchPad(0)
          If intRow > staff.Length
             sqlState = "02000"  ' Return an error SQLSTATE
          Else
             ' Generate a row in the output table
             ' based on the staff array data.
             name = staff(intRow).GetName()
             position = staff(intRow).GetPosition()
             salary = (staff(intRow).GetSalary()) * factor
             nameNullInd = 0
             positionNullInd = 0
             salaryNullInd = 0
          End If
```

```
            intRow = intRow + 1
            scratchPad(0) = intRow  ' Write scratchpad

          Case 1    ' Case SQLUDF_TF_CLOSE:

          Case 2    ' Case SQLUDF_TF_FINAL:
        End Select

      End Sub

    End Class
```

示例 2: Visual Basic 参数样式 SQL 标量函数

本示例显示下列内容:

- 参数样式 SQL 标量函数的 CREATE FUNCTION 语句
- 参数样式 SQL 标量函数的 Visual Basic 代码

此标量函数会返回每个输入值所操作的单一计数值。对于输入值集第 n 个位置中的输入值，输出标量值是值 n。对于标量函数的每个调用（每个调用都与行或值输入集中的每一行或值相关联），计数值会增加 1，并返回计数的当前值。然后，会将计数保存在暂存区内存缓冲区中，以在标量函数的每个调用之间保留计数值。

例如，如果我们具有如下定义的表，那么可以很容易地调用此标量函数:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

可以使用如下简单查询来调用标量函数:

```
SELECT my_count(i1) as count, i1 FROM T;
```

此类查询的输出会是:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

此标量 UDF 很简单。不只是返回行数，您还可以使用标量函数来格式化现有列中的数据。例如，您可能会将字符串附加至地址列中的每个值、根据一系列输入字符串构建复杂字符串，或者对您必须在其中存放中间结果的数据集执行复杂数学求值。

```
CREATE FUNCTION mycount(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
NO SQL
SCRATCHPAD 10
FINAL CALL
FENCED
EXECUTION CONTROL SAFE
NOT DETERMINISTIC
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp';
```

```
Class empOps
  Public Shared Sub CountUp(byVal input As Int32, _
                            byRef outCounter As Int32, _
                            byVal nullIndInput As Int16, _
                            byRef nullIndOutCounter As Int16, _
                            byRef sqlState As String, _
```

```
                                byVal qualName As String, _
                                byVal specName As String, _
                                byRef sqlMessageText As String, _
                                byVal scratchPad As Byte(), _
                                byVal callType As Int32)

        Dim counter As Int32
        counter = 1

        Select callType
           case -1           ' case SQLUDF_TF_OPEN_CALL
              scratchPad(0) = counter
              outCounter = counter
              nullIndOutCounter = 0
           case 0             'case SQLUDF_TF_FETCH_CALL:
              counter = scratchPad(0)
              counter = counter + 1
              outCounter = counter
              nullIndOutCounter = 0
              scratchPad(0) = counter
           case 1            'case SQLUDF_CLOSE_CALL:
              counter = scratchPad(0)
              outCounter = counter
              nullIndOutCounter = 0
           case Else          ' Should never enter here
              ' These cases won't occur for the following reasons:
              ' Case -2  (SQLUDF_TF_FIRST)     —>No FINAL CALL in CREATE stmt
              ' Case 2   (SQLUDF_TF_FINAL)     —>No FINAL CALL in CREATE stmt
              ' Case 255 (SQLUDF_TF_FINAL_CRA) —>No SQL used in the function
              '
              ' * Note!*
              ' ---------
              ' The Else is required so that at compile time
              ' out parameter outCounter is always set *
              outCounter = 0
              nullIndOutCounter = -1
        End Select
     End Sub

  End Class
```

Visual Basic .NET CLR 过程示例

在了解过程（又称为存储过程）的基本原理，以及 .NET 公共语言运行时例程的基础知识之后，您可以开始在应用程序中使用 CLR 过程。本主题包含一些使用 Visual Basic 实现的 CLR 过程示例；这些示例说明了受支持的参数样式、传递参数（其中包括 dbinfo 结构）以及如何返回结果集和其他信息。

开始之前

在使用 CLR 过程示例之前，您不妨阅读下列概念主题:

- 第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 48 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 5 页的『使用例程的益处』
- 构建公共语言运行时 (CLR) .NET 例程

关于此任务

要获取使用 Visual Basic 编写的 CLR UDF 示例，请参阅:

- 第 65 页的『Visual Basic .NET CLR 函数示例』

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

过程

在创建您自己的 Visual Basic CLR 过程时，请使用下列示例作为参考:

- 70
- 70
- 71
- 72
- 74
- 74
- 75

示例

Visual Basic 外部代码文件

下列示例显示各种 Visual Basic 过程实现。每个示例都包含两个部分: CREATE PROCEDURE 语句以及过程的外部 Visual Basic 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 Visual Basic 源文件称为 gwenVbProc.vb 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

Namespace bizLogic

   Class empOps
              ...
     ' Visual Basic procedures
              ...
   End Class
End Namespace
```

在此文件顶部指示文件包含。如果此文件中的任何过程包含 SQL，那么将需要 IBM.Data.DB2 包含。此文件中有一个名称空间声明，以及一个包含过程的类 empOps。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。

必须记下此文件的名称、名称空间以及包含给定过程实现的类的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 可以找到 CLR 过程的组合件和类。

示例 1: Visual Basic 参数样式 GENERAL 过程

本示例显示下列内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL 过程的 Visual Basic 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据职员薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE SetEmpBonusGEN(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC setEmpBonusGEN
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGEN'
```

```
  Public Shared Sub SetEmpBonusGEN(ByVal empId As String, _
                                   ByRef bonus As Decimal, _
                                   ByRef empName As String)

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     myCommand = DB2Context.GetCommand()
     myCommand.CommandText = _
               "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
             + "FROM EMPLOYEE " _
             + "WHERE EMPNO = '" + empId + "'"
     myReader = myCommand.ExecuteReader()

     If myReader.Read()  ' If employee record is found
        ' Get the employee's full name and salary
        empName = myReader.GetString(0) + " " _
                + myReader.GetString(1) + ". " _
                + myReader.GetString(2)

        salary = myReader.GetDecimal(3)

        If bonus = 0
           If salary > 75000
              bonus = salary * 0.025
           Else
              bonus = salary * 0.05
           End If
        End If
     Else  ' Employee not found
        empName = ""       ' Set output parameter
     End If

     myReader.Close()

  End Sub
```

示例 2: Visual Basic 参数样式 GENERAL WITH NULLS 过程

本示例显示下列内容:

- 参数样式 GENERAL WITH NULLS 过程的 CREATE PROCEDURE 语句
- 参数样式 GENERAL WITH NULLS 过程的 Visual Basic 代码

此过程接受职员 ID 和当前奖金金额作为输入。如果输入参数不为 NULL，那么将检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到职员数据，那么将返回一个 NULL 字符串和一个整数。

```
CREATE PROCEDURE SetEmpBonusGENNULL(IN empId CHAR(6),
                                    INOUT bonus Decimal(9,2),
                                    OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusGENNULL
```

```
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusGENNULL'

  Public Shared Sub SetEmpBonusGENNULL(ByVal empId As String, _
                                       ByRef bonus As Decimal, _
                                       ByRef empName As String, _
                                       byVal nullInds As Int16())

     Dim salary As Decimal
     Dim myCommand As DB2Command
     Dim myReader As DB2DataReader

     salary = 0

     If nullInds(0) = -1   ' Check if the input is null
        nullInds(1) = -1   ' Return a NULL bonus value
        empName = ""       ' Set output parameter
        nullInds(2) = -1   ' Return a NULL empName value
        Return
     Else
        myCommand = DB2Context.GetCommand()
        myCommand.CommandText = _
                  "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
                + "FROM EMPLOYEE " _
                + "WHERE EMPNO = '" + empId + "'"

        myReader = myCommand.ExecuteReader()

        If myReader.Read()  ' If employee record is found
           ' Get the employee's full name and salary
           empName = myReader.GetString(0) + " " _
                   + myReader.GetString(1) + ". " _
                   + myReader.GetString(2)

           salary = myReader.GetDecimal(3)

           If bonus = 0
              If salary > 75000
                 bonus = Salary * 0.025
                 nullInds(1) = 0 'Return a non-NULL value
              Else
                 bonus = salary * 0.05
                 nullInds(1) = 0 ' Return a non-NULL value
              End If
           Else  'Employee not found
              empName = ""       ' Set output parameter
              nullInds(2) = -1   ' Return a NULL value
           End If
        End If

        myReader.Close()

     End If

  End Sub
```

示例 3: Visual Basic 参数样式 SQL 过程

本示例显示下列内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 参数样式 SQL 过程的 Visual Basic 代码

此过程接受职员 ID 和当前奖金金额作为输入。它会检索职员的姓名和薪水。如果当前奖金金额为零，那么将根据薪水来计算新的奖金，然后将此奖金与职员姓名一起返回。如果找不到此职员，那么将返回一个 NULL 字符串。

```
CREATE PROCEDURE SetEmpBonusSQL(IN empId CHAR(6),
                                INOUT bonus Decimal(9,2),
                                OUT empName VARCHAR(60))
SPECIFIC SetEmpBonusSQL
LANGUAGE CLR
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!SetEmpBonusSQL'
```

```
  Public Shared Sub SetEmpBonusSQL(byVal empId As String, _
                                   byRef bonus As Decimal, _
                                   byRef empName As String, _
                                   byVal empIdNullInd As Int16, _
                                   byRef bonusNullInd As Int16, _
                                   byRef empNameNullInd As Int16, _
                                   byRef sqlState As String, _
                                   byVal funcName As String, _
                                   byVal specName As String, _
                                   byRef sqlMessageText As String)

    ' Declare local host variables
    Dim salary As Decimal
    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader

    salary = 0

    If empIdNullInd = -1   ' Check if the input is null
       bonusNullInd = -1   ' Return a NULL Bonus value
       empName = ""
       empNameNullInd = -1 ' Return a NULL empName value
    Else
      myCommand = DB2Context.GetCommand()
      myCommand.CommandText = _
                 "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
               + "FROM EMPLOYEE " _
               + " WHERE EMPNO = '" + empId + "'"

       myReader = myCommand.ExecuteReader()

       If myReader.Read()  ' If employee record is found
          ' Get the employee's full name and salary
          empName = myReader.GetString(0) + " "
                  + myReader.GetString(1) _
                  + ". " +  myReader.GetString(2)
          empNameNullInd = 0
          salary = myReader.GetDecimal(3)

          If bonus = 0
             If salary > 75000
                bonus = salary * 0.025
                bonusNullInd = 0  ' Return a non-NULL value
             Else
                bonus = salary * 0.05
                bonusNullInd = 0  ' Return a non-NULL value
             End If
          End If
       Else  ' Employee not found
          empName = ""            ' Set output parameter
          empNameNullInd = -1     ' Return a NULL value
       End If
```

```
            myReader.Close()
        End If

    End Sub
```

示例 4: 返回结果集的 Visual Basic 参数样式 GENERAL 过程

本示例显示下列内容:

- 返回结果集的外部 Visual Basic 过程的 CREATE PROCEDURE 语句
- 返回结果集的参数样式 GENERAL 过程的 Visual Basic 代码

此过程接受表名作为参数。它返回一个包含输入参数所指定的全部表行的结果集。通过为过程返回时打开的给定查询结果集保留一个 DB2DataReader 来实现此目的。具体地说，如果未执行 reader.Close()，那么将返回结果集。

```
CREATE PROCEDURE ReturnResultSet(IN tableName VARCHAR(20))
SPECIFIC ReturnResultSet
DYNAMIC RESULT SETS 1
LANGUAGE CLR
PARAMETER STYLE GENERAL
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnResultSet'
```

```
Public Shared Sub ReturnResultSet(byVal tableName As String)

    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader

    myCommand = DB2Context.GetCommand()

    ' Set the SQL statement to be executed and execute it.
    myCommand.CommandText = "SELECT * FROM " + tableName
    myReader = myCommand.ExecuteReader()

    ' The DB2DataReader contains the result of the query.
    ' This result set can be returned with the procedure,
    ' by simply NOT closing the DB2DataReader.
    ' Specifically, do NOT execute reader.Close()

End Sub
```

示例 5: 访问 dbinfo 结构的 Visual Basic 参数样式 SQL 过程

本示例显示下列内容:

- 访问 dbinfo 结构的过程的 CREATE PROCEDURE 语句
- 访问 dbinfo 结构的参数样式 SQL 过程的 Visual Basic 代码

必须在 CREATE PROCEDURE 语句中指定 DBINFO 子句，才能访问 dbinfo 结构。CREATE PROCEDURE 语句中的 dbinfo 结构不需要参数，但是，必须在外部例程代码中为此结构创建参数。此过程只返回 dbinfo 结构中 dbname 字段的当前数据库名值。

```
CREATE PROCEDURE ReturnDbName(OUT dbName VARCHAR(20))
SPECIFIC ReturnDbName
LANGUAGE CLR
PARAMETER STYLE SQL
DBINFO
FENCED
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!ReturnDbName'
```

```
  Public Shared Sub ReturnDbName(byRef dbName As String, _
                                 byRef dbNameNullInd As Int16, _
                                 byRef sqlState As String, _
                                 byVal funcName As String, _
                                 byVal specName As String, _
                                 byRef sqlMessageText As String, _
                                 byVal dbinfo As sqludf_dbinfo)

    ' Retrieve the current database name from the
    ' dbinfo structure and return it.
    dbName = dbinfo.dbname
    dbNameNullInd = 0  ' Return a non-null value

    ' If you want to return a user-defined error in
    ' the SQLCA you can specify a 5 digit user-defined
    ' SQLSTATE and an error message string text.
    ' For example:
    '
    ' sqlState = "ABCDE"
    ' msg_token = "A user-defined error has occurred"
    '
    ' These will be returned by DB2 in the SQLCA.  It
    ' will appear in the format of a regular DB2 sqlState
    ' error.
  End Sub
```

示例 6: 使用 PROGRAM TYPE MAIN 样式的 Visual Basic 过程

本示例显示下列内容:

- 使用主程序样式的过程的 CREATE PROCEDURE 语句
- 使用 MAIN 程序样式的 Visual Basic 参数样式 GENERAL WITH NULLS 代码

必须在带有值 MAIN 的 CREATE PROCEDURE 语句中指定 PROGRAM TYPE 子句，才能使用主程序样式来实现例程。在 CREATE PROCEDURE 语句中指定参数，但是在代码实现中，会通过 argc 整数参数和 argv 数组参数，将参数传递至例程。

```
CREATE PROCEDURE MainStyle(IN empId CHAR(6),
                           INOUT bonus Decimal(9,2),
                           OUT empName VARCHAR(60))
SPECIFIC mainStyle
DYNAMIC RESULT SETS 0
LANGUAGE CLR
PARAMETER STYLE GENERAL WITH NULLS
FENCED
PROGRAM TYPE MAIN
EXTERNAL NAME 'gwenVbProc.dll:bizLogic.empOps!Main'
```

```
  Public Shared Sub Main( byVal argc As Int32, _
                          byVal argv As Object())

    Dim myCommand As DB2Command
    Dim myReader As DB2DataReader
    Dim empId As String
    Dim bonus As Decimal
    Dim salary As Decimal
    Dim nullInds As Int16()

    empId = argv(0)  ' argv[0] (IN)    nullInd = argv[3]
    bonus = argv(1)  ' argv[1] (INOUT) nullInd = argv[4]
                     ' argv[2] (OUT)   nullInd = argv[5]
    salary = 0
    nullInds = argv(3)
```

```
If nullInds(0) = -1     ' Check if the empId input is null
   nullInds(1) = -1     ' Return a NULL Bonus value
   argv(1) = ""         ' Set output parameter empName
   nullInds(2) = -1     ' Return a NULL empName value
   Return
Else
   ' If the employee exists and the current bonus is 0,
   ' calculate a new employee bonus based on the employee's
   ' salary.  Return the employee name and the new bonus
   myCommand = DB2Context.GetCommand()
   myCommand.CommandText = _
             "SELECT FIRSTNME, MIDINIT, LASTNAME, SALARY " _
          + " FROM EMPLOYEE " _
          + " WHERE EMPNO = '" + empId + "'"

   myReader = myCommand.ExecuteReader()


   If myReader.Read()  ' If employee record is found
      ' Get the employee's full name and salary
      argv(2) = myReader.GetString(0) + " "  _
              + myReader.GetString(1) + ". " _
              + myReader.GetString(2)
      nullInds(2) = 0
      salary = myReader.GetDecimal(3)

      If bonus = 0
         If salary > 75000
            argv(1) = salary * 0.025
            nullInds(1) = 0  ' Return a non-NULL value
         Else
            argv(1) = Salary * 0.05
            nullInds(1) = 0  ' Return a non-NULL value
         End If
      End If
   Else  ' Employee not found
      argv(2) = ""      ' Set output parameter
      nullInds(2) = -1  ' Return a NULL value
   End If

   myReader.Close()
End If

End Sub
```

示例: C# .NET CLR 过程中的 XML 和 XQuery 支持

在了解过程的基本原理、.NET 公共语言运行时例程的基础知识、XQuery 和 XML 之后，您可以开始创建并使用具有 XML 功能的 CLR 过程。

下列示例演示一个带有类型为 XML 的参数的 C# .NET CLR 过程，以及演示如何更新和查询 XML 数据。

先决条件

在使用 CLR 过程示例之前，您不妨阅读下列概念主题:

- .NET 公共语言运行时 (CLR) 例程
- 从 DB2 命令窗口创建 .NET CLR 例程
- 使用例程的益处

下列示例利用名为如下定义的表 xmlDataTable:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Bob</name>
                              <town>Toronto</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (10, XMLPARSE(DOCUMENT '<doc>
                               <type>animal</type>
                               <name>bird</name>
                               </doc>' PRESERVE WHITESPACE))@
```

过程 在创建您自己的 C# CLR 过程时，请使用下列示例作为参考:

- 『C# 外部代码文件』
- 第 78 页的『示例 1: 具有 XML 功能的 C# 参数样式 GENERAL 过程』

C# 外部代码文件

本示例包含两个部分: CREATE PROCEDURE 语句以及过程的外部 C# 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 C# 源文件称为 gwenProc.cs 且具有下列格式:

```
using System;
using System.IO;
using System.Data;
using IBM.Data.DB2;
using IBM.Data.DB2Types;

namespace bizLogic
{
   class empOps
   {          ...
     // C# procedures
              ...
   }
}
```

在此文件顶部指示文件包含。如果此文件中的任何过程包含 SQL，那么将需要 IBM.Data.DB2 包含。如果此文件中的任何过程包含类型为 XML 的参数或变量，那么将需要 IBM.Data.DB2Types 包含。此文件中有一个名称空间声明，以及一个包含过程的类 empOps。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。

必须记下此文件的名称、名称空间以及包含给定过程实现的类的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 数据库系统可以找到 CLR 过程的组合件和类。

示例 1: 具有 XML 功能的 C# 参数样式 GENERAL 过程

本示例显示下列内容:

- 参数样式 GENERAL 过程的 CREATE PROCEDURE 语句
- 带有 XML 参数的参数样式 GENERAL 过程的 C# 代码

此过程接受两个参数（整数 inNum 和 inXML）。会将这些值插入表 xmlDataTable。然后使用 XQuery 来检索 XML 值。使用 SQL 来检索另一个 XML 值。会将检索到的 XML 值分别指定给两个输出参数（outXML1 和 outXML2）。不返回结果集。

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                               IN inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K),
                               OUT inXML  XML as CLOB (1K)
                             )
LANGUAGE CLR
PARAMETER STYLE GENERAL
DYNAMIC RESULT SETS 0
FENCED
THREADSAFE
DETERMINISTIC
NO DBINFO
MODIFIES SQL DATA
PROGRAM TYPE SUB
EXTERNAL NAME 'gwenProc.dll:bizLogic.empOps!xmlProc1' ;

//*************************************************************************
//  Stored Procedure: xmlProc1
//
//  Purpose:   insert XML data into XML column
//
//  Parameters:
//
//   IN:    inNum -- the sequence of XML data to be insert in xmldata table
//          inXML -- XML data to be inserted
```

```
//   OUT:   outXML1 -- XML data returned - value retrieved using XQuery
//          outXML2 -- XML data returned - value retrieved using SQL
//*************************************************************************

    public static void xmlProc1 (     int      inNum, DB2Xml   inXML,
                                  out DB2Xml   outXML1, out DB2Xml   outXML2 )
    {
       // Create new command object from connection context
       DB2Parameter parm;
       DB2Command cmd;
       DB2DataReader reader = null;
       outXML1 = DB2Xml.Null;
       outXML2 = DB2Xml.Null;

       // Insert input XML parameter value into a table
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "INSERT INTO "
                       + "xmlDataTable( num , xdata ) "
                       + "VALUES( ?, ? )";

       parm = cmd.Parameters.Add("@num", DB2Type.Integer );
       parm.Direction = ParameterDirection.Input;
       cmd.Parameters["@num"].Value = inNum;
       parm = cmd.Parameters.Add("@data", DB2Type.Xml);
       parm.Direction = ParameterDirection.Input;
       cmd.Parameters["@data"].Value = inXML ;
       cmd.ExecuteNonQuery();
       cmd.Close();

       // Retrieve XML value using XQuery
                     and assign value to an XML output parameter
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "XQUERY for $x " +
                       "in db2-fn:xmlcolumn(\"xmlDataTable.xdata\")/doc "+
                       "where $x/make = \'Mazda\' " +
                       "return <carInfo>{$x/make}{$x/model}</carInfo>";
       reader = cmd.ExecuteReader();
       reader.CacheData= true;

       if (reader.Read())
       {  outXML1 = reader.GetDB2Xml(0);  }
       else
       {  outXML1 = DB2Xml.Null;  }

       reader.Close();
       cmd.Close();

       // Retrieve XML value using SQL
                     and assign value to an XML output parameter value
       cmd = DB2Context.GetCommand();
       cmd.CommandText = "SELECT xdata "
                        + "FROM xmlDataTable "
                        + "WHERE num = ?";

       parm = cmd.Parameters.Add("@num", DB2Type.Integer );
       parm.Direction = ParameterDirection.Input;
       cmd.Parameters["@num"].Value = inNum;
       reader = cmd.ExecuteReader();
       reader.CacheData= true;

       if (reader.Read())
       {  outXML2 = reader.GetDB2Xml(0);  }
       else
       {  outXML = DB2Xml.Null;  }

       reader.Close() ;
```

```
        cmd.Close();

        return;
    }
```

示例: C 过程中的 XML 和 XQuery 支持

在了解过程的基本原理、C 例程的基础知识、XQuery 和 XML 之后，您可以开始创建并使用具有 XML 功能的 C 过程。

下列示例演示一个带有类型为 XML 的参数的 C 过程，以及演示如何更新和查询 XML 数据。

先决条件

在使用 C 过程示例之前，您不妨阅读下列概念主题:

- 使用例程的益处

下列示例利用名为如下定义的表 xmlDataTable:

```
CREATE TABLE xmlDataTable
(
   num INTEGER,
   xdata XML
)

INSERT INTO xmlDataTable VALUES
   (1, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Pontiac</make>
                              <model>Sunfire</model>
                              </doc>' PRESERVE WHITESPACE)),
   (2, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Mazda</make>
                              <model>Miata</model>
                              </doc>' PRESERVE WHITESPACE)),
   (3, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mary</name>
                              <town>Vancouver</town>
                              <street>Waterside</street>
                              </doc>' PRESERVE WHITESPACE)),
   (4, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Mark</name>
                              <town>Edmonton</town>
                              <street>Oak</street>
                              </doc>' PRESERVE WHITESPACE)),
   (5, XMLPARSE(DOCUMENT '<doc>
                              <type>animal</type>
                              <name>dog</name>
                              </doc>' PRESERVE WHITESPACE)),
   (6, NULL),
   (7, XMLPARSE(DOCUMENT '<doc>
                              <type>car</type>
                              <make>Ford</make>
                              <model>Taurus</model>
                              </doc>' PRESERVE WHITESPACE)),
   (8, XMLPARSE(DOCUMENT '<doc>
                              <type>person</type>
                              <name>Kim</name>
                              <town>Toronto</town>
                              <street>Elm</street>
                              </doc>' PRESERVE WHITESPACE)),
   (9, XMLPARSE(DOCUMENT '<doc>
```

```
                                    <type>person</type>
                                    <name>Bob</name>
                                    <town>Toronto</town>
                                    <street>Oak</street>
                                    </doc>' PRESERVE WHITESPACE)),
        (10, XMLPARSE(DOCUMENT '<doc>
                                     <type>animal</type>
                                     <name>bird</name>
                                     </doc>' PRESERVE WHITESPACE))
```

过程 在创建您自己的 C 过程时，请使用下列示例作为参考:

- 『C 外部代码文件』
- 『示例 1: 具有 XML 功能的 C 参数样式 SQL 过程』

C 外部代码文件

本示例包含两个部分: CREATE PROCEDURE 语句以及过程的外部 C 代码实现（可根据其构建相关联的组合件）。

包含下列示例过程实现的 C 源文件称为 gwenProc.SQC 且具有下列格式:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlda.h>
#include <sqlca.h>
#include <sqludf.h>
#include <sql.h>
#include <memory.h>

// C procedures
               ...
```

在此文件顶部指示文件包含。嵌入式 SQL 例程中的 XML 支持不需要额外的包含文件。

必须记下此文件的名称以及对应于过程实现的函数的名称。这些名称很重要，因为每个过程的 CREATE PROCEDURE 语句的 EXTERNAL 子句必须指定此信息，以便 DB2 数据库管理器可以找到对应于 C 过程的库和入口点。

示例 1: 具有 XML 功能的 C 参数样式 SQL 过程

本示例显示下列内容:

- 参数样式 SQL 过程的 CREATE PROCEDURE 语句
- 带有 XML 参数的参数样式 SQL 过程的 C 代码

此过程接收两个输入参数。第一个输入参数为 inNum 且属于 INTEGER 类型。第二个输入参数为 inXML 且属于 XML 类型。使用输入参数的值将行插入表 xmlDataTable。然后使用 SQL 语句来检索 XML 值。使用 XQuery 表达式来检索另一个 XML 值。会将检索到的 XML 值分别指定给两个输出参数（out1XML 和 out2XML）。不返回结果集。

```
CREATE PROCEDURE  xmlProc1   ( IN inNUM  INTEGER,
                             IN inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K),
                             OUT inXML  XML as CLOB (1K)
                           )
LANGUAGE C
PARAMETER STYLE SQL
DYNAMIC RESULT SETS 0
```

```
  FENCED
  THREADSAFE
  DETERMINISTIC
  NO DBINFO
  MODIFIES SQL DATA
  PROGRAM TYPE SUB
  EXTERNAL NAME 'gwenProc!xmlProc1' ;

  //*************************************************************************
  //  Stored Procedure: xmlProc1
  //
  //  Purpose:   insert XML data into XML column
  //
  //  Parameters:
  //
  //   IN:    inNum -- the sequence of XML data to be insert in xmldata table
  //          inXML -- XML data to be inserted
  //   OUT:   out1XML -- XML data returned - value retrieved using XQuery
  //          out2XML -- XML data returned - value retrieved using SQL
  //*************************************************************************

#ifdef __cplusplus
extern "C"
#endif
SQL_API_RC SQL_API_FN testSecA1(sqlint32* inNum,
                                SQLUDF_CLOB* inXML,
                                SQLUDF_CLOB* out1XML,
                                SQLUDF_CLOB* out2XML,
                                SQLUDF_NULLIND *inNum_ind,
                                SQLUDF_NULLIND *inXML_ind,
                                SQLUDF_NULLIND *out1XML_ind,
                                SQLUDF_NULLIND *out2XML_ind,
                                SQLUDF_TRAIL_ARGS)
{
  char *str;
  FILE *file;

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
       sqlint32 hvNum1;
       SQL TYPE IS XML AS CLOB(200) hvXML1;
       SQL TYPE IS XML AS CLOB(200) hvXML2;
       SQL TYPE IS XML AS CLOB(200) hvXML3;
  EXEC SQL END DECLARE SECTION;

  /* Check null indicators for input parameters */
  if ((*inNum_ind < 0) ||  (*inXML_ind < 0)) {
     strcpy(sqludf_sqlstate, "38100");
     strcpy(sqludf_msgtext, "Received null input");
     return 0;
  }

  /* Copy input parameters to host variables */
  hvNum1 = *inNum;
  hvXML1.length = inXML->length;
  strncpy(hvXML1.data, inXML->data, inXML->length);

  /* Execute SQL statement */
  EXEC SQL
     INSERT INTO xmlDataTable (num, xdata) VALUES (:hvNum1, :hvXML1);

  /* Execute SQL statement */
  EXEC SQL
     SELECT xdata INTO :hvXML2
        FROM xmlDataTable
           WHERE num = :hvNum1;
```

```
  sprintf(stmt5, "SELECT XMLQUERY('for $x in $xmldata/doc
                                   return <carInfo>{$x/model}</carInfo>'
                                   passing by ref xmlDataTable.xdata
                                       as \"xmldata\" returning sequence)
                    FROM xmlDataTable WHERE num = ?");

  EXEC SQL PREPARE selstmt5 FROM :stmt5 ;
  EXEC SQL DECLARE c5 CURSOR FOR selstmt5;
  EXEC SQL OPEN c5 using :hvNum1;
  EXEC SQL FETCH c5 INTO :hvXML3;

  exit:

  /* Set output return code */
  *outReturnCode = sqlca.sqlcode;
  *outReturnCode_ind = 0;

  return 0;
}
```

C# .NET CLR 函数示例

在了解用户定义的函数 (UDF) 的基本原理以及 CLR 例程的基础知识之后，您可以开始在应用程序和数据库环境中利用 CLR UDF。本主题包含一些帮您入门的 CLR UDF 示例。

开始之前

在使用 CLR UDF 示例之前，您不妨阅读下列概念主题:

- 第 39 页的第 3 章，『.NET 公共语言运行时 (CLR) 例程』
- 第 48 页的『从 DB2 命令窗口创建 .NET CLR 例程』
- 第 11 页的『外部标量函数』
- 构建公共语言运行时 (CLR) .NET 例程

下列示例利用 SAMPLE 数据库中包括的名为 EMPLOYEE 的表。

关于此任务

要获取使用 C# 编写的 CLR 过程示例，请参阅:

- 第 57 页的『C# .NET CLR 过程示例』

过程

在创建您自己的 C# CLR UDF 时，请使用下列示例作为参考:

- 83
- 84
- 86

示例

C# 外部代码文件

下列示例显示各种 C# UDF 实现。为每个 UDF 提供了 CREATE FUNCTION 语句及对应的 C# 源代码（可根据其构建相关联的组合件）。包含下列示例中所使用函数声明的 C# 源文件称为 gwenUDF.cs 且具有下列格式:

```
using System;
using System.IO;
using IBM.Data.DB2;

namespace bizLogic
{
   ...
   // Class definitions that contain UDF declarations
   // and any supporting class definitions
   ...
}
```

必须将函数声明包含在 C# 文件的类中。可选择性地使用名称空间。如果使用名称空间，那么该名称空间必须出现在组合件路径名（在 CREATE PROCEDURE 语句的 EXTERNAL 子句中提供）中。如果函数包含 SQL，那么将需要 `IBM.Data.DB2.` 包含。

示例 1: C# 参数样式 SQL 表函数

本示例显示下列内容:

- 参数样式 SQL 表函数的 CREATE FUNCTION 语句
- 参数样式 SQL 表函数的 C# 代码

此表函数会返回一个包含根据数据数组建立的职员数据行的表。本示例有两个相关联的类。类 `person` 表示职员，而类 `empOps` 包含使用类 `person` 的例程表 UDF。会根据输入参数的值来更新职员薪水信息。第一次调用表函数时，会在此表函数本身中创建本示例中的数据数组。此外，还可通过从文件系统上的文本文件读取数据来创建了此类数组。数组数据值将写入暂存区，以便可以在表函数的后续调用中访问此数据。

在每次调用表函数时，会从数组中读取一个记录，并在函数所返回的表中生成一行。通过将表函数的输出参数设为期望的行值，在表中生成行。在最终调用表函数之后，会返回所生成行的表。

```
CREATE FUNCTION tableUDF(double)
RETURNS TABLE (name varchar(20),
               job varchar(20),
               salary double)
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!tableUDF'
LANGUAGE CLR
PARAMETER STYLE SQL
NOT DETERMINISTIC
FENCED
THREADSAFE
SCRATCHPAD 10
FINAL CALL
EXECUTION CONTROL SAFE
DISALLOW PARALLEL
NO DBINFO
```

```
// The class Person is a supporting class for
// the table function UDF, tableUDF, below.
class Person
{
     private String name;
     private String position;
     private Int32 salary;

     public Person(String newName, String newPosition, Int32
     newSalary)
     {
        this.name = newName;
        this.position = newPosition;
```

```
      this.salary = newSalary;
    }

    public String getName()
    {
      return this.name;
    }

    public String getPosition()
    {
      return this.position;
    }

    public Int32 getSalary()
    {
      return this.salary;
    }
}

class empOps
{

  public static void TableUDF( Double factor, out String name,
                   out String position, out Double salary,
                   Int16 factorNullInd, out Int16 nameNullInd,
                   out Int16 positionNullInd, out Int16 salaryNullInd,
                   ref String sqlState, String funcName,
                   String specName, ref String sqlMessageText,
                   Byte[] scratchPad, Int32 callType)
  {

    Int16 intRow = 0;

    // Create an array of Person type information
    Person[] Staff = new
    Person[3];
    Staff[0] = new Person("Gwen", "Developer", 10000);
    Staff[1] = new Person("Andrew", "Developer", 20000);
    Staff[2] = new Person("Liu", "Team Leader", 30000);

    salary = 0;
    name = position = "";
    nameNullInd = positionNullInd = salaryNullInd = -1;

    switch(callType)
    {
      case (-2):  // Case SQLUDF_TF_FIRST:
        break;

      case (-1):  // Case SQLUDF_TF_OPEN:
        intRow = 1;
        scratchPad[0] = (Byte)intRow;  // Write to scratchpad
        break;
      case (0):   // Case SQLUDF_TF_FETCH:
        intRow = (Int16)scratchPad[0];
        if (intRow > Staff.Length)
        {
          sqlState = "02000";  // Return an error SQLSTATE
        }
        else
        {
          // Generate a row in the output table
          // based on the Staff array data.
          name =
          Staff[intRow-1].getName();
          position = Staff[intRow-1].getPosition();
          salary = (Staff[intRow-1].getSalary()) * factor;
          nameNullInd = 0;
```

```
                positionNullInd = 0;
                salaryNullInd = 0;
              }
              intRow++;
              scratchPad[0] = (Byte)intRow;  // Write scratchpad
              break;

            case (1):   // Case SQLUDF_TF_CLOSE:
              break;

            case (2):   // Case SQLUDF_TF_FINAL:
              break;
          }
        }
      }
```

示例 2: C# 参数样式 SQL 标量函数

本示例显示下列内容:

- 参数样式 SQL 标量函数的 CREATE FUNCTION 语句
- 参数样式 SQL 标量函数的 C# 代码

此标量函数会返回每个输入值所操作的单一计数值。对于输入值集第 n 个位置中的输入值，输出标量值是值 n。对于标量函数的每个调用（每个调用都与行或值输入集中的每一行或值相关联），计数值会增加 1，并返回计数的当前值。然后，会将计数保存在暂存区内存缓冲区中，以在标量函数的每个调用之间保留计数值。

例如，如果我们具有如下定义的表，那么可以很容易地调用此标量函数:

```
CREATE TABLE T (i1 INTEGER);
INSERT INTO T VALUES 12, 45, 16, 99;
```

可以使用如下简单查询来调用标量函数:

```
  SELECT countUp(i1) as count, i1 FROM T;
```

此类查询的输出会是:

```
COUNT           I1
-----------     ----------
1               12
2               45
3               16
4               99
```

此标量 UDF 很简单。不只是返回行数，您还可以使用标量函数来格式化现有列中的数据。例如，您可能会将字符串附加至地址列中的每个值、根据一系列输入字符串构建复杂字符串，或者对您必须在其中存放中间结果的数据集执行复杂数学求值。

```
CREATE FUNCTION countUp(INTEGER)
RETURNS INTEGER
LANGUAGE CLR
PARAMETER STYLE SQL
SCRATCHPAD 10
FINAL CALL
NO SQL
FENCED
THREADSAFE
NOT DETERMINISTIC
EXECUTION CONTROL SAFE
EXTERNAL NAME 'gwenUDF.dll:bizLogic.empOps!CountUp' ;
```

```
class empOps
{
   public static void CountUp(     Int32 input,
                                out Int32 outCounter,
                                    Int16 inputNullInd,
                                out Int16 outCounterNullInd,
                                ref String sqlState,
                                    String funcName,
                                    String specName,
                                ref String sqlMessageText,
                                    Byte[] scratchPad,
                                    Int32 callType)
   {
      Int32 counter = 1;

      switch(callType)
      {
         case -1: // case SQLUDF_FIRST_CALL
           scratchPad[0] = (Byte)counter;
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         case 0:  // case SQLUDF_NORMAL_CALL:
           counter = (Int32)scratchPad[0];
           counter = counter + 1;
           outCounter = counter;
           outCounterNullInd = 0;
           scratchPad[0] =
           (Byte)counter;
           break;
         case 1:  // case SQLUDF_FINAL_CALL:
           counter =
           (Int32)scratchPad[0];
           outCounter = counter;
           outCounterNullInd = 0;
           break;
         default: // Should never enter here
                  // * Required so that at compile time
                  //   out parameter outCounter is always set *
           outCounter = (Int32)(0);
           outCounterNullInd = -1;
           sqlState="ABCDE";
           sqlMessageText = "Should not get here: Default
           case!";
           break;
      }
   }
}
```

第 4 章 IBM Data Server Provider for .NET 概述

IBM 数据服务器 .NET 提供程序扩展了对 ADO.NET 接口的数据服务器支持。此提供程序使您能够安全而高性能地访问 IBM 数据服务器。

IBM 数据服务器 .NET 提供程序 客户机程序包中包括两个提供程序。这些提供程序有时又称为公共 .NET 提供程序。

DB2 .NET Provider (IBM.Data.DB2.dll)

通过 DB2 .NET Provider, .NET 应用程序可访问以下数据库管理系统:

- DB2 Database for Linux, UNIX, and Windows V9.1、V9.5、V9.7、V9.8 和 V10.1
- DB2 Universal Database™ V8 for Windows, UNIX, and Linux
- DB2 z/OS 版 V8、V9 和 V10, 通过 DB2 Connect™
- IBM DB2 for IBM i V5R4、V6R1 和 V7R1, 通过 DB2 Connect（对于 IBM DB2 V9.7 FP4 及更高版本）
- IBM DB2 for IBM i V5R4 和 V6R1, 通过 DB2 Connect（对于 IBM DB2 V9.7 FP3 及更低版本）
- IBM Informix V11.10、V11.50 和 V11.70

余下主题讨论了公共 DB2 .NET Provider。

Informix 数据库服务器 .NET Provider (IBM.Data.Informix.dll)

通过 Informix 数据库服务器 .NET Provider, .NET 应用程序可访问以下数据库管理系统:

- IBM Informix V11.10 和 V11.50

。有关此提供程序的更多信息, 请参阅 IBM Informix Dynamic Server 信息中心。

要开发和运行使用 数据服务器 .NET 提供程序的应用程序, 您需要 .NET Framework。

除 IBM 数据服务器 .NET 提供程序之外, IBM Database Add-Ins for Visual Studio 还允许您使用 Microsoft Visual Studio 快速轻松地为 IBM 数据服务器开发 .NET 应用程序。还可以使用该加载件来创建数据库对象（例如, 索引和表）以及开发服务器端对象（例如, 存储过程和用户定义的函数）。

DB2 的 IBM Data Server Provider for .NET 数据库系统要求

DB2 .NET Provider (IBM.Data.DB2.dll)

通过 DB2 .NET Provider, .NET 应用程序可访问以下数据库管理系统:

- DB2 Database for Linux, UNIX, and Windows V9.1、V9.5、V9.7、V9.8 和 V10.1
- DB2 Universal Database V8 for Windows, UNIX, and Linux
- DB2 z/OS 版 V8、V9 和 V10, 通过 DB2 Connect

- IBM DB2 for IBM i V5R4、V6R1 和 V7R1，通过 DB2 Connect（对于 IBM DB2 V9.7 FP4 及更高版本）
- IBM DB2 for IBM i V5R4 和 V6R1，通过 DB2 Connect（对于 IBM DB2 V9.7 FP3 及更低版本）
- IBM Informix V11.10、V11.50 和 V11.70

余下主题讨论了公共 DB2 .NET Provider。

Informix 数据库服务器 .NET Provider (IBM.Data.Informix.dll)

通过 Informix 数据库服务器 .NET Provider，.NET 应用程序可访问以下数据库管理系统:

- IBM Informix V11.10 和 V11.50

。有关此提供程序的更多信息，请参阅 IBM Informix Dynamic Server 信息中心。

安装 IBM Data Provider for .NET 之前，必须已有下列其中一个 .NET Framework 版本:

- .NET Framwork V2.0
- .NET Framwork V3.0
- .NET Framwork V3.5
- .NET Framwork V4.0

在未安装 .NET Framework 的情况下，IBM Data Server Client 和驱动程序安装程序将无法安装 IBM Data Server Provider for .NET。您将需要以手动方式安装数据提供程序。

对于 DB2 i 版，需要在服务器上安装以下修订: APAR II13348。

对 ADO.NET 应用程序的 32 位和 64 位支持

IBM Data Server Provider for .NET 版本支持 32 位和 64 位 .NET 应用程序。DB2 V9.5 或更高版本客户机和服务器中包含 IBM Data Server Provider for .NET。

下表指定 IBM Data Server Provider for .NET 和 Microsoft .NET Framework 的组合提供的 32 位和 64 位支持。

表 9. IBM Data Server Provider for .NET 和 Microsoft .NET Framework 提供的 32 位和 64 位支持

IBM Data Server Provider for .NET 和 Microsoft .NET Framework 版本	32 位支持	64 位支持
IBM Data Server Provider for .NET 和 Microsoft .NET Framework V2.0、V3.0、V3.5 或 V4.0	Yes	是

CLR 存储过程和用户定义的函数在 IBM Data Server Provider for .NET 的 32 位和 64 位版本中受支持。在此修订包之前，仅在 32 位修订版中受支持。

64 位 ADO.NET 应用程序中的 32 位支持

在 64 位客户机程序包中，IBM Data Server Package for .NET 版本支持 32 位 .NET 提供程序。安装 64 位 IBM Data Server Package 时，会同时安装和配置 32 位和 64 位提供程序。

从 IBM DB2 V9.7 FP2 开始，32 位提供程序与 64 位提供程序打包在一起。

编写应用程序以使用 IBM Data Server Provider for .NET

使用 ADO.NET 公共基类进行通用编码

.NET Framework V2.0、V3.0 和 V3.5 提供了名为 System.Data.Common 的名称空间，该名称空间提供了一组可以由任何 .NET 数据提供程序共享的基类。这会让通用 ADO.NET 数据库应用程序开发方法（针对不同数据库间提供一致的编程接口）顺利进行。

以下 C# 代码演示建立数据库连接的通用方法。

```
DbProviderFactory factory = DbProviderFactories.GetFactory("IBM.Data.DB2");
DbConnection conn = factory.CreateConnection();
DbConnectionStringBuilder sb = factory.CreateConnectionStringBuilder();

if( sb.ContainsKey( "Database" ) )
{
   sb.Remove( "database" );
   sb.Add( "database", "SAMPLE" );
}

conn.ConnectionString = sb.ConnectionString;

conn.Open();
```

DbProviderFactory 对象是任何通用 ADO.NET 应用程序的开始点。此对象创建 .NET 数据提供程序对象（例如连接、数据适配器、命令和数据阅读器）的通用实例，这些实例与特定的数据库产品配合工作。在以上示例中，传递至 GetFactory 方法的 "IBM.Data.DB2" 字符串唯一标识 IBM Data Server Provider for .NET，并且导致 DbProviderFactory 实例初始化，该实例创建特定于 IBM Data Server Provider for .NET 的数据库提供程序对象实例。

DbConnection 对象可连接至 DB2 系列IDS 数据库（就像 DB2Connection 对象，它实际上是从 DbConnection 继承的）。通过使用 DbConnectionStringBuilder 类，您可以确定数据提供程序的连接字符串关键字并生成定制连接字符串。以上示例中的代码检查 IBM Data Server Provider for .NET 中是否存在名为 "database" 的关键字，如果存在，那么会生成连接字符串以连接至 SAMPLE 数据库。

使用 IBM Data Server Provider for .NET 来从应用程序连接至数据库

使用 IBM Data Server Provider for .NET 时，会通过 DB2Connection 类建立数据库连接。

过程

要连接至数据库，请执行以下操作:

1. 创建用于存储连接参数的字符串。典型连接字符串的格式为:

```
Server=<ip address/localhost>:<port number>;
Database=<db name>;
UID=<userID>;
PWD=<password>;
Connect Timeout=<Timeout value>
```

可能的连接字符串的示例:

示例 1:

```
String connectString = "Database=SAMPLE";
// When used, attempts to connect to the SAMPLE database.
```

注: 如果仅在连接字符串中指定数据库名称，那么必须将其他信息（例如，服务器、用户标识和密码）包括在 db2dsdriver.cfg 文件中。

示例 2:

```
String cs = "Server=srv:50000;Database=SAMPLE;UID=db2adm;PWD=ab1d;Connect Timeout=30";
// When used, attempts to connect to the SAMPLE database on the server
// 'srv' through port 50000 using 'db2adm' and 'ab1d' as the user id and
// password respectively. If the connection attempt takes more than thirty seconds,
// the attempt will be terminated and an error will be generated.
```

2. 将 connectString 传递至 DB2Connection 构造函数。
   - 使用 C# 连接到数据库:

     ```
     String connectString = "Database=SAMPLE";
     DB2Connection conn = new DB2Connection(connectString);
     conn.Open();
     return conn;
     ```

   - 在 Visual Basic .NET 中连接到数据库:

     ```
     Dim connectString As String = "Database=SAMPLE"
     Dim conn As DB2Connection = new DB2Connection(connectString)
     conn.Open()
     Return conn
     ```

3. 使用 DB2Connection 对象的 Open 方法正式连接至 connectString 中标识的数据库。

使用 IBM Data Server Provider for .NET 进行连接合用

第一次对 DB2 数据库打开连接时，会创建连接池。关闭连接时，连接会进入该池，准备供需要连接的其他应用程序在同一进程内重复使用。

IBM Data Server Provider for .NET 使用一组规范化连接字符串属性来确定连接池。通过使用规范化属性，应用程序重复使用连接的机会将增加。

缺省情况下，IBM Data Server Provider for .NET 将启用连接合用。

注: 您可以使用 Pooling=false 连接字符串关键字/值对来关闭连接合用。但是，如果关闭连接合用，那么 COM+ 应用程序将不工作。

通过设置下列连接字符串关键字，您可以控制连接池的行为:

- 最小和最大池大小（**MinPoolSize** 和 **MaxPoolSize**）
- 连接返回至池之前可保持空闲状态的时长（**ConnectionLifetimeInPool**）

通过 IBM Data Server Provider for .NET 创建可信连接

从 V9.5 FP1 开始，.NET 应用程序通过使用连接字符串关键字来支持可信上下文。

在连接字符串中，可用的关键字如下所示:

- TrustedContextSystemUserID 或 tcsuid，此关键字指定要与连接配合使用的可信上下文 SYSTEM AUTHID。

- TrustedContextSystemPassword 或 tcspwd，此关键字指定与连接配合使用的可信上下文 SYSTEM AUTHID 的相应密码。

如果在未指定 TrustedContextSystemUserID 关键字值的情况下指定 TrustedContextSystemPassword 关键字，那么将抛出 InvalidArgument 异常。在可信上下文方案中，UserID 关键字也是必需的。

目前以下版本支持通过 IBM Data Server Provider for .NET 创建可信上下文:

- DB2 Database for Linux, UNIX, and Windows V9.5、V9.7 和 V9.8
- DB2 Universal Database V9 and V10 for z/OS

示例

假定已在服务器上使用以下信息建立可信上下文:

```
CREATE TRUSTED CONTEXT ctxName1
BASED UPON CONNECTION USING SYSTEM AUTHID masteruser
ATTRIBUTES ( PROTOCOL 'TCPIP',
             ADDRESS '9.26.146.201',
             ENCRYPTION 'NONE' )
ENABLEWITH USE FOR userapp1 WITH AUTHENTICATION, userapp2 WITH AUTHENTICATION;
```

SYSTEM AUTHID 为 masteruser，其相应密码为 masterpassword。每个特定的用户/应用程序（userapp1 和 userapp2）都有相应的密码（passapp1 和 passapp2）。

要使用此可信上下文，应用程序将发出连接字符串，如下所示:

- 应用程序 1

  ```
  database=db;server=server1:446;
  UserID=userapp1;Password=passapp1;
  TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
  ```

- 应用程序 2

  ```
  database=db;server=server1:446;
  UserID=userapp2;Password=passapp2;
  TrustedContextSystemUserID=masteruser;TrustedContextSystemPassword=masterpassword
  ```

注: 在可信上下文环境中，UserID 关键字与连接的最终用户相对应，这与标准应用程序的情况相同。

按照 .NET 程序打开和关闭某个连接:

```
[C#]
DB2Connection conn = new DB2Connection();

conn.ConnectionString = "database=db;server=server1:446;"
     + "UserID=userapp1;Password=passapp1;"
     + "TrustedContextSystemUserID=masteruser;"
     + "TrustedContextSystemPassword=masterpassword;"

conn.Open();

// Do processing as userapp1, such as querying tables

conn.Close();
conn.ConnectionString = "database=db;server=server1:446;UserID=userapp2;"
     + "Password=passapp2;TrustedContextSystemUserID=masteruser;"
     + "TrustedContextSystemPassword=masterpassword;"

conn.Open();
```

```
// Do processing as userapp2

conn.Close();
```

如果可信上下文处理由于您尚未在服务器上设置任何可信上下文而失败，或者服务器不支持可信上下文，那么将抛出 SQLCODE 为 CLI0197E 的错误。如果 TrustedContextSystemUserID 关键字值无效（例如太长），那么将抛出 SQLCODE 为 CLI0124E 的错误。服务器可能会报告 SQLCODE 为 SQL1046N、SQL30082N 或 SQL0969N 的错误，并且本机错误代码为 -20361。任何这些错误都将导致 Open() 失败。

注: 可信上下文处理将在您下次与服务器进行通信时进行。

ADO.NET 数据库应用程序中的 SQL 数据类型表示

ADO.NET 数据库应用程序可引用 DB2 SQL 数据类型值作为参数值（用作所执行的 SQL 语句的组成部分）和变量，但是，必须使用适当的 IBM 数据服务器 .NET 提供程序数据类型值和 .NET Framework 数据类型值以确保访问或检索值时不会发生数据截断或数据丢失。

必须使用 IBM 数据服务器 .NET 提供程序 对象，才能指定要用作所要执行的 SQL 语句一部分的参数值。`DB2Parameter` 对象用于表示要添加至 `DB2Command` 对象（它表示 SQL 语句）的参数。指定参数的数据类型值时，必须使用 `IBM.Data.DB2Types` 名称空间中提供的 IBM 数据服务器 .NET 提供程序数据类型值。`IBM.Data.DB2Types` 名称空间提供类和结构以表示每个受支持的 DB2 SQL 数据类型。

对于可能临时保存 SQL 数据类型值的局部变量，必须使用适当 IBM 数据服务器 .NET 提供程序数据类型（如 `IBM.Data.DB2Types` 名称空间中所定义）。

下表显示 DB2Type 数据类型、DB2 数据类型、 Informix 数据类型、Microsoft .NET Framework 类型以及 DB2Types 类和结构之间的映射。

表 10. 数据类型、类和结构之间的映射

类别	DB2Types 类和结构	DB2Type 数据类型	DB2 数据类型	Informix 数据类型	.NET 数据类型
二进制数据	DB2Binary	Binary	CHAR FOR BIT DATA		Byte[]
	DB2Binary	Binary[7]	BINARY		Byte[]
	DB2Binary	VarBinary[7]	VARBINARY		Byte[]
	DB2Binary	LongVarBinary[5]	LONG VARCHAR FOR BIT DATA		Byte[]
字符数据	DB2String	Char	CHAR	CHAR	String
	DB2String	VarChar	VARCHAR	VARCHAR	String
	DB2String	LongVarChar[5]	LONG VARCHAR	LVARCHAR	String
图形数据	DB2String	Graphic	GRAPHIC		String
	DB2String	VarGraphic	VARGRAPHIC		String
	DB2String	LongVarGraphic[5]	LONG VARGRAPHIC		String

5. DB2 .NET 公共语言运行时例程不支持将这些数据类型用作参数。

6. 类型为 DB2Type.Xml 的 DB2ParameterClass.ParameterName 属性可以接受下列类型的变量: String、byte[]、DB2Xml 以及 XmlReader。

7. 这些数据类型只适用于 DB2 z/OS 版。

8. 仅 DB2 z/OS 版 V9 和更高发行版以及 DB2 Database for Linux, UNIX, and Windows V9.5 和更高发行版支持此数据类型。

9. Date 对象和 Time 对象可以是时间戳记字符串文字。Timestamp 对象可以是日期字符串文字

表 10. 数据类型、类和结构之间的映射 (续)

类别	DB2Types 类和结构	DB2Type 数据类型	DB2 数据类型	Informix 数据类型	.NET 数据类型
LOB 数据	DB2Clob	Clob	CLOB	CLOB 和 TEXT	String
	DB2Blob	Blob	BLOB	BLOB 和 BYTE	Byte[]
	DB2Clob	DbClob	DBCLOB		String
数字数据	DB2Int16	SmallInt	SMALLINT	BOOLEAN 和 SMALLINT	Int16
	DB2Int32	Integer	INT	INTEGER、INT 和 SERIAL	Int32
	DB2Int64	BigInt 和 BigSerial	BIGINT	BIGINT、BIGSERIAL、INT8 和 SERIAL8	Int64
	DB2Real 和 DB2Real370	Real	REAL	REAL 和 SMALLFLOAT	Single
	DB2Double	Double	DOUBLE PRECISION	DECIMAL (≤ 29) 和 DOUBLE PRECISION	Double
	DB2Double	Float	FLOAT	DECIMAL (32) 和 FLOAT	Double
	DB2Decimal	Decimal	DECIMAL	MONEY	Decimal
	DB2DecimalFloat	DecimalFloat	DECFLOAT(16|34)[58]		Decimal
	DB2Decimal	Numeric	DECIMAL	DECIMAL (≤ 29) 和 NUMERIC	Decimal
日期/时间数据	DB2Date	Date	DATE	DATETIME（日期精度）	DateTime String[9]
	DB2Time	Time	TIME	DATETIME（时间精度）	TimeSpan String[9]
	DB2TimeStamp	Timestamp	TIMESTAMP	DATETIME（时间和日期精度）	DateTime String[9]
	DB2TimeStamp Offset	TimestampWithTimeZone	TIMESTAMP WITH TIME ZONE	不适用	DateTimeOffset String[9]
行标识数据	DB2RowId	RowId	ROWID		Byte[]
XML 数据	DB2Xml	Xml[6]	XML		Byte[]

使用 IBM Data Server Provider for .NET 从应用程序中执行 SQL 语句

使用 IBM Data Server Provider for .NET 时，会通过 DB2Command 类（使用其方法 ExecuteReader() 和 ExecuteNonQuery() 及其属性 CommandText、CommandType 和 Transaction）执行 SQL 语句。

关于此任务

对于生成输出的 SQL 语句，应使用 ExecuteReader() 方法，并且可通过 DB2DataReader 对象检索其结果。对于所有其他 SQL 语句，应该使用 ExecuteNonQuery() 方法。DB2Command 对象的 Transaction 属性应初始化为 DB2Transaction 对象。DB2Transaction 对象负责回滚和落实数据库事务。

使用 C# 执行 UPDATE 语句:

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";

cmd.ExecuteNonQuery();
```

在 Visual Basic .NET 中执行 UPDATE 语句:

```
' assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";
cmd.ExecuteNonQuery();
```

使用 C# 执行 SELECT 语句:

```
// assume a DB2Connection conn
DB2Command cmd = conn.CreateCommand();
DB2Transaction trans = conn.BeginTransaction();
cmd.Transaction = trans;
cmd.CommandText = "SELECT deptnumb, location " +
                  "  FROM org " +
                  "  WHERE deptnumb < 25";

DB2DataReader reader = cmd.ExecuteReader();
```

在 Visual Basic .NET 中执行 SELECT 语句:

```
' assume a DB2Connection conn
Dim cmd As DB2Command = conn.CreateCommand()
Dim trans As DB2Transaction = conn.BeginTransaction()
cmd.Transaction = trans
cmd.CommandText = "UPDATE staff " +
                  "  SET salary = (SELECT MIN(salary) " +
                  "                  FROM staff " +
    "                  WHERE id >= 310) " +
                  "  WHERE id = 310";

cmd.ExecuteNonQuery()
```

应用程序已执行数据库事务后，必须使其回滚或落实。这是通过 DB2Transaction 对象的 Commit() 和 Rollback() 方法完成的。

使用 C# 回滚或落实事务:

```
// assume a DB2Transaction object conn
trans.Rollback();
...
trans.Commit();
```

在 Visual Basic .NET 中回滚或落实事务:

```
' assume a DB2Transaction object conn
trans.Rollback()
...
trans.Commit()
```

使用 IBM Data Server Provider for .NET 从应用程序中读取结果集

使用 IBM Data Server Provider for .NET 时，会通过 DB2DataReader 对象读取结果集。DB2DataReader 方法 Read() 用来进至结果集中的下一行。

关于此任务

方法 GetString()、GetInt32()、GetDecimal() 和用于所有可用数据类型的其他方法可用来从输出的各列抽取数据。DB2DataReader 的 Close() 方法用于关闭 DB2DataReader 对象，后者应始终在读取完输出时完成。

使用 C# 读取结果集:

```
// assume a DB2DataReader reader
Int16 deptnum = 0;
String location="";

// Output the results of the query
while(reader.Read())
{
  deptnum = reader.GetInt16(0);
  location = reader.GetString(1);
  Console.WriteLine("    " + deptnum + " " + location);
}
reader.Close();
```

使用 Visual Basic .NET 读取结果集:

```
' assume a DB2DataReader reader
Dim deptnum As Int16 = 0
Dim location As String ""

' Output the results of the query
Do While (reader.Read())
  deptnum = reader.GetInt16(0)
  location = reader.GetString(1)
  Console.WriteLine("    " & deptnum & " " & location)
Loop
reader.Close();
```

使用 IBM Data Server Provider for .NET 从应用程序中调用存储过程

使用 IBM Data Server Provider for .NET 时，可使用 DB2Command 对象来调用存储过程。

关于此任务

CommandType 属性的缺省值是 CommandType.Text。此值适用于 SQL 语句，并且还可用于调用存储过程。但是，将 CommandType 设置为 CommandType.StoredProcedure 时，可以更方便地调用存储过程。在这种情况下，您只需要指定存储过程名以及任何参数。

使用存储过程时，可使用主变量、命名参数或定位参数来传递参数。但是，它们不能在同一 SQL 语句中组合使用。

以下 C# 和 Visual Basic 示例演示如何在将 CommandType 属性设置为 CommandType.StoredProcedure 或 CommandType.Text 的情况下调用名为 INOUT_PARAM 的存储过程。

过程

- 使用 C# 时，通过将 DB2Command 的 CommandType 属性设置为 CommandType.Text 来调用存储过程:

```
// assume a DB2Connection conn
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
String procCall = "CALL INOUT_PARAM (@param1, @param2, @param3)";
cmd.Transaction = trans;
cmd.CommandType = CommandType.Text;
cmd.CommandText = procCall;

// Register input-output and output parameters for the DB2Command
cmd.Parameters.Add( new DB2Parameter("@param1", "Value1");
cmd.Parameters.Add( new DB2Parameter("@param2", "Value2");
DB2Parameter param3 = new DB2Parameter("@param3", IfxType.Integer);
param3.Direction = ParameterDirection.Output;
cmd.Parameters.Add( param3 );

// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

- 使用 Visual Basic 时，通过将 DB2Command 的 CommandType 属性设置为 CommandType.Text 来调用存储过程:

```
' assume a DB2Connection conn
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
Dim procCall As String = "CALL INOUT_PARAM (?, ?, ?)"
cmd.Transaction = trans
cmd.CommandType = CommandType.Text
cmd.CommandText = procCall

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

  注: CALL 和 EXECUTE PROCEDURE 都受支持。

- 使用 C# 时，通过将 DB2Command 的 CommandType 属性设置为 CommandType.StoredProcedure 来调用存储过程。使用此方法时不支持使用命名参数:

```
// assume a DB2Connection conn
DB2Transaction trans = conn.BeginTransaction();
DB2Command cmd = conn.CreateCommand();
String procName = "INOUT_PARAM";
cmd.Transaction = trans;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = procName;

// Register input-output and output parameters for the DB2Command
...
```

```
// Call the stored procedure
Console.WriteLine("  Call stored procedure named " + procName);
cmd.ExecuteNonQuery();
```

- 使用 Visual Basic 时，通过将 DB2Command 的 CommandType 属性设置为 CommandType.StoredProcedure 来调用存储过程:

```
' assume a DB2Connection conn
Dim trans As DB2Transaction = conn.BeginTransaction()
Dim cmd As DB2Command = conn.CreateCommand()
Dim procName As String = "INOUT_PARAM"
cmd.Transaction = trans
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = procName

' Register input-output and output parameters for the DB2Command
...

' Call the stored procedure
Console.WriteLine("  Call stored procedure named " & procName)
cmd.ExecuteNonQuery()
```

- 通过参数名进行限定时，可按任何顺序将参数传递至存储过程。仅 DB2 for Linux, UNIX and Windows 数据服务器支持此命名参数功能。例如，在以下 SQL 语句中，定义了存储过程，然后带按另一顺序排列的参数调用了该存储过程:

```
CREATE PROCEDURE schema.my_proc ( IN var1 int, INOUT var2 int )
 LANGUAGE SQL
 BEGIN
  -- procedure code here
 END

CALL my_proc (var2=>@param2, var1=>@param1)
```

同时访问 CURSOR 类型输出参数返回的结果集

使用 IBM Data Server Provider for .NET 时，DB2Type.Cursor 指定为同时访问输出参数中的所有游标。

关于此任务

对于具有多个 CURSOR 类型输出参数的存储过程，将 DB2TYPE.Cursor 绑定至输出参数对象允许同时访问输出参数中的所有游标。

例如，OrderDetails 存储过程声明三个游标，每个游标给出有关产品及其销售的相关信息。

```
CREATE OR REPLACE TYPE cur AS CURSOR
CREATE PROCEDURE OrderDetails (p_startDate TIMESTAMP, p_endDate TIMESTAMP,
  OUT prodDetails cur, OUT prodOrderDetails cur, OUT custOrderDetails cur)
LANGUAGE SQL
BEGIN
  SET prodDetails = CURSOR WITH HOLD FOR
    SELECT p.pid, price, quantity FROM products p, inventory i
      WHERE p.pid = i.pid AND p.pid IN (SELECT DISTINCT pid FROM orders) ORDER BY pid;
  SET prodOrderDetails = CURSOR WITH HOLD FOR
    SELECT pid, COUNT(*), SUM (quantity) FROM orders
      WHERE date >= p_startDate AND date <= p_endDate GROUP BY pid ORDER BY pid;
  SET custOrderDetails = CURSOR WITH HOLD FOR
    SELECT pid, custID, COUNT(*), SUM(quantity) FROM orders
      WHERE date >= p_startDate AND date <= p_endDate
      GROUP BY pid, custID ORDER by pid, custID;
  OPEN prodDetails;
  OPEN prodOrderDetails;
  OPEN custOrderDetails;
END;
```

调用程序需要同时访问这些游标，以便它可通过每个游标收集特定产品的相关信息并计算折扣。为提供对游标的同时访问，存储过程将这些游标作为输出参数返回。绑定 CURSOR 类型输出参数时，该应用程序必须将 DB2Type 设置为 DB2Type.Cursor 才能进行同时访问。

```
//C# Code sample
cmd.CommandText = "CALL OrderDetails(
      @p_startDate, @p_endDate, @prodDetails, @prodOrderDetails, @custOrderDetails)";
cmd.Parameters.Add("@p_startDate", DateTime.Parse("1/1/2010"));
cmd.Parameters.Add("@p_endDate", DateTime.Parse("12/31/2010"));
cmd.Parameters.Add("@prodDetails", DB2Type.Cursor);
cmd.Parameters["@prodDetails"].Direction = ParameterDirection.Output;
cmd.Parameters.Add("@prodOrderDetails", DB2Type.Cursor);
cmd.Parameters["@prodOrderDetails"].Direction = ParameterDirection.Output;
cmd.Parameters.Add("@custOrderDetails", DB2Type.Cursor);
cmd.Parameters["@custOrderDetails"].Direction = ParameterDirection.Output;
cmd.ExecuteNonQuery();
DB2DataReader prodDetailsDR =
    (DB2DataReader)cmd.Parameters["@prodDetails"].Value;
DB2DataReader prodOrderDetailsDR =
    (DB2DataReader)cmd.Parameters["@prodOrderDetails"].Value;
DB2DataReader custOrderDetailsDR =
    (DB2DataReader)cmd.Parameters["@custOrderDetails"].Value;

while (prodOrderDetailsDR.Read())
{
    pid = prodOrderDetailsDR.GetInt32(0);
    numOrders = prodOrderDetailsDR.GetInt32(1);
    totalOrderQuantity = prodOrderDetailsDR.GetInt32(2);
    prodDetailsDR.Read();
    price = prodDetailsDR.GetDecimal(1);
    currentInventory = prodDetailsDR.GetInt32(2);
    int totalCustOrders = 0;
    while (custOrderDetailsDR.Read())
    {
        custID = custOrderDetailsDR.GetInt32(1);
        numOrdersByCust = custOrderDetailsDR.GetInt32(2);
        totalCustOrders += numOrdersByCust;
        totalOrderQuantityByCust = custOrderDetailsDR.GetInt32(3);
        //Calculate discount based on numOrders, numOrdersByCust,
        //  totalOrderQuantity, totalOrderQuantityByCust, price and currentInventory
        if (totalCustOrders == numOrders) //done with this pid
                                  break;
    }
}
prodDetailsDR.Close();
prodOrderDetailsDR .Close();
custOrderDetailsDR .Close();
```

只能在调用 ExecuteNonQuery 方法后通过 Value 属性访问游标类型输出参数中的数据阅读器。

如果使用 ExecuteReader 或 ExecuteResultSet 方法执行该命令，那么会在 DB2DataReader 或 DB2ResultSet 对象中返回结果集。后续结果集必须通过调用 NextResult 方法顺序访问。尽管已绑定输出参数，但访问输出参数 Value 属性还是会导致 InvalidOperation 异常，因为未使用 ExecuteNonQuery 方法执行该查询。

同步使用游标时，该应用程序可能需要先落实工作，然后继续读取游标。应用程序要发出落实命令而不损坏已打开的游标，该游标必须在存储过程内声明为可挂起。

使用 pureQuery 来优化 .NET 应用程序中的查询

.NET 客户机驱动程序可以利用 pureQuery 技术中的功能。这些功能使现有的 .NET 应用程序查询能够作为静态 SQL 执行。静态查询可以避免需要在运行时准备某些语句。这有助于提高应用程序的安全性和性能。

开始之前

为了利用 pureQuery 技术功能部件，您必须使用 IBM InfoSphere® Optim™ pureQuery Runtime 2.1 或更高版本启用 IBM Data Server Provider for .NET V9.5.3 或更高版本的 pureQuery。

关于此任务

本任务将完成用于发现和使用 .NET 应用程序的静态查询的基本步骤。

过程

1. 设置 .NET 应用程序以捕获潜在的语句:
   a. 找到用于管理数据库连接的代码部分。
   b. 将 captureMode 关键字设置为 on。
   c. 如果设置了 executionMode 关键字，请确保将值设置为 dynamic。
   d. 将 collection 设置为程序包名的集合名称（collection.rootPkgName）。
   e. 将 rootPkgName 设置为程序包名的根程序包（collection.rootPkgName）。
   f. 将 pureQueryXML 关键字设置为用于存储所捕获语句的路径和文件名。
   g. 执行应用程序，以便在指定的 pureQueryXML 文件中捕获语句。
   h. 在命令提示符中，运行 **db2cap** 实用程序以便将捕获文件与数据库绑定。**db2cap** 命令有几个需要传递的参数。
2. 更改 .NET 应用程序，以便以静态方式运行所捕获的语句:
   a. 找到用于管理数据库连接的代码部分。
   b. 除去 captureMode 关键字或者将其设置为 off。
   c. 将 executionMode 关键字设置为 static。

结果

现在，.NET 应用程序应该以静态方式执行所捕获的语句。未能捕获的语句将继续以动态方式执行。

示例

以下代码是设置连接字符串以捕获执行的语句的示例。

```
[C#]
string myConnectionString =
    "Database=Sample;captureMode=ON;pureQueryXML=c:\temp\capfile.xml";
DB2Connection myConn = new DB2Connection(myConnectionString);
string myInsertQuery = "INSERT INTO STAFF (ID, NAME) Values(...)";
DB2Command myDB2Command = new DB2Command(myInsertQuery);
myDB2Command.Connection = myConn;
myConn.Open();
myDB2Command.ExecuteNonQuery();
myConn.Close();

[Visual Basic]
Dim myConnectionString As String = _
    "Database=Sample;captureMode=ON;pureQueryXML=c:\temp\capfile.xml"
Dim myConn As New DB2Connection(myConnectionString)
Dim myInsertQuery As String = "INSERT INTO STAFF (ID, NAME) Values(...)"
Dim myDB2Command As New DB2Command(myInsertQuery)
myDB2Command.Connection = myConn
myConn.Open()
myDB2Command.ExecuteNonQuery()
myConn.Close()
```

下一个示例更改连接字符串，以便以静态方式运行所捕获的语句。

```
[C#]
string myConnectionString =
    "Database=Sample;executionMode=STATIC;pureQueryXML=c:\temp\capfile.xml";
DB2Connection myConn = new DB2Connection(myConnectionString);
string myInsertQuery = "INSERT INTO STAFF (ID, NAME) Values(...)";
DB2Command myDB2Command = new DB2Command(myInsertQuery);
myDB2Command.Connection = myConn;
myConn.Open();
myDB2Command.ExecuteNonQuery();
myConn.Close();

[Visual Basic]
Dim myConnectionString As String = _
    "Database=Sample;captureMode=ON;pureQueryXML=c:\temp\capfile.xml"
Dim myConn As New DB2Connection(myConnectionString)
Dim myInsertQuery As String = "INSERT INTO STAFF (ID, NAME) Values(...)"
Dim myDB2Command As New DB2Command(myInsertQuery)
myDB2Command.Connection = myConn
myConn.Open()
myDB2Command.ExecuteNonQuery()
myConn.Close()
```

对 .NET 应用程序启用 pureQuery

Optim pureQuery 是必须购买才允许使用的许可功能部件。您可以通过购买 IBM InfoSphere Optim pureQuery Runtime 产品获得 pureQuery 功能部件。

开始之前

在尝试启用 pureQuery 之前，您必须已安装 IBM InfoSphere Optim pureQuery Runtime 2.1 或更版本。您还必须至少已安装包括 IBM 数据服务器 .NET 提供程序 的下列产品中的一个:

- IBM 数据服务器 ODBC、CLI 和 .NET 驱动程序 V9.5.3 或者 IBM Data Server Driver Package V9.5.4（或更高版本）
- IBM 数据服务器运行时客户机 V9.5.3（或更高版本）
- IBM 数据服务器客户机 V9.5.3（或更高版本）
- DB2 for Linux, UNIX, and Windows V9.5 FP3（或更高版本）

关于此任务

本任务对 .NET 应用程序启用 pureQuery。完成本任务之后，您将能够在 .NET 应用程序中使用 pureQuery 技术的功能。

过程

1. 激活用于 .NET 的 pureQuery 许可证文件。
   - 对于 IBM Data Server Driver Package:
     a. 在 pureQuery 安装目录中找到正确的许可证文件版本。例如，`C:\Program Files\IBM\purequery\license\clientv97\`。

        注: 可以使用驱动程序软件包中提供的 **db2level** 命令来确定驱动程序软件包的版本。

     b. 将许可证文件 `dspq_rt.lic` 复制到 IBM Data Server Driver Package 许可证目录。您可以在 IBM Data Server Driver Package 的安装目录中找到许可证目录。例如，`C:\Program Files\IBM\IBM DATA SERVER DRIVER\license\`。

- 对于其他 DB2 客户机或数据服务器:
  - 在命令提示符中，执行以下命令: `db2licm -a C:\pqRuntime21\pureQuery\dspq_rt.lic`，其中 `C:\pqRuntime21\` 是您的 pureQuery 运行时安装的目录。

2. 对于 Microsoft Windows Vista 或 Windows 7 操作系统，要完成 pureQuery 许可证文件的激活，您必须执行下列其中一个步骤。这些步骤只需要运行一次。在不采取其中的一个步骤的情况下尝试在 Windows Vista 或 Windows 7 操作系统上使用 pureQuery 功能部件将导致一个错误，该错误陈述无法找到有效许可证密钥。
   - 在管理员帐户下运行您的第一个捕获。使用管理员帐户成功完成第一个捕获后，将为系统上的用户启用 pureQuery 功能部件。
   - 配置 IBM Data Server Driver Package 许可证目录以允许对将使用 pureQuery 功能部件的用户帐户的写访问。

结果

已经为驱动程序激活 pureQuery 许可证文件。现在，.NET 应用程序可以利用客户机优化功能了。

下一步做什么

激活 pureQuery 许可证文件之后，您可以开始使用 pureQuery 为 .NET 应用程序优化数据提供程序。

提供对 Microsoft Entity Framework 的支持

通过 IBM Data Server Provider for .NET，使用 IBM 数据服务器来利用 Microsoft ADO.NET Entity Framework。可使用受支持服务器版本来生成 EDM 模式、对实体应用程序写入和执行实体 SQL 和 LINQ 语句。

系统要求

IBM Data Server Provider for .NET 使用以下 IBM 数据服务器:

- DB2 Database for Linux, UNIX, and Windows V8 到 V9.8
- IBM Data Server Client V9.5.3 或更高版本
- IBM Data Server Runtime Client V9.5.3 或更高版本
- IBM 数据服务器 ODBC、CLI 和 .NET 驱动程序 V9.5.3 或 IBM Data Server Driver Package V9.5.4 或更高版本
- DB2 通用数据库 AS/400® 版 V5R4、V6R1 和 V7R1，通过 DB2 Connect（对于 IBM DB2 V9.7 FP4 及更高版本）
- 用于 iSeries® 的 DB2 通用数据库 AS/400 版 V5R4 和 V6R1，通过 DB2 Connect（对于 IBM DB2 V9.7 FP3 和更低版本）
- DB2 z/OS 版 V8 到 V10
- IBM Informix V11.170 或更高版本

必须具有带 Microsoft ADO.NET Entity Framework 的 Microsoft .NET Framework 3.5 SP1。要使用 Microsoft Entity Data Model 向导或 ADO.NET Entity Designer 来处理实体数据模型，还需要 Microsoft Visual Studio 2008 或更高版本。对带有 Visual Studio 2010 的 .NET Framework 4.0 的支持是从 DB2 for Linux, UNIX, and Windows V9.7 FP4 开始引入的。

下表列示 IBM 实体提供程序支持的规范函数。该数据提供程序会将规范函数转换为对应数据源函数。

表 11. IBM 实体提供程序对规范函数的支持

规范函数类型	LINQ 函数	DB2 for Linux, UNIX and Windows	DB2 for z/OS	DB2 for IBM i	Informix
Aggregate	Average	Y	Y	Y	Y
	BigCount	Y	Y	Y	Y
	Count	Y	Y	Y	Y
	Maximum	Y	Y	Y	Y
	Minimum	Y	Y	Y	Y
	NewGuid[1]	Y*	Y*	Y*	Y*
	StDev	Y	Y	Y	Y
	StDevP	Y	Y	Y	Y
	Sum	Y	Y	Y	Y
	Var	Y	Y	Y	Y
	VarP	Y	Y	Y	Y
Bitwise	BitWiseAnd[1]	Y	Y*	Y*	Y
	BitWiseNot[1]	Y	Y*	Y*	Y
	BitWiseOr[1]	Y	Y*	Y*	Y
	BitWiseXor[1]	Y	Y*	Y*	Y
Math	Abs	Y	Y	Y	Y
	Ceiling	Y	Y	Y	Y
	Floor	Y	Y	Y	Y
	Power	Y	Y	Y	Y
	Round (value,digits)	Y	Y	Y	Y
	Truncate (value,digits)	Y	Y	Y	Y
String	Concat	Y	Y	Y	Y
	Contains[1]	Y	Y	Y	Y*
	EndsWith	Y	Y	Y	Y
	IndexOf[1]	Y	Y	Y	Y*
	Left	Y	Y	Y	Y
	Length	Y	Y	Y	Y
	LTrim	Y	Y	Y	Y
	Replace	Y	Y	Y	Y
	Right	Y	Y	Y	Y
	RTrim	Y	Y	Y	Y
	StartsWith	Y	Y	Y	Y
	Substring	Y	Y	Y	Y
	ToLower	Y	Y	Y	Y
	ToUpper	Y	Y	Y	Y
	Trim	Y	Y	Y	Y

表 11. IBM 实体提供程序对规范函数的支持 (续)

规范函数类型	LINQ 函数	DB2 for Linux, UNIX and Windows	DB2 for z/OS	DB2 for IBM i	Informix
Datetime	AddNanoseconds	Y	Y	Y	Y
	AddMicroseconds	Y	Y	Y	Y
	AddMilliseconds	Y	Y	Y	Y
	AddSeconds	Y	Y	Y	Y
	AddMinutes	Y	Y	Y	Y
	AddHours	Y	Y	Y	Y
	AddDays	Y	Y	Y	Y
	AddMonths	Y	Y	Y	Y
	AddYears	Y	Y	Y	Y
	CreateDateTime	Y	Y	Y	Y
	CreateDateTimeOffset		Y		
	CurrentDateTimeOffset[1]		Y		
	CreateTime	Y	Y	Y	Y
	CurrentDateTime	Y	Y	Y	Y
	CurrentUtcDateTime	Y	Y	Y	
	Day	Y	Y	Y	Y
	DayOfYear	Y	Y	Y	Y
	DiffNanoseconds[1]	Y	Y	Y	Y*
	DiffMicroseconds[1]	Y	Y	Y	Y*
	DiffMilliseconds[1]	Y	Y	Y	Y*
	DiffSeconds[1]	Y	Y	Y	Y*
	DiffMinutes[1]	Y	Y	Y	Y*
	DiffHours[1]	Y	Y	Y	Y*
	DiffDays[1]	Y	Y	Y	Y*
	DiffMonths[1]	Y	Y	Y	Y*
	DiffYears[1]	Y	Y	Y	Y*
	GetTotalOffsetMinutes[1]		Y		
	Hour	Y	Y	Y	Y
	Millisecond	Y	Y	Y	Y
	Minute	Y	Y	Y	Y
	Month	Y	Y	Y	Y
	Second	Y	Y	Y	Y
	Truncate (datetime exp)	Y	Y	Y	Y
	Year	Y	Y	Y	Y

已知限制

注:

某些规范函数完全依赖服务器提供的函数支持。如果遇到 SQL0440N* 错误，那么它指示服务器不支持错误消息中提到的函数。请与服务器的 IBM 技术支持人员联系以获取有关受支持功能的更多信息。

1 - 此规范函数仅在 V9.7 FP4 及更高版本中受支持。

通用:

- 仅支持数据库的第一个方案；使用 Entity Framework 之前，所有数据库对象必须已存在
- 不支持存储特定函数的调用

- “服务器资源管理器添加连接”对话框中设置的可信上下文连接属性不会传递至 Entity Framework 连接，这是对 Server Explorer 的公认限制

DB2 for z/OS:

- 数据类型 REAL 不受支持。应用程序需要在表的模式中使用 FLOAT’，或在客户机模式 (EDM) 中将类型指定为 FLOAT，即使服务器上的实际类型为 REAL 也是如此
- 特定于 V8/V7: 可能对包括 Take/Top/First/Intersect/Except 表达式的查询生成异常，用于指示语法错误。未定义包括这些表达式的查询的结果

示例:

```
1) var query = from o in context.Orders
where o.ShipCity == "Seattle"
select o;
var result = query.First();

2) var mexico =
context.OrderDetails.Where(od => od.Order.ShipCountry
   == "Mexico").Select(od => od.Product);
   var canada =
context.OrderDetails.Where(od => od.Order.ShipCountry
   == "Canada").Select(od => od.Product);
   var query = mexico.Intersect(canada);

3) var query =
context.Customers.Select(e => e).Except(context.Orders.Where
   (o => o.ShipCountry == "Mexico").Select(o => o.Customer));

4) var query = context.Orders.Include("OrderDetails").Top("1");

5) var query = context.Orders.Include("OrderDetails").
   Include("OrderDetails.Product").Take(3).Select(o => o);
```

IBM Informix 服务器:

- 可能对带有相关子查询的某些查询生成类似“尚未实现”或“语法错误”的异常。一些典型场景包括:
  1. 带有页面调度的相关子查询。
  2. 基于相关子查询或基于通过浏览生成的集合的任何元素，使用接受元素选择器的分组方法的 Entity Framework 查询（请参阅以下示例）。
  3. 具有基于 REF 构造的 DEREF 构造的查询。

示例:

```
var query = from p in context.Products
group p by p.Category.CategoryID into g
select new
{
g.Key,
ExpensiveProducts = from p2 in g where p2.UnitPrice >
g.Average(p3 => p3.UnitPrice)
select p2
};
```

注:

某些规范函数完全依赖服务器提供的函数支持。如果遇到 SQL0440N* 错误，那么它指示服务器不支持错误消息中提到的函数。请与服务器的 IBM 技术支持人员联系以获取有关受支持功能的更多信息。

使用 Enterprise Library 数据访问模块

Enterprise Library 是应用程序块的集合，用于帮助开发者面对常见的开发挑战。应用程序块作为源代码提供，可以在开发项目按原样使用，也可以针对开发项目进行修改。

您可以从 http://codeplex.com/entlibcontrib/SourceControl/PatchList.aspx 获取用于 IBM 数据服务器的 Enterprise Library 数据访问模块以及其他模块。

有关如何安装 Enterprise Library 数据访问模块并将其与 IBM 数据服务器（DB2、IDS 和 U2）配合使用的信息，请参阅下载程序包中的自述文件。

资源

下面是多个描述了如何使用数据访问模块的在线资源:

- EntLib Contrib Project 主页: http://www.codeplex.com/entlibcontrib
- patterns & practices for Enterprise Library: http://www.codeplex.com/entlib
- Microsoft Enterprise Library 主页: http://msdn.microsoft.com/en-us/library/cc467894.aspx
- IBM DB2 for .NET: http://www.ibm.com/software/data/db2/windows/dotnet.html

构建 .NET 应用程序

构建 Visual Basic .NET 应用程序

DB2 产品提供了用于编译和链接 DB2 Visual Basic .NET 应用程序的 bldapp.bat 批处理文件。此文件与可以使用此文件构建的样本程序一起放在 sqllib\samples\.NET\vb 目录中。该批处理文件采用一个参数（即 %1）来表示要编译的源文件的名称（不带 .vb 扩展名）。

关于此任务

本任务指导您完成使用 bldapp.bat 借助 DbAuth 样本文件构建 Visual Basic .NET 应用程序的基本步骤。

过程

要根据源文件 DbAuth.vb 来构建程序 DbAuth，请输入以下命令:

```
bldapp DbAuth
```

要确保您有运行可执行文件时所需的参数，可以根据输入的参数数目指定不同的参数组合:

1. 无参数。请仅输入程序名:

   ```
   DbAuth
   ```

2. 1 个参数。请输入程序名和数据库别名:

   ```
   DbAuth <db_alias>
   ```

3. 2 个参数。请输入程序名以及用户标识和密码:

   ```
   DbAuth <userid> <passwd>
   ```

4. 3 个参数。请输入程序名以及数据库别名、用户标识和密码:

```
DbAuth <db_alias> <userid> <passwd>
```

5. 4 个参数。请输入程序名以及服务器名称、端口号、用户标识和密码:

```
DbAuth <server> <portnum> <userid> <passwd>
```

6. 5 个参数。请输入程序名以及数据库别名、服务器名称、端口号、用户标识和密码:

```
DbAuth <db_alias> <server> <portnum> <userid> <passwd>
```

下一步做什么

要构建和运行 LCTrans 样本程序，您需要遵循源文件 LCTrans.vb 中提供的更详细指示信息。

构建 C# .NET 应用程序

DB2 产品提供了用于编译和链接 DB2 C# .NET 应用程序的 bldapp.bat 批处理文件。此批处理文件与可以使用此文件构建的样本程序一起放在 sqllib\samples\.NET\cs 目录中。

该批处理文件采用一个参数（即 %1）来表示要编译的源文件的名称（不带 .cs 扩展名）。

关于此任务

本任务指导您完成使用 bldapp.bat 借助 DbAuth 样本文件构建 C# .NET 应用程序的基本步骤。

过程

要根据源文件 DbAuth.cs 来构建程序 DbAuth，请输入以下命令:

```
bldapp DbAuth
```

要确保您有运行可执行文件时所需的参数，可以根据输入的参数数目指定不同的参数组合:

1. 无参数。请仅输入程序名:

```
DbAuth
```

2. 1 个参数。请输入程序名和数据库别名:

```
DbAuth <db_alias>
```

3. 2 个参数。请输入程序名以及用户标识和密码:

```
DbAuth <userid> <passwd>
```

4. 3 个参数。请输入程序名以及数据库别名、用户标识和密码:

```
DbAuth <db_alias> <userid> <passwd>
```

5. 4 个参数。请输入程序名以及服务器名称、端口号、用户标识和密码:

```
DbAuth <server> <portnum> <userid> <passwd>
```

6. 5 个参数。请输入程序名以及数据库别名、服务器名称、端口号、用户标识和密码:

```
DbAuth <db_alias> <server> <portnum> <userid> <passwd>
```

下一步做什么

要构建和运行 LCTrans 样本程序，您需要遵循源文件 LCTrans.cs 中提供的更详细指示信息。

Visual Basic .NET 应用程序的编译和链接选项

本主题描述编译和链接 Visual Basic .NET 应用程序时的各个可用选项。

在 Windows 上使用 Microsoft Visual Basic .NET 编译器构建 Visual Basic .NET 应用程序时，使用下列编译和链接选项（如在 bldapp.bat 批处理文件中所述）。

注: .NET Framework V1.1 仅在与 .NET Provider V9.5 及更低版本配合使用时受支持。

使用 bldapp 的独立 VB .NET 应用程序的编译和链接选项

独立 VB .NET 应用程序的编译和链接选项:

%BLDCOMP%

编译器的变量。缺省值为 vbc，即 Microsoft Visual Basic .NET 编译器。

/r:"%DB2PATH%"\bin\%VERSION%\IBM.Data.DB2.dll

引用您正在使用的 .NET Framework 版本的 DB2 动态链接库。

%DB2PATH%

%DB2PATH% 变量表示 DB2 产品安装的根路径。%DB2PATH% 变量未出现在 IBM 数据服务器 ODBC 和 CLI 驱动程序 或数据服务器驱动程序包 安装上。使用 IBM IBM 数据服务器 ODBC 和 CLI 驱动程序 或数据服务器驱动程序包 时，将 %DB2PATH% 替换为安装了驱动程序产品的路径。

%VERSION%

应用程序支持多个版本的 .NET Framework。对于每个版本，DB2 都有一个动态链接库。对于 .NET Framework V2.0、3.0 和 3.5，%VERSION% 指向 netf20\ 子目录。

使用 bldapp 的松散耦合样本程序 LCTrans 的编译和链接选项:

%BLDCOMP%

编译器的变量。缺省值为 vbc，即 Microsoft Visual Basic .NET 编译器。

/out:RootCOM.dll

根据 RootCOM.vb 源文件输出 LCTrans 应用程序所使用的 RootCOM 动态链接库。

/out:SubCOM.dll

根据 SubCOM.vb 源文件输出 LCTrans 应用程序所使用的 SubCOM 动态链接库。

/target:library %1.cs

根据输入源文件（RootCOM.vb 或 SubCOM.vb）来创建动态链接库。

/r:System.EnterpriseServices.dll

引用 Microsoft Windows 系统企业服务数据链路库。

/r:"%DB2PATH%"\bin\%VERSION%\IBM.Data.DB2.dll

引用您正在使用的 .NET Framework 版本的 DB2 动态链接库。

%DB2PATH%

%DB2PATH% 变量表示 DB2 产品安装的根路径。%DB2PATH% 变量未出现在 IBM 数据服务器 ODBC 和 CLI 驱动程序 或数据服务器驱动程序

包 安装上。使用 IBM IBM 数据服务器 ODBC 和 CLI 驱动程序 或数据服务器驱动程序包 时，将 %DB2PATH% 替换为安装了驱动程序产品的路径。

%VERSION%
应用程序支持多个版本的 .NET Framework。对于每个版本，DB2 都有一个动态链接库。对于 .NET Framework V2.0 和 3.0，%VERSION% 指向 netf20\ 子目录。

/r:System.Data.dll
引用 Microsoft Windows 系统数据动态链接库。

/r:System.dll
引用 Microsoft Windows 系统动态链接库。

/r:System.Xml.dll
引用 Microsoft Windows 系统 XML 动态链接库（对于 SubCOM.vb）。

/r:SubCOM.dll
引用 SubCOM 动态链接库（对于 RootCOM.vb 和 LCTrans.vb）。

/r:RootCOM.dll
引用 RootCOM 动态链接库（对于 LCTrans.vb）。

请参阅编译器文档，以了解其他编译器选项。

C# .NET 应用程序编译和链接选项

本主题描述编译和链接 C# .NET 应用程序时的各种可用选项。

在 Windows 上使用 Microsoft C# 编译器构建 C++ 应用程序时，使用 DB2 中提供的编译和链接选项（如在 bldapp.bat 批处理文件中所述）。

注: .NET Framework V1.1 仅在与 .NET Provider V9.5 及更低版本配合使用时受支持。

使用 bldapp 的独立 C# 应用程序的编译和链接选项:

独立 C# 应用程序的编译和链接选项:

%BLDCOMP%
编译器的变量。缺省值为 csc，即 Microsoft C# 编译器。

/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll
引用您正在使用的 .NET Framework 版本的 DB2 动态链接库。

%VERSION%
应用程序支持多个版本的 .NET Framework。对于每个版本，DB2 都有一个动态链接库。对于 .NET Framework V2.0、3.0 和 3.5，%VERSION% 指向 netf20\ 子目录。

使用 bldapp 的松散耦合样本程序 LCTrans 的编译和链接选项:

%BLDCOMP%
编译器的变量。缺省值为 csc，即 Microsoft C# 编译器。

/out:RootCOM.dll
根据 RootCOM.cs 源文件输出 LCTrans 应用程序所使用的 RootCOM 动态链接库。

/out:SubCOM.dll
根据 SubCOM.cs 源文件输出 LCTrans 应用程序所使用的 SubCOM 动态链接库。

/target:library %1.cs
根据输入源文件（RootCOM.cs 或 SubCOM.cs）来创建动态链接库。

/r:System.EnterpriseServices.dll
引用 Microsoft Windows 系统企业服务数据链路库。

/r:"%DB2PATH%"\bin\%VERSION%IBM.Data.DB2.dll
引用您正在使用的 .NET Framework 版本的 DB2 动态链接库。

%VERSION%
应用程序支持多个版本的 .NET Framework。对于每个版本，DB2 都有一个动态链接库。对于 .NET Framework V2.0、3.0 和 3.5，%VERSION% 指向 netf20\ 子目录。

/r:System.Data.dll
引用 Microsoft Windows 系统数据动态链接库。

/r:System.dll
引用 Microsoft Windows 系统动态链接库。

/r:System.Xml.dll
引用 Microsoft Windows 系统 XML 动态链接库（对于 SubCOM.cs）。

/r:SubCOM.dll
引用 SubCOM 动态链接库（对于 RootCOM.cs 和 LCTrans.cs）。

/r:RootCOM.dll
引用 RootCOM 动态链接库（对于 LCTrans.cs）。

请参阅编译器文档，以了解其他编译器选项。

第 5 章 IBM OLE DB Provider for DB2

IBM OLE DB Provider for DB2 允许 DB2 充当 OLE DB 提供程序的资源管理器。此支持使基于 OLE DB 的应用程序能够使用 OLE 接口来抽取或查询 DB2 数据。

Microsoft OLE DB 是一组 OLE/COM 接口，它们使应用程序能够对各种信息源中存储的数据进行一致的访问。OLE DB 体系结构定义了 OLE DB 使用者和 OLE DB 提供程序。OLE DB 使用者是任何使用 OLE DB 接口的系统或应用程序；OLE DB 提供程序是任何提供 OLE DB 接口的组件。

IBM OLE DB Provider for DB2 的提供程序名称为 IBMDADB2，它使 OLE DB 使用者能够访问 DB2 数据库服务器上的数据。如果已安装 DB2 Connect，那么这些 OLE DB 使用者还可以访问主机 DBMS（例如 DB2 z/OS 版、DB2 服务器 VM 和 VSE 版或 DB2 通用数据库 AS/400 版）上的数据。

IBM OLE DB Provider for DB2 提供了下列功能:

- 支持第 0 级 OLE DB 提供程序规范，其中包括某些附加的第 1 级接口。
- 自由的线程化提供程序实现，此实现使应用程序能够在一个线程中创建组件并在任何其他线程中使用那些组件。
- 错误查找服务，此服务用于返回 DB2 错误消息。

注意，IBM OLE DB Provider 驻留在客户机上，并且与同样受 DB2 数据库系统支持的 OLE DB 表功能有所不同。

本文档的后续章节将描述 IBM OLE DB Provider for DB2 的特定实现。有关 Microsoft OLE DB 2.0 规范的更多信息，请参阅 Microsoft Press 提供的 Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK。

版本一致性

IBM OLE DB Provider for DB2 与 Microsoft OLE DB 规范的 V2.7 或更高版本一致。

系统要求

请参阅 IBM OLE DB Provider for DB2 数据服务器的声明函，以了解受支持的 Windows 操作系统。

要安装 IBM OLE DB Provider for DB2，必须先在上面列示的其中一款受支持操作系统中运行。您还需要安装一个完整的 DB2 产品、IBM 数据服务器 ODBC 和 CLI 驱动程序 或 IBM Data Server Driver Package。

IBM OLE DB Provider for DB2 支持的应用程序类型

使用 IBM OLE DB Provider for DB2，可创建下列类型的应用程序:

- ADO 应用程序，其中包括:
  - Microsoft Visual Studio C++ 应用程序
  - Microsoft Visual Basic 应用程序

- 使用 OLE DB .NET Data Provider 的 ADO.NET 应用程序
- 使用 OLE DB 接口来直接访问 IBMDADB2 的 C/C++ 应用程序，其中包括其数据访问使用者对象由 ATL COM AppWizard 生成的 ATL 应用程序。

OLE DB 服务

IBM OLE DB Provider 所支持的线程模型

IBM OLE DB Provider for DB2 支持自由线程化模型。这允许应用程序在一个线程中创建组件并在任何其他线程中使用那些组件。

通过 IBM OLE DB Provider 进行大对象处理

要通过 IBMDADB2 提供程序获取数据并将数据设置为存储器对象 (DBTYPE_IUNKNOWN)，请使用 ISequentialStream 接口，如下所示:

- 要将存储器对象与参数绑定，DBBINDING 结构中的 DBOBJECT 的 dwFlag 字段只能包含值 STGM_READ。IBMDADB2 将执行所绑定对象的 ISequentialStream 接口的 Read 方法。
- 要从存储器对象中获取数据，应用程序必须运行该存储器对象的 ISequentialStream 接口的 Read 方法。
- 获取数据时，长度部件的值是实际数据的长度，而不是 IUnknown 指针的长度。

IBM OLE DB Provider 所支持的模式行集

下表列示 IDBSchemaRowset 所支持的模式行集。在行集中，不受支持的列将设置为 null。

表 12. IBM OLE DB Provider for DB2 支持的模式行集

受支持的 GUID	受支持的限制	受支持的列	注意
DBSCHEMA _COLUMN_PRIVILEGES	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	COLUMN_NAME GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_COLUMNS	COLUMN_NAME TABLE_NAME TABLE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH COLUMN_DEFAULT COLUMN_FLAGS COLUMN_HASDEFAULT COLUMN_NAME DATA_TYPE DESCRIPTIONIS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION TABLE_NAME TABLE_SCHEMA	

表 12. IBM OLE DB Provider for DB2 支持的模式行集 (续)

受支持的 GUID	受支持的限制	受支持的列	注意
DBSCHEMA_FOREIGN_KEYS	FK_TABLE_NAME FK_TABLE_SCHEMA PK_TABLE_NAME PK_TABLE_SCHEMA	DEFERRABILITY DELETE_RULE FK_COLUMN_NAME FK_NAME FK_TABLE_NAME FK_TABLE_SCHEMA ORDINAL PK_COLUMN_NAME PK_NAME PK_TABLE_NAME PK_TABLE_SCHEMA UPDATE_RULE	必须至少指定下列其中一项限制: PK_TABLE_NAME 或 FK_TABLE_NAME 不允许使用“%”通配符。
DBSCHEMA_INDEXES	TABLE_NAME TABLE_SCHEMA	CARDINALITY CLUSTERED COLLATION COLUMN_NAME INDEX_NAME INDEX_SCHEMA ORDINAL_POSITION PAGES TABLE_NAME TABLE_SCHEMA TYPE UNIQUE	不支持排序顺序。如果指定了排序顺序，那么会将其忽略。
DBSCHEMA_PRIMARY_KEYS	TABLE_NAME TABLE_SCHEMA	COLUMN_NAME ORDINAL PK_NAME TABLE_NAME TABLE_SCHEMA	必须至少指定以下限制: TABLE_NAME 不允许使用“%”通配符。
DBSCHEMA _PROCEDURE_PARAMETERS	PARAMETER_NAME PROCEDURE_NAME PROCEDURE_SCHEMA	CHARACTER_MAXIMUM_LENGTH CHARACTER_OCTET_LENGTH DATA_TYPE DESCRIPTIONIS_NULLABLE NUMERIC_PRECISION NUMERIC_SCALE ORDINAL_POSITION PARAMETER_DEFAULT PARAMETER_HASDEFAULT PARAMETER_NAME PARAMETER_TYPE PROCEDURE_NAME PROCEDURE_SCHEMA TYPE_NAME	
DBSCHEMA_PROCEDURES	PROCEDURE_NAME PROCEDURE_SCHEMA	DESCRIPTIONPROCEDURE_NAME PROCEDURE_SCHEMA PROCEDURE_TYPE	

表 12. IBM OLE DB Provider for DB2 支持的模式行集 (续)

受支持的 GUID	受支持的限制	受支持的列	注意
DBSCHEMA_PROVIDER_TYPES	DATA_TYPE BEST_MATCH	AUTO_UNIQUE_VALUE BEST_MATCH CASE_SENSITIVE CREATE_PARAMS COLUMN_SIZE DATA_TYPE FIXED_PREC_SCALE IS_FIXEDLENGTH IS_LONG IS_NULLABLE LITERAL_PREFIX LITERAL_SUFFIX LOCAL_TYPE_NAME MINIMUM_SCALE MAXIMUM_SCALE SEARCHABLE TYPE_NAMEUNSIGNED_ATTRIBUTE	
DBSCHEMA_STATISTICS	TABLE_NAME TABLE_SCHEMA	CARDINALITY TABLE_NAME TABLE_SCHEMA	不支持排序顺序。如果指定了排序顺序，那么会将其忽略。
DBSCHEMA _TABLE_PRIVILEGES	TABLE_NAME TABLE_SCHEMA	GRANTEE GRANTOR IS_GRANTABLE PRIVILEGE_TYPE TABLE_NAME TABLE_SCHEMA	
DBSCHEMA_TABLES	TABLE_NAME TABLE_SCHEMA TABLE_TYPE	DESCRIPTIONTABLE_NAME TABLE_SCHEMA TABLE_TYPE	

IBM OLE DB Provider 自动启用的 OLE DB 服务

缺省情况下，IBM OLE DB Provider for DB2 通过在提供者的类标识 (CLSID) 下添加 DWORD 值为 0xFFFFFFFF 的注册表条目 OLEDB_SERVICES 来自动启用所有 OLE DB 服务。此值的含义如下:

表 13. OLE DB 服务

启用的服务	DWORD 值
所有服务（缺省）	0xFFFFFFFF
除合用和自动授权以外的所有服务	0xFFFFFFFC
除客户机游标以外的所有服务	0xFFFFFFFB
除合用、授权和游标以外的所有服务	0xFFFFFFF8
无服务	0x000000000

IBM OLE DB Provider 所支持的游标方式

IBM OLE DB Provider for DB2 提供了对只读、只前进、可更新可滚动和可更新可滚动游标的本机支持。

DB2 与 OLE DB 之间的数据类型映射

IBM OLE DB Provider for DB2 支持在 DB2 数据类型与 OLE DB 数据类型之间进行数据类型映射。

下表提供受支持映射以及用于指示列数据类型和参数数据类型的可用名称的完整列表。

表 14. DB2 数据类型与 OLE DB 数据类型之间的数据类型映射

DB2 数据类型	OLE DB 数据类型指示符	OLE DB 标准类型名称	特定于 DB2 的名称
SMALLINT	DBTYPE_I2	“DBTYPE_I2”	“SMALLINT”
INTEGER	DBTYPE_I4	“DBTYPE_I4”	“INTEGER”或“INT”
BIGINT	DBTYPE_I8	“DBTYPE_I8”	“BIGINT”
REAL	DBTYPE_R4	“DBTYPE_R4”	“REAL”
FLOAT	DBTYPE_R8	“DBTYPE_R8”	“FLOAT”
DOUBLE	DBTYPE_R8	“DBTYPE_R8″	“DOUBLE”或“DOUBLE PRECISION”
DECIMAL	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“DEC”或“DECIMAL”
NUMERIC	DBTYPE_NUMERIC	“DBTYPE_NUMERIC”	“NUM”或“NUMERIC”
DATE	DBTYPE_DBDATE	“DBTYPE_DBDATE”	“DATE”
TIME	DBTYPE_DBTIME	“DBTYPE_DBTIME”	“TIME”
TIMESTAMP	DBTYPE_DBTIMESTAMP	“DBTYPE_DBTIMESTAMP”	“TIMESTAMP”
CHAR	DBTYPE_STR	“DBTYPE_CHAR”	“ C H A R ” 或 “CHARACTER”
VARCHAR	DBTYPE_STR	“DBTYPE_VARCHAR”	“VARCHAR”
L O N G VARCHAR	DBTYPE_STR	“DBTYPE_LONGVARCHAR”	“LONG VARCHAR”
CLOB	DBTYPE_STR 和 DBCOLUMNFLAGS_ISLONG 或 DBPARAMFLAGS_ISLONG	“DBTYPE_CHAR” “DBTYPE_VARCHAR” “DBTYPE_LONGVARCHAR” 和 DBCOLUMNFLAGS_ISLONG 或 DBPARAMFLAGS_ISLONG	“CLOB”
GRAPHIC	DBTYPE_WSTR	“DBTYPE_WCHAR”	“GRAPHIC”
VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WVARCHAR”	“VARGRAPHIC”
L O N G VARGRAPHIC	DBTYPE_WSTR	“DBTYPE_WLONGVARCHAR”	“LONG VARGRAPHIC”

表 14. DB2 数据类型与 OLE DB 数据类型之间的数据类型映射 (续)

DB2 数据类型	OLE DB 数据类型指示符	OLE DB 标准类型名称	特定于 DB2 的名称
DBCLOB	DBTYPE_WSTR 和 DBCOLUMNFLAGS_ISLONG 或 DBPARAMFLAGS_ISLONG	“DBTYPE_WCHAR” “DBTYPE_WVARCHAR” “DBTYPE_WLONGVARCHAR” 和 DBCOLUMNFLAGS_ISLONG 或 DBPARAMFLAGS_ISLONG	“DBCLOB”
CHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_BINARY”	
VARCHAR(n) FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_VARBINARY”	
LONG VARCHAR FOR BIT DATA	DBTYPE_BYTES	“DBTYPE_LONGVARBINARY”	
BLOB	DBTYPE_BYTES 和 DBCOLUMNFLAGS_ISLONG 或 DBPARAMFLAGS_ISLONG	“DBTYPE_BINARY” “DBTYPE_VARBINARY” “DBTYPE_LONGVARBINARY” 和 DBCOLUMNFLAGS_ISLONG 或 DBPARAMFLAGS_ISLONG	“BLOB”

用于设置将数据从 OLE DB 类型转换为 DB2 类型的数据转换

IBM OLE DB Provider for DB2 支持用于设置将数据从 OLE DB 类型转换为 DB2 类型的数据转换。

从 OLE DB 类型到 DB2 类型的受支持的数据转换

下表列示从 OLE DB 类型到 DB2 类型的数据转换。注意，在某些情况下可能会发生数据截断，这取决于数据的类型和值。

表 15. 从 OLE DB 类型到 DB2 类型的数据转换

OLE DB 类型指示符	DB2 数据类型																					
																		对于位数据				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X	X	X	X	X				X	X											
DBTYPE_I2	X	X	X	X	X	X				X	X											
DBTYPE_I4	X	X	X	X	X	X				X	X											
DBTYPE_I8	X	X	X	X	X	X				X	X											
DBTYPE_UI1	X	X	X	X	X	X				X	X											
DBTYPE_UI2	X	X	X	X	X	X				X	X											
DBTYPE_UI4	X	X	X	X	X	X				X	X											
DBTYPE_UI8	X	X	X	X	X	X				X	X											
DBTYPE_R4	X	X	X	X	X	X				X	X											
DBTYPE_R8	X	X	X	X	X	X				X	X											
DBTYPE_CY																						
DBTYPE_DECIMAL	X	X	X	X	X	X				X	X											
DBTYPE_NUMERIC	X	X	X	X	X	X				X	X											

表 15. 从 OLE DB 类型到 DB2 类型的数据转换 (续)

OLE DB 类型指示符	DB2 数据类型																					
																		对于位数据				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DATE																						
DBTYPE_BOOL	X	X	X	X	X	X				X	X											
DBTYPE_BYTES			X			X				X	X	X				X		X	X	X		
DBTYPE_BSTR - 待确定																						
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR														X	X	X						
DBTYPE_VARIANT - 待确定																						
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN										X	X	X	X	X	X	X	X	X	X	X		
DBTYPE_GUID																						
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						

表 15. 从 OLE DB 类型到 DB2 类型的数据转换 (续)

OLE DB 类型指示符	DB2 数据类型																					
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	对于位数据			BLOB	DATALINK
																		CHAR	VARCHAR	LONG VARCHAR		
DBTYPE_DBDATE							X		X	X	X											
DBTYPE_DBTIME								X	X	X	X											
DBTYPE_DBTIMESTAMP							X	X	X	X	X											
DBTYPE_FILETIME																						
DBTYPE_PROP_VARIANT																						
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

用于将数据从 DB2 类型转换为 OLE DB 类型的数据转换

为了获取数据，IBM OLE DB Provider 允许将数据由 DB2 类型转换为 OLE DB 类型。

从 DB2 类型到 OLE DB 类型的受支持数据转换

下表列示从 DB2 类型到 OLE DB 类型的受支持数据转换。注意，在某些情况下可能会发生数据截断，这取决于数据的类型和值。

表 16. 从 DB2 类型到 OLE DB 类型的数据转换

	DB2 数据类型																					
																		对于位数据				
OLE DB 类型指示符	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_EMPTY																						
DBTYPE_NULL																						
DBTYPE_RESERVED																						
DBTYPE_I1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_I8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI1	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI2	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_UI8	X	X	X	X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R4	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_R8	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_CY	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_DECIMAL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_NUMERIC	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X

表 16. 从 DB2 类型到 OLE DB 类型的数据转换 (续)

OLE DB 类型指示符	DB2 数据类型																					
																		对于位数据				
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DATE	X	X		X	X		X	X	X	X	X	X		X	X	X						X
DBTYPE_BOOL	X	X		X	X	X				X	X	X		X	X	X		X	X	X		X
DBTYPE_BYTES	X	X		X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_BSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_STR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_WSTR	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_VARIANT	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_IDISPATCH																						
DBTYPE_IUNKNOWN	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
DBTYPE_GUID										X	X	X		X	X	X		X	X	X		X
DBTYPE_ERROR																						
DBTYPE_BYREF																						
DBTYPE_ARRAY																						
DBTYPE_VECTOR																						
DBTYPE_UDT																						
DBTYPE_DBDATE							X	X	X	X	X	X		X	X	X		X	X	X		X

表 16. 从 DB2 类型到 OLE DB 类型的数据转换 (续)

OLE DB 类型指示符	DB2 数据类型																				
																		对于位数据			
	SMALLINT	INTEGER	BIGINT	REAL	FLOAT DOUBLE	DECIMAL NUMERIC	DATE	TIME	TIMESTAMP	CHAR	VARCHAR	LONG VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	LONG VARGRAPHIC	DBCLOB	CHAR	VARCHAR	LONG VARCHAR	BLOB	DATALINK
DBTYPE_DBTIME							X	X	X	X	X	X		X	X	X						X
DBTYPE_DBTIMESTAMP							X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_FILETIME			X				X	X	X	X	X	X		X	X	X		X	X	X		X
DBTYPE_PROP_VARIANT	X	X	X	X	X					X	X	X		X	X	X		X	X	X		X
DBTYPE_HCHAPTER																						
DBTYPE_VARNUMERIC																						

注: 当应用程序执行 ISequentialStream::Read 以便从存储器对象中获取数据时，所返回的数据的格式取决于列数据类型:

- 对于非字符和非二进制数据类型而言，列的数据作为一系列字节返回，这些字节在操作系统中代表那些值。
- 对于字符数据类型而言，数据先转换为 DBTYPE_STR。
- 对于 DBCLOB 而言，数据先转换为 DBTYPE_WCHAR。

IBM OLE DB Provider 限制

IBM OLE DB Provider 的限制是:

- 使用 `ITransactionLocal` 接口，IBMDADB2 支持自动落实事务作用域和用户控制的事务作用域。自动落实事务作用域是缺省作用域。不支持嵌套事务。
- 当命令文本包含参数时，不支持 `RestartPosition`。
- IBMDADB2 不会将通过 DBID 参数（这些是 `IOpenRowset` 接口所使用的参数）传递的表名括在引号中。而是，OLE DB 使用者必须在需要引号时对表名添加引号。

IBM OLE DB Provider 对 OLE DB 组件和接口的支持

下列各表列示 IBM OLE DB Provider for DB2 和 Microsoft OLE DB Provider for ODBC 所支持的 OLE DB 组件和接口。

表 17. Blob

接口	DB2	ODBC Provider
ISequentialStream	是	是

表 18. 命令

接口	DB2	ODBC Provider
IAccessor	是	是
ICommand	是	是
ICommandPersist	否	否
ICommandPrepare	是	是
ICommandProperties	是	是
ICommandText	是	是
ICommandWithParameters	是	是
IColumnsInfo	是	是
IColumnsRowset	是	是
IConvertType	是	是
ISupportErrorInfo	是	是

表 19. 数据源

接口	DB2	ODBC Provider
IConnectionPoint	否	是
IDBAsynchNotify (consumer)	否	否
IDBAsynchStatus	否	否
IDBConnectionPointContainer	否	是
IDBCreateSession	是	是
IDBDataSourceAdmin	否	否
IDBInfo	是	是
IDBInitialize	是	是
IDBProperties	是	是
IPersist	是	否
IPersistFile	是	是
ISupportErrorInfo	是	是

表 20. 枚举符

接口	DB2	ODBC Provider
IDBInitialize	是	是
IDBProperties	是	是
IParseDisplayName	是	否

表 20. 枚举符 (续)

接口	DB2	ODBC Provider
ISourcesRowset	是	是
ISupportErrorInfo	是	是

表 21. 错误查找服务

接口	DB2	ODBC Provider
IErrorLookUp	是	是

表 22. 错误对象

接口	DB2	ODBC Provider
IErrorInfo	是	否
ISQLErrorInfo（定制）	是	否

表 23. 多个结果

接口	DB2	ODBC Provider
IMultipleResults	是	是
ISupportErrorInfo	是	是

表 24. 行集

接口	DB2	ODBC Provider
IAccessor	Yes	是
IColumnsRowset	Yes	是
IColumnsInfo	Yes	是
IConvertType	Yes	是
IChapteredRowset	否	否
IConnectionPointContainer	Yes	是
IDBAsynchStatus	否	否
IParentRowset	否	否
IRowset	Yes	是
IRowsetChange	Yes	是
IRowsetChapterMember	否	否
IRowsetFind	否	否
IRowsetIdentity	Yes	是
IRowsetIndex	否	否
IRowsetInfo	Yes	是
IRowsetLocate	Yes	是
IRowsetNotify（使用者）	Yes	否
IRowsetRefresh	游标服务组件	是
IRowsetResynch	游标服务组件	是
IRowsetScroll	是[1]	是
IRowsetUpdate	游标服务组件	是

表 24. 行集 (续)

接口	DB2	ODBC Provider
IRowsetView	否	否
ISupportErrorInfo	Yes	是
注: 1. 要返回的值是近似值。将不会跳过已删除的行。		

表 25. 会话

接口	DB2	ODBC Provider
IAlterIndex	否	否
IAlterTable	否	否
IDBCreateCommand	Yes	是
IDBSchemaRowset	Yes	是
IGetDataSource	Yes	是
IIndexDefinition	否	否
IOpenRowset	Yes	是
ISessionProperties	Yes	是
ISupportErrorInfo	Yes	是
ITableDefinition	否	否
ITableDefinitionWithConstraints	否	否
ITransaction	Yes	是
ITransactionJoin	Yes	是
ITransactionLocal	Yes	是
ITransactionObject	否	否
ITransactionOptions	否	是

表 26. 视图对象

接口	DB2	ODBC Provider
IViewChapter	否	否
IViewFilter	否	否
IViewRowset	否	否
IViewSort	否	否

IBM OLE DB Provider 对 OLE DB 属性的支持

下表列示 IBM OLE DB Provider for DB2 所支持的 OLE DB 属性:

表 27. IBM OLE DB Provider for DB2 所支持的属性: 数据源 (DBPROPSET_DATASOURCE)

属性	缺省值	R/W
DBPROP_MULTIPLECONNECTIONS	VARIANT_FALSE	R
DBPROP_RESETDATASOURCE	DBPROPVAL_RD_RESETALL	R/W

表 28. IBM OLE DB Provider for DB2 所支持的属性: DB2 数据源 (DBPROPSET_DB2DATASOURCE)

属性	缺省值	R/W
DB2PROP_REPORTISLONGFORLONGTYPES	VARIANT_FALSE	R/W
DB2PROP_RETURNCHARASWCHAR	VARIANT_TRUE	R/W
DB2PROP_SORTBYORDINAL	VARIANT_FALSE	R/W

表 29. IBM OLE DB Provider for DB2 所支持的属性: 数据源信息 (DBPROPSET_DATASOURCEINFO)

属性	缺省值	R/W
DBPROP_ACTIVESESSIONS	0	R
DBPROP_ASYNCTXNABORT	VARIANT_FALSE	R
DBPROP_ASYNCTXNCOMMIT	VARIANT_FALSE	R
DBPROP_BYREFACCESSORS	VARIANT_FALSE	R
DBPROP_COLUMNDEFINITION	DBPROPVAL_CD_NOTNULL	R
DBPROP_CONCATNULLBEHAVIOR	DBPROPVAL_CB_NULL	R
DBPROP_CONNECTIONSTATUS	DBPROPVAL_CS_INITIALIZED	R
DBPROP_DATASOURCENAME	不适用	R
DBPROP_DATASOURCEREADONLY	VARIANT_FALSE	R
DBPROP_DBMSNAME	不适用	R
DBPROP_DBMSVER	不适用	R
DBPROP_DSOTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
DBPROP_GROUPBY	DBPROPVAL_GB_CONTAINS_SELECT	R
DBPROP_IDENTIFIERCASE	DBPROPVAL_IC_UPPER	R
DBPROP_MAXINDEXSIZE	0	R
DBPROP_MAXROWSIZE	0	R
DBPROP_MAXROWSIZEINCLUDESBLOB	VARIANT_TRUE	R
DBPROP_MAXTABLEINSELECT	0	R
DBPROP_MULTIPLEPARAMSETS	VARIANT_FALSE	R
DBPROP_MULTIPLERESULTS	DBPROPVAL_MR_SUPPORTED	R
DBPROP_MULTIPLESTORAGEOBJECTS	VARIANT_TRUE	R
DBPROP_MULTITABLEUPDATE	VARIANT_FALSE	R
DBPROP_NULLCOLLATION	DBPROPVAL_NC_LOW	R
DBPROP_OLEOBJECTS	DBPROPVAL_OO_BLOB	R
DBPROP_ORDERBYCOLUMNSINSELECT	VARIANT_FALSE	R
DBPROP _OUTPUTPARAMETERAVAILABILITY	DBPROPVAL_OA_ATEXECUTE	R
DBPROP_PERSISTENTIDTYPE	DBPROPVAL_PT_NAME	R
DBPROP_PREPAREABORTBEHAVIOR	DBPROPVAL_CB_DELETE	R
DBPROP_PROCEDURETERM	″STORED PROCEDURE″	R
DBPROP_PROVIDERFRIENDLYNAME	″IBM OLE DB Provider for DB2″	R
DBPROP_PROVIDERNAME	″IBMDADB2.DLL″	R
DBPROP_PROVIDEROLEDBVER	″02.7″	R

表 29. IBM OLE DB Provider for DB2 所支持的属性: 数据源信息 (DBPROPSET_DATASOURCEINFO) (续)

属性	缺省值	R/W
DBPROP_PROVIDERVER	不适用	R
DBPROP_QUOTEIDENTIFIERCASE	DBPROPVAL_IC_SENSITIVE	R
DBPROP _ROWSETCONVERSIONSONCOMMAND	VARIANT_TRUE	R
DBPROP_SCHEMATERM	″SCHEMA″	R
DBPROP_SCHEMAUSAGE	DBPROPVAL_SU_DML_STATEMENTS | DBPROPVAL_SU_TABLE_DEFINITION | DBPROPVAL_SU_INDEX_DEFINITION | DBPROPVAL_SU_PRIVILEGE_DEFINITION	R
DBPROP_SQLSUPPORT	DBPROPVAL_SQL_ODBC_EXTENDED | DBPROPVAL_SQL_ESCAPECLAUSES | DBPROPVAL_SQL_ANSI92_ENTRY	R
DBPROP_SERVERNAME	不适用	R
DBPROP_STRUCTUREDSTORAGE	DBPROPVAL_SS_ISEQUENTIALSTREAM	R
DBPROP_SUBQUERIES	DBPROPVAL_SQ_CORRELATEDSUBQUERIES | DBPROPVAL_SQ_COMPARISON | DBPROPVAL_SQ_EXISTS | DBPROPVAL_SQ_IN | DBPROPVAL_SQ_QUANTIFIED |	R
DBPROP_SUPPORTEDTXNDDL	DBPROPVAL_TC_ALL	R
DBPROP_SUPPORTEDTXNISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY | DBPROPVAL_TI_READCOMMITTED | DBPROPVAL_TI_READUNCOMMITTED | DBPROPVAL_TI_SERIALIZABLE |	R
DBPROP_SUPPORTEDTXNISORETAIN	DBPROPVAL_TR_COMMIT_DC | DBPROPVAL_TR_ABORT_NO |	R
DBPROP_TABLETERM	″TABLE″	R
DBPROP_USERNAME	不适用	R

表 30. IBM OLE DB Provider for DB2 所支持的属性: 初始化 (DBPROPSET_DBINIT)

属性	缺省值	R/W
DBPROP_AUTH_PASSWORD	不适用	R/W
DBPROP_INIT_TIMEOUT (1)	0	R/W
DBPROP_AUTH_PERSIST _SENSITIVE_AUTHINFO	VARIANT_FALSE	R/W
DBPROP_AUTH_USERID	不适用	R/W
DBPROP_INIT_DATASOURCE	不适用	R/W
DBPROP_INIT_HWND	不适用	R/W

表 30. IBM OLE DB Provider for DB2 所支持的属性: 初始化 (DBPROPSET_DBINIT) (续)

属性	缺省值	R/W
DBPROP_INIT_MODE	DB_MODE_READWRITE	R/W
DBPROP_INIT_OLEDBSERVICES	0xFFFFFFFF	R/W
DBPROP_INIT_PROMPT	DBPROMPT_NOPROMPT	R/W
DBPROP_INIT_PROVIDERSTRING	不适用	R/W

表 31. IBM OLE DB Provider for DB2 所支持的属性: 行集 (DBPROPSET_ROWSET)

属性	缺省值	R/W
DBPROP_ABORTPRESERVE	VARIANT_FALSE	R
DBPROP_ACCESSORDER	DBPROPVAL_AO_RANDOM	R
DBPROP_BLOCKINGSTORAGEOBJECTS	VARIANT_FALSE	R
DBPROP_BOOKMARKS	VARIANT_FALSE	R/W
DBPROP_BOOKMARKSKIPPED	VARIANT_FALSE	R
DBPROP_BOOKMARKTYPE	DBPROPVAL_BMK_NUMERIC	R
DBPROP_CACHEDEFERRED	VARIANT_FALSE	R/W
DBPROP_CANFETCHBACKWARDS	VARIANT_FALSE	R/W
DBPROP_CANHOLDROWS	VARIANT_FALSE	R
DBPROP_CANSCROLLBACKWARDS	VARIANT_FALSE	R/W
DBPROP_CHANGEINSERTEDROWS	VARIANT_FALSE	R
DBPROP_COMMITPRESERVE	VARIANT_TRUE	R/W
DBPROP_COMMANDTIMEOUT	0	R/W
DBPROP_DEFERRED	VARIANT_FALSE	R
DBPROP_IAccessor	VARIANT_TRUE	R
DBPROP_IColumnsInfo	VARIANT_TRUE	R
DBPROP_IColumnsRowset	VARIANT_TRUE	R/W
DBPROP_IConvertType	VARIANT_TRUE	R
DBPROP_IMultipleResults	VARIANT_FALSE	R/W
DBPROP_IRowset	VARIANT_TRUE	R
DBPROP_IRowChange	VARIANT_FALSE	R/W
DBPROP_IRowsetFind	VARIANT_FALSE	R
DBPROP_IRowsetIdentity	VARIANT_TRUE	R
DBPROP_IRowsetInfo	VARIANT_TRUE	R
DBPROP_IRowsetLocate	VARIANT_FALSE	R/W
DBPROP_IRowsetScroll	VARIANT_FALSE	R/W
DBPROP_IRowsetUpdate	VARIANT_FALSE	R
DBPROP_ISequentialStream	VARIANT_TRUE	R
DBPROP_ISupportErrorInfo	VARIANT_TRUE	R
DBPROP_LITERALBOOKMARKS	VARIANT_FALSE	R
DBPROP_LITERALIDENTITY	VARIANT_TRUE	R
DBPROP_LOCKMODE	DBPROPVAL_LM_SINGLEROW	R/W
DBPROP_MAXOPENROWS	32767	R

表 31. IBM OLE DB Provider for DB2 所支持的属性: 行集 (DBPROPSET_ROWSET) (续)

属性	缺省值	R/W
DBPROP_MAXROWS	0	R/W
DBPROP_NOTIFICATIONGRANULARITY	DBPROPVAL_NT_SINGLEROW	R/W
DBPROP_NOTIFICATION PHASES	DBPROPVAL_NP_OKTODO DBPROPBAL_NP_ABOUTTODO DBPROPVAL_NP_SYNCHAFTER DBPROPVAL_NP_FAILEDTODO DBPROPVAL_NP_DIDEVENT	R
DBPROP_NOTIFYROWSETRELEASE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP _NOTIFYROWSETFETCHPOSITIONCHANGE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYCOLUMNSET	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYROWDELETE	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_NOTIFYROWINSERT	DBPROPVAL_NP_OKTODO DBPROPVAL_NP_ABOUTTODO	R
DBPROP_ORDEREDBOOKMARKS	VARIANT_FALSE	R
DBPROP_OTHERINSERT	VARIANT_FALSE	R
DBPROP_OTHERUPDATEDELETE	VARIANT_FALSE	R/W
DBPROP_OWNINSERT	VARIANT_FALSE	R
DBPROP_OWNUPDATEDELETE	VARIANT_FALSE	R
DBPROP_QUICKRESTART	VARIANT_FALSE	R/W
DBPROP_REMOVEDELETED	VARIANT_FALSE	R/W
DBPROP_ROWTHREADMODEL	DBPROPVAL_RT_FREETHREAD	R
DBPROP_SERVERCURSOR	VARIANT_TRUE	R
DBPROP_SERVERDATAONINSERT	VARIANT_FALSE	R
DBPROP_UNIQUEROWS	VARIANT_FALSE	R/W
DBPROP_UPDATABILITY	0	R/W

表 32. IBM OLE DB Provider for DB2 所支持的属性: DB2 行集 (DBPROPSET_DB2ROWSET)

属性	缺省值	R/W
DBPROP_ISLONGMINLENGTH	32000	R/W

表 33. IBM OLE DB Provider for DB2 所支持的属性: 会话 (DBPROPSET_SESSION)

属性	缺省值	R/W
DBPROP_SESS_AUTOCOMMITISOLEVELS	DBPROPVAL_TI_CURSORSTABILITY	R/W

注:

1. 仅当使用 TCP/IP 协议来连接到服务器时，超时才适用。只有在 TCP/IP 套接字连接期间，才会强制实施超时限制。如果套接字连接在指定的超时到期之前完成，那么将不会对余下的初始化过程强制实施超时限制。如果使用了客户机重新路由功能，那么超时将加倍。通常，启用客户机重新路由功能后，连接超时行为由客户机重新路由功能指定。

使用 IBM OLE DB Provider 来连接到数据源

下列示例说明如何使用 IBM OLE DB Provider for DB2 来连接到 DB2 数据源:

示例 1: 使用 ADO 的 Visual Basic 应用程序

```
Dim db As ADODB.Connection
Set db = New ADODB.Connection
db.Provider = "IBMDADB2"
db.CursorLocation = adUseClient
...
```

示例 2: 使用 IDataInitialize 和服务组件的 C/C++ 应用程序

```
hr = CoCreateInstance (
     CLSID_MSDAINITIALIZE,
     NULL,
     CLSCTX_INPROC_SERVER,
     IID_IDataInitialize,
     (void**)&pIDataInitialize);

hr = pIDataInitialize->CreateDBInstance(
     CLSID_IBMDADB2, // ClassID of IBMDADB2
     NULL,
     CLSCTX_INPROC_SERVER,
     NULL,
     IID_IDBInitialize,
     (IUnknown**)&pIDBInitialize);
```

ADO 应用程序

ADO 连接字符串关键字

要指定 ADO（ActiveX 数据对象）连接字符串关键字，请在提供程序（连接）字符串中使用 *关键字=值* 格式来指定关键字。要指定多个关键字，请使用分号 (;) 对各个关键字进行定界。

下表描述 IBM OLE DB Provider for DB2 所支持的关键字:

表 34. IBM OLE DB Provider for DB2 支持的关键字

关键字	值	含义
DSN	数据库别名	数据库目录中的 DB2 数据库别名。
UID	用户标识	用于连接到 DB2 服务器的用户标识。
PWD	UID 的密码	用于连接到 DB2 服务器的用户标识的密码。

其他 CLI 配置关键字也会影响 IBM OLE DB Provider 的行为。

使用 Visual Basic ADO 应用程序连接至数据源

要使用 IBM OLE DB Provider for DB2 连接至 DB2 数据源，请指定 IBMDADB2 提供程序名称。

ADO 应用程序中的可更新可滚动游标

IBM OLE DB Provider for DB2 提供了对只读、只前进、只读可滚动和可更新可滚动游标的本机支持。要访问可更新可滚动游标的 ADO 应用程序，可以将游标位置设置为 `adUseClient` 或 `adUseServer`。将游标位置设置为 `adUseServer` 将导致游标在服务器上具体化。

ADO 应用程序的限制

ADO 应用程序的限制是:

- 调用了存储过程的 ADO 应用程序必须创建并显式地绑定其参数。 DB2 Server for VSE & VM 不支持用于自动生成参数的 `Parameters.Refresh` 方法。
- 不支持缺省参数值。
- 使用服务器端可滚动游标来插入新行时，请使用带有 Fieldlist 和 Values 自变量的 AddNew() 方法。这比在对每个列执行 Update() 调用后调用不带自变量的 AddNew() 效率高。每个 AddNew() 和 Update() 调用都是对服务器发出的不同请求，因此，其效率不如单一的 AddNew() 调用。
- 新插入的行不可通过服务器端可滚动游标进行更新。
- 使用服务器端可滚动游标时，包含长整型数据或 LOB 数据的表不可更新。

IBM OLE DB Provider 对 ADO 方法和属性的支持

IBM OLE DB Provider 支持下列 ADO 方法和属性:

表 35. 命令方法

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
Cancel	ICommand	是
CreateParameter		是
Execute		是

表 36. 命令属性

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
ActiveConnection	（特定于 ADO）	
命令文本	ICommandText	是
命令超时	ICommandProperties::SetProperties DBPROP_COMMANDTIMEOUT	是
CommandType	（特定于 ADO）	
预编译	ICommandPrepare	是
状态	（特定于 ADO）	

表 37. 命令集合

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
参数	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	是
属性	ICommandProperties IDBProperties	是

表 38. 连接方法

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
BeginTrans CommitTrans RollbackTrans	ITransactionLocal	是（但不能嵌套） 是（但不能嵌套） 是（但不能嵌套）
Execute	ICommand IOpenRowset	是
Open	IDBCreateSession IDBInitialize	是
OpenSchema adSchemaColumnPrivileges adSchemaColumns adSchemaForeignKeys adSchemaIndexes adSchemaPrimaryKeys adSchemaProcedureParam adSchemaProcedures adSchemaProviderType adSchemaStatistics adSchemaTablePrivileges adSchemaTables	IDBSchemaRowset	是 是 是 是 是 是 是 是 是 是 是 是
Cancel		是

表 39. 连接属性

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
属性 adXactCommitRetaining adXactRollbackRetaining	ITransactionLocal	是 是
CommandTimeout	ICommandProperties DBPROP_COMMAND_TIMEOUT	是
ConnectionString	（特定于 ADO）	

表 39. 连接属性 (续)

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
ConnectionTimeout	IDBProperties DBPROP_INIT_TIMEOUT	否
CursorLocation: adUseClient adUseNone adUseServer	（使用 OLE DB 游标服务） （未使用）	是 否 是
DefaultDataBase	IDBProperties DBPROP_CURRENTCATALOG	否
IsolationLevel	ITransactionLocal DBPROP_SESS _AUTOCOMMITISOLEVELS	是
方式 adModeRead adModeReadWrite adModeShareDenyNone adModeShareDenyRead adModeShareDenyWrite adModeShareExclusive adModeUnknown adModeWrite	IDBProperties DBPROP_INIT_MODE	否 是 否 否 否 否 否 否
提供程序	ISourceRowset::GetSourceRowset	是
状态	（特定于 ADO）	
版本	（特定于 ADO）	

表 40. 连接集合

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
错误	IErrorRecords	是
属性	IDBProperties	是

表 41. 错误属性

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
描述 NativeError Number Source SQLState	IErrorRecords	是 是 是 是 是
HelpContext HelpFile		否 否

表 42. 字段方法

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
AppendChunk GetChunk	ISequentialStream	是 是

表 43. 字段属性

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
实际大小	IAccessor IRowset	是
属性 DataFormat DefinedSize Name NumericScale Precision Type	IColumnInfo	是 是 是 是 是 是
OriginalValue	IRowsetUpdate	是（游标服务）
UnderlyingValue	IRowsetRefresh IRowsetResynch	是 （游标服务） 是 （游标服务）
值	IAccessor IRowset	是

表 44. 字段集合

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
属性	IDBProperties IRowsetInfo	是

表 45. 参数方法

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
AppendChunk	ISequentialStream	是
属性 Direction Name NumericScale Precision Scale Size Type	ICommandWithParameter DBSCHEMA _PROCEDURE_PARAMETERS	是 否 是 是 是 是 是

表 45. 参数方法 (续)

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
值	IAccessor ICommand	是

表 46. Parameter 集合

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
属性		是

表 47. RecordSet 方法

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
AddNew	IRowsetChange	是
Cancel		是
CancelBatch	IRowsetUpdate::Undo	是（游标服务）
CancelUpdate		是（游标服务）
Clone	IRowsetLocate	是
Close	IAccessor IRowset	是
CompareBookmarks		否
Delete	IRowsetChange	是
GetRows	IAccessor IRowset	是
Move	IRowset IRowsetLocate	是
MoveFirst	IRowset IRowsetLocate	是
MoveNext	IRowset IRowsetLocate	是
MoveLast	IRowsetLocate	是
MovePrevious	IRowsetLocate	是
NextRecordSet	IMultipleResults	是
Open	ICommand IOpenRowset	是
Requery	ICommand IOpenRowset	是
Resync	IRowsetRefresh	是（游标服务）
Supports	IRowsetInfo	是

表 47. RecordSet 方法 (续)

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
更新 UpdateBatch	IRowsetChange IRowsetUpdate	是 是（游标服务）

表 48. RecordSet 属性

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
AbsolutePage	IRowsetLocate IRowsetScroll	是 是[1]
AbsolutePosition	IRowsetLocate IRowsetScroll	是 是[1]
ActiveConnection	IDBCreateSession IDBInitialize	是
BOF	（特定于 ADO）	
Bookmark	IAccessor IRowsetLocate	是
CacheSize	IRowsetLocate 中的 cRows IRowset	是
CursorType adOpenDynamic adOpenForwardOnly adOpenKeySet adOpenStatic	ICommandProperties	否 是 是 是
EditMode	IRowsetUpdate	是（游标服务）
EOF	（特定于 ADO）	
Filter	IRowsetLocate IRowsetView IRowsetUpdate IViewChapter IViewFilter	否
LockType	ICommandProperties	是
MarshallOption		否
MaxRecords	ICommandProperties IOpenRowset	是
PageCount	IRowsetScroll	是[1]
PageSize	（特定于 ADO）	
排序	（特定于 ADO）	
Source	（特定于 ADO）	
状态	（特定于 ADO）	

表 48. RecordSet 属性 (续)

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
状态	IRowsetUpdate	是（游标服务）
注: 1. 要返回的值是近似值。将不会跳过已删除的行。		

表 49. RecordSet 集合

方法/属性	OLE DB 接口/属性	IBM OLE DB 支持
字段	IColumnInfo	是
属性	IDBProperties IRowsetInfo::GetProperties	是

C/C++ 应用程序的编译和链接以及 IBM OLE DB Provider

使用了常量 CLSID_IBMDADB2 的 C/C++ 应用程序必须包括 SQLLIB\include 目录中的 ibmdadb2.h 文件。这些应用程序必须在 include 语句前定义 DBINITCONSTANTS。以下示例说明 C/C++ 预处理器伪指令的正确顺序:

```
#define DBINITCONSTANTS
#include "ibmdadb2.h"
```

在 C/C++ 应用程序中使用 IBM OLE DB Provider 来连接到数据源

要在 C/C++ 应用程序中使用 IBM OLE DB Provider for DB2 来连接到 DB2 数据源，您可以使用两个 OLE DB 核心接口（IDBPromptInitialize 或 IDataInitialize）中的一个，也可以调用 COM API CoCreateInstance。IDataInitialize 接口由 OLE DB 服务组件提供，而 IDBPromptInitialize 由数据链路组件提供。

COM+ 分布式事务支持和 IBM OLE DB Provider

在 Windows 2000 或 XP 上的 Microsoft Component Services (COM+) 环境中运行的 OLE DB 应用程序可以使用 ITransactionJoin 接口来加入与多个 DB2 Database for Linux, UNIX, and Windows、主机和 System i® 数据库服务器以及其他符合 COM+ 规范的资源管理器进行的分布式事务。

先决条件

要使用 IBM OLE DB Provider for DB2 所提供的 COM+ 分布式事务支持，请确保服务器符合下列先决条件。

注: 这些要求仅适用于那些基于 Windows 并安装了 DB2 客户机的计算机。

- 带有 Service Pack 3 或更高版本的 Windows 2000
- Windows XP

在 C/C++ 数据库应用程序中启用 COM+ 支持

要以 COM+ 事务方式运行 C 或 C++ 应用程序，您可以使用 CoCreateInstance 来创建 IBMDADB2 数据源实例，获取会话对象，然后使用 JoinTransaction。有关更多信息，请参阅有关如何将 C 或 C++ 应用程序连接到数据源的描述。

要以 COM+ 事务方式来运行 ADO 应用程序，请参阅有关如何将 C 或 C++ 应用程序连接到数据源的描述。

要以事务方式使用 COM+ 软件包中的组件，请将该组件的 Transactions 属性设置为下列其中一个值:

- 『Required』
- 『Required New』
- 『Supported』

有关这些值的信息，请参阅 COM+ 文档。

第 6 章 OLE DB .NET Data Provider

OLE DB .NET Data Provider 使用 IBM DB2 OLE DB Driver，在 ConnectionString 对象中，将后者作为 IBMDADB2 进行引用。

OLE DB .NET Data Provider 支持的连接字符串关键字与 IBM OLE DB Provider for DB2 支持的那些关键字相同。我们不再测试此提供程序。建议用户使用 IBM 数据服务器 .NET 提供程序。

并且，OLE DB .NET Data Provider 的限制也与 IBM DB2 OLE DB Provider 相同。OLE DB .NET Data Provider 还有其他限制，以下主题对这些限制作了阐述：《*开发 ADO.NET 和 OLE DB 应用程序*》中的『OLE DB .NET Data Provider 限制』。

要使用 OLE DB .NET Data Provider，必须安装 .NET Framework V2.0、V3.0 或 V3.5。

对于 DB2 通用数据库 AS/400 版 R520、R530 和 R540，服务器上需要以下修订：APAR ii13348。

OLE DB .NET Data Provider 的所有支持的连接关键字显示在表 1 中：

表 50. OLE DB .NET Data Provider 的有用 **ConnectionString** *关键字*

关键字	值	含义
PROVIDER	IBMDADB2	指定 IBM OLE DB Provider for DB2（必需）
DSN 或 **Data Source**	数据库别名	数据库目录中编目的 DB2 数据库别名
UID	用户标识	用于连接到 DB2 数据服务器的用户标识
PWD	密码	用于连接到 DB2 数据服务器的用户标识的密码

注：要获取 **ConnectionString** 关键字的完整列表，请参阅 Microsoft 文档。

创建 OleDbConnection 以连接到 SAMPLE 数据库的示例是：

```
[Visual Basic .NET]
Dim con As New OleDbConnection("Provider=IBMDADB2;" +
    "Data Source=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OleDbConnection con = new OleDbConnection("Provider=IBMDADB2;" +
                "Data Source=sample;UID=userid;PWD=password;" );
con.Open()
```

OLE DB .NET Data Provider 限制

我们已不再对 OLE DB .NET Data Provider 进行测试。建议用户使用 IBM 数据服务器 .NET 提供程序。

下表标识 OLE DB .NET Data Provider 的使用限制:

表 51. OLE DB .NET Data Provider 限制

类或功能	限制描述	受影响的 DB2 服务器
ASCII 字符流	在使用 DbType.AnsiString 或 DbType.AnsiStringFixedLength 时，不能将 ASCII 字符流与 OleDbParameters 配合使用。 OLE DB .NET Data Provider 将抛出以下异常: "Specified cast is not valid" **变通方法**: 使用 DbType.Binary 而不是 DbType.AnsiString 或 DbType.AnsiStringFixedLength。	所有
ADORecord	不支持 ADORecord。	所有
ADORecordSet 和 Timestamp	如 MSDN 所述，ADORecordSet 变体时间将解析为 1 秒。因此，将 DB2 Timestamp 列存储到 ADORecordSet 时，秒的所有小数部分都将丢失。同样，根据 ADORecordSet 填写 DataSet 之后，DataSet 中的 Timestamp 列将不会包含秒的任何小数部分。 **变通方法**: 此变通方法仅对 DB2 Universal Database for Linux, UNIX, and Windows V8.1 FP4 或更高版本生效。为避免分数秒的损失，可设置以下 CLI 关键字: MAPTIMESTAMPDESCRIBE = 2 此关键字会将 Timestamp 描述为 WCHAR(26)。要设置该关键字，请从 DB2 命令窗口执行以下命令: db2 update cli cfg for section common using MAPTIMESTAMPDESCRIBE 2	所有
Chapters	Chapters 不受支持。	所有
键信息	同时打开了 IDataReader 时，OLE DB .NET Data Provider 无法检索键信息。	DB2 for VM/VSE
来自存储过程的键信息	OLE DB .NET Data Provider 只能检索来自 DB2 Database for Linux, UNIX, and Windows 的存储过程所返回结果集的键信息。这是因为，用于除 Linux、UNIX 和 Windows 以外的平台的 DB2 服务器不会返回有关在存储过程中打开的结果集的扩展描述信息。 要检索 DB2 Database for Linux, UNIX, and Windows 上的存储过程所返回结果集的键信息，需要在 DB2 服务器上设置以下注册表变量: db2set DB2_APM_PERFORMANCE=8 设置这个服务器端 DB2 注册表变量将在服务器上长时间保留结果集元数据，从而使 OLE DB 能够成功地检索键信息。但是，根据服务器工作负载的不同，元数据可能不会被保留到 OLE DB Provider 查询信息之时。因此，无法保证始终能够获得存储过程所返回结果集的键信息。 要检索关于 CALL 语句的任何键信息，应用程序必须执行该 CALL 语句。调用 OleDbDataAdapter.FillSchema() 或 OleDbCommand.ExecuteReader (CommandBehavior.SchemaOnly | CommandBehavior.KeyInfo) 并不会实际地执行存储过程调用。因此，将不会检索到存储过程将要返回的结果集的任何键信息。	所有

表 51. OLE DB .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器

批处理 SQL 语句的键信息	使用返回多个结果的批处理 SQL 语句时，`FillSchema()` 方法将尝试只检索批处理 SQL 语句列表中第一个 SQL 语句的模式信息。如果此语句未返回结果集，那么将不会创建任何表。例如:
	`[C#]`
	`cmd.CommandText =`
	`"INSERT INTO ORG(C1) VALUES(1000); SELECT C1 FROM ORG;";`
	`da = new OleDbDataAdapter(cmd);`
	`da.FillSchema(ds, SchemaType.Source);`
	由于批处理 SQL 语句中的第一个语句是“INSERT”语句，此语句不返回结果集，因此将不会在数据集中创建任何表。
		所有

`OleDbCommandBuilder`	如果 SELECT 语句包含任何具有下列数据类型的列，那么 `OleDbCommandBuilder` 自动生成的 UPDATE、DELETE 和 INSERT 语句不正确: • CLOB • BLOB • DBCLOB • LONG VARCHAR • LONG VARCHAR FOR BIT DATA • LONG VARGRAPHIC 如果正在连接到除 DB2 Database for Linux, UNIX, and Windows 以外的 DB2 服务器，那么具有下列数据类型的列也将引起此问题: • VARCHAR[1] • VARCHAR FOR BIT DATA[1] • VARGRAPHIC[1] • REAL • FLOAT 或 DOUBLE • TIMESTAMP **注**: 1. 如果将具有这些数据类型的列定义为大于 254 字节的 VARCHAR 值、大于 254 字节的 VARCHAR 值 FOR BIT DATA 或者大于 127 字节的 VARGRAPHIC，那么这些列适用。仅当正在连接到除 DB2 Database for Linux, UNIX, and Windows 以外的 DB2 服务器时，此条件才有效。 `OleDbCommandBuilder` 将生成在 WHERE 子句中的相等比较中使用所有所选列的 SQL 语句，但不能在相等比较中使用先前列示的数据类型。 **注**: 注意，此限制将影响依靠 `OleDbCommandBuilder` 来自动生成 UPDATE、DELETE 和 INSERT 语句的 `IDbDataAdapter.Update()` 方法。如果生成的语句包含先前列示的任何一种数据类型，那么 UPDATE 操作将失败。 **变通方法**: 您需要显示除去属于先前通过已生成 SQL 语句的 WHERE 子句列示的数据类型的所有列。建议编写您自己的 UPDATE、DELETE 和 INSERT 语句。	所有
`OleDbCommandBuilder.DeriveParameters`	使用 `DeriveParameters()` 时，区分大小写至关重要。`OleDbCommand.CommandText` 所指定存储过程名的大小写必须与 DB2 系统目录表中存储的大小写匹配。要查看存储过程名的存储方式，请在不提供过程名限制的情况下调用 OpenSchema (OleDbSchemaGuid.Procedures)。这将返回所有存储过程名。缺省情况下，DB2 将存储过程名以大写方式进行存储，因此，您通常需要以大写方式指定存储过程名。	所有

表 51. OLE DB .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器
OleDbCommandBuilder.DeriveParameters	OleDbCommandBuilder.DeriveParameters() 方法在生成的 OleDbParameterCollection 中不包括 ReturnValue 参数。缺省情况下，SqlClient 和 IBM Data Server Provider for .NET 将该参数及 ParameterDirection.ReturnValue 添加到所生成的 ParameterCollection。	所有
OleDbCommandBuilder.DeriveParameters	对于重载的存储过程，OleDbCommandBuilder.DeriveParameters() 方法将失败。如果有多个名为“MYPROC”的存储过程，并且每个存储过程都接受不同数目的参数或不同类型的参数，那么 OleDbCommandBuilder.DeriveParameters() 将检索所有重载的存储过程的所有参数。	所有
OleDbCommandBuilder.DeriveParameters	如果应用程序未使用模式来限定存储过程，那么 DeriveParameters() 将返回该过程名的所有参数。因此，如果对于同一个过程名存在多个模式，那么 DeriveParameters() 将返回所有同名过程的所有参数。	所有
OleDbConnection.ChangeDatabase	不支持 OleDbConnection.ChangeDatabase() 方法。	所有
OleDbConnection.ConnectionString	在连接字符串中使用不可打印字符（例如“\b”、“\a”或“\O”）将导致抛出异常。 下列关键字具有限制: **Data Source** 数据源是数据库的名称，而不是服务器的名称。您可以指定 SERVER 服务器，但 IBMDADB2 提供程序将忽略此关键字。 **Initial Catalog 和 Connect Timeout** 不支持这些关键字。通常，OLE DB .NET Data Provider 将忽略所有无法识别的和不受支持的关键字。但是，指定这些关键字将引起以下异常: `Multiple-step OLE DB operation generated errors. Check each OLE DB status value, if available. No work was done.` **ConnectionTimeout** ConnectionTimeout 是只读的。	所有

OleDbConnection.GetOleDbSchemaTable	限制值区分大小写，并且必须与系统目录表中存储的数据库对象的大小写（缺省为大写）匹配。
	
	例如，如果您已经以此方式创建表:
	`CREATE TABLE abc(c1 SMALLINT)`
	
	DB2 会以大写方式将表名（“ABC”）存储在系统目录中。因此，您必须使用“ABC”作为限制值。例如:
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, "ABC", "TABLE" });`
	
	变通方法: 如果数据定义中需要区分大小写或空格，那么必须将它们括在引号中。例如:
	`cmd.CommandText = "create table \"Case Sensitive\"(c1 int)";`
	`cmd.ExecuteNonQuery();`
	`tablename = "\"Case Sensitive\"";`
	`schemaTable = con.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, tablename, "TABLE" });`
		所有

OleDbDataAdapter 和 DataColumnMapping	源列名区分大小写。它必须与 DB2 目录中存储的大小写（缺省为大写）匹配。 例如: `colMap = new DataColumnMapping("EMPNO", "Employee ID");`	所有

表 51. OLE DB .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器
`OleDbDataReader.GetSchemaTable`	OLE DB .NET Data Provider 无法从不返回扩展描述信息的服务器检索扩展描述信息。如果您正在连接到不支持扩展描述的服务器（受影响的服务器），那么 `IDataReader.GetSchemaTable()` 所返回的元数据表中的下列各列无效: • `IsReadOnly` • `IsUnique` • `IsAutoIncrement` • `BaseSchemaName` • `BaseCatalogName`	DB2 for OS/390® V7 或更低版本 DB2 for OS/400 DB2 for VM/VSE
存储过程: 没有结果集的列名	DB2 for OS/390 V6.1 服务器不返回从存储过程返回的结果集的列名。OLE DB .NET Data Provider 将把这些未命名的列映射到序号位置（例如“1”、“2”或“3”）。这与 MSDN 中阐述的映射不同: `"Column1"`、`"Column2"`或`"Column3"`。	DB2 for OS/390 V6.1

提示与技巧

OLE DB .NET Data Provider 应用程序中的连接合用

OLE DB .NET Data Provider 使用 OLE DB 会话合用自动合用连接。您可以使用连接字符串自变量来启用或禁用 OLE DB 服务（包括合用）。例如，以下连接字符串将禁用 OLE DB 会话合用和自动事务授权。

```
Provider=IBMDADB2;OLE DB Services=-4;Data Source=SAMPLE;
```

下表描述可以用于设置 OLE DB 服务的 ADO 连接字符串属性:

表 52. 使用 ADO 连接字符串属性来设置 OLE DB 服务

启用的服务	连接字符串中的值
所有服务（缺省）	″OLE DB Services = -1;″
除合用以外的所有服务	″OLE DB Services = -2;″
除合用和自动授权以外的所有服务	″OLE DB Services = -4;″
除客户机游标以外的所有服务	″OLE DB Services = -5;″
除客户机游标与合用以外的所有服务	″OLE DB Services = -6;″
无服务	″OLE DB Services = 0;″

有关 OLE DB 会话合用或资源合用以及如何通过覆盖 OLE DB 提供程序服务缺省值来禁用合用的更多信息，请参阅 MSDN 资料库中的 OLE DB Programmer’s Reference，网址为:

```
http://msdn.microsoft.com/library
```

OLE DB .NET Data Provider 应用程序中的时间列

下列各节描述如何在 OLE DB .NET Data Provider 应用程序中实现时间列。

使用参数标记进行插入

您想要将时间值插入到时间列中:

```
command.CommandText = "insert into mytable(c1) values( ? )";
```

其中，列 c1 是时间列。下面是两种将时间值与参数标记绑定的方法:

使用 OleDbParameter.OleDbType = OleDbType.DBTime

由于 OleDbType.DBTime 映射到 TimeSpan 对象，因此您必须提供 TimeSpan 对象作为参数值。此参数值不能是 String 或 DateTime 对象，它必须是 TimeSpan 对象。例如:

```
p1.OleDbType = OleDbType.DBTime;
p1.Value = TimeSpan.Parse("0.11:20:30");
rowsAffected = cmd.ExecuteNonQuery();
```

TimeSpan 的格式以格式为“[-]d.hh:mm:ss.ff”的字符串表示，MSDN 文档对此作了阐述。

使用 OleDbParameter.OleDbType = OleDbType.DateTime

这将强制 OLE DB .NET Data Provider 将参数值转换为 DateTime 对象（而不是 TimeSpan 对象），因此，参数值可以是任何可以转换为 DateTime 对象的有效字符串/对象。这意味着，“11:20:30”之类的值有效。值还可以是 DateTime 对象。由于 TimeSpan 对象无法转换为 DateTime 对象，因此，值不能是 TimeSpan 对象 – TimeSpan 未实现 IConvertible。

例如:

```
p1.OleDbType = OleDbType.DBTimeStamp;
p1.Value = "11:20:30";
rowsAffected = cmd.ExecuteNonQuery();
```

检索

要检索时间列，您需要使用 IDataRecord.GetValue() 方法或 OleDbDataReader.GetTimeSpan() 方法。

例如:

```
TimeSpan ts1 = ((OleDbDataReader)reader).GetTimeSpan( 0 );
TimeSpan ts2 = (TimeSpan) reader.GetValue( 0 );
```

OLE DB .NET Data Provider 应用程序中的 ADORecordset 对象

有关使用 ADORecordset 对象的注意事项。

- ADO 类型 adDBTime 类映射到 .NET Framework DateTime 类。OleDbType.DBTime 与 TimeSpan 对象相对应。
- 不能将 TimeSpan 对象赋予 ADORecordset 对象的 Time 字段。这是因为，ADORecordset 对象的 Time 字段应该包含 DateTime 对象。将 TimeSpan 对象赋予 ADORecordset 对象时，您将接收到以下消息:

  ```
  Method's type signature is not Interop compatible.
  ```

  在 Time 字段中，只能填充 DateTime 对象或者可以解析为 DateTime 对象的 String。
- 在 DataSet 中填充使用 OleDbDataAdapter 的 ADORecordset 时，ADORecordset 中的 Time 字段将转换为 DataSet 中的 TimeSpan 列。
- Recordsets 不存储主键或约束。因此，根据使用 MissingSchemaAction.AddWithKey 的 Recordset 来填充 DataSet 时，不会添加任何键信息。

第 7 章 ODBC .NET Data Provider

ODBC .NET Data Provider 使用 CLI 驱动程序对 DB2 数据源进行 ODBC 调用。因此，ODBC .NET Data Provider 支持的连接字符串关键字与 CLI 驱动程序支持的连接字符串关键字相同。我们不再测试此提供程序。建议用户使用 IBM 数据服务器 .NET 提供程序。

而且，ODBC .NET Data Provider 的限制与 CLI 驱动程序的限制相同。ODBC .NET Data Provider 还有其他限制，以下主题对这些限制作了阐述：《*开发 ADO.NET 和 OLE DB 应用程序*》中的『ODBC .NET Data Provider 限制』。

要使用 ODBC .NET Data Provider，必须安装 .NET Framework V2.0、V3.0 或 V3.5。对于 DB2 通用数据库 AS/400 版 V5R4 及更低版本，服务器上需要以下修订：APAR II13348。

表 1 中列示了 ODBC .NET Data Provider 的受支持的连接关键字：

*表 53. ODBC .NET Data Provider 的有用 **ConnectionString** 关键字*

关键字	值	含义
DSN	数据库别名	数据库目录中编目的 DB2 数据库别名
UID	用户标识	用于连接到 DB2 服务器的用户标识
PWD	密码	用于连接到 DB2 服务器的用户标识的密码

注：要获取 **ConnectionString** 关键字的完整列表，请参阅 Microsoft 文档。

以下代码是创建 OdbcConnection 以连接到 SAMPLE 数据库的示例：

```
[Visual Basic .NET]
Dim con As New OdbcConnection("DSN=sample;UID=userid;PWD=password;")
con.Open()

[C#]
OdbcConnection con = new OdbcConnection("DSN=sample;UID=userid;PWD=password;");
con.Open()
```

ODBC .NET Data Provider 限制

我们已不再对 ODBC .NET Data Provider 进行测试。建议用户使用 IBM 数据服务器 .NET 提供程序。

下表标识 ODBC .NET Data Provider 的使用限制:

表 54. ODBC .NET Data Provider 限制

类或功能	限制描述	受影响的 DB2 服务器
ASCII 字符流	在使用 DbType.AnsiString 或 DbType.AnsiStringFixedLength 时，不能将 ASCII 字符流与 OdbcParameters 配合使用。 ODBC .NET Data Provider 将抛出以下异常: "Specified cast is not valid" **变通方法**: 使用 DbType.Binary 而不是 DbType.AnsiString 或 DbType.AnsiStringFixedLength。	所有

Command.Prepare	如果 CommandText 自从上次准备后已更改，那么在执行命令（Command.ExecuteNonQuery 或 Command.ExecuteReader）之前，必须显式地运行 OdbcCommand.Prepare()。如果未再次调用 OdbcCommand.Prepare()，那么 ODBC .NET Data Provider 将执行先前准备的 CommandText。
	
	例如:
	[C#]
	command.CommandText="select CLOB('ABC') from table1";
	command.Prepare();
	command.ExecuteReader();
	command.CommandText="select CLOB('XYZ') from table2";
	
	// This ends up re-executing the first statement
	command.ExecuteReader();
		所有

表 54. ODBC .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器

CommandBehavior.
SequentialAccess
	使用 IDataReader.GetChars() 从通过 CommandBehavior.SequentialAccess 创建的阅读器读取内容时，必须分配大小足以存放整个列的缓冲区。否则，将发生以下异常:
	`Requested range extends past the end of the array.`
	`  at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`  Char[] destination, Int32 startIndex, Int32 length)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(Int32 i,`
	`  Int64 dataIndex, Char[] buffer, Int32 bufferIndex, Int32 length)`
	`  at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
	以下示例演示如何分配足够大的缓冲区:
	`CREATE TABLE myTable(c0 int, c1 CLOB(10K))`
	`SELECT c1 FROM myTable;`
	`[C#]`
	`cmd.CommandText = "SELECT c1 from myTable";`
	`IDataReader reader =`
	`cmd.ExecuteReader(CommandBehavior.SequentialAccess);`
	`Int32 iChunkSize = 10;`
	`Int32 iBufferSize = 10;`
	`Int32 iFieldOffset = 0;`
	`Char[] buffer = new Char[ iBufferSize ];`
	`reader.Read();`
	`reader.GetChars(0, iFieldOffset, buffer, 0, iChunkSize);`
	对 GetChars() 的调用将抛出以下异常:
	`"Requested range extends past the end of the array"`
	要确保 GetChars() 不抛出上面提到的异常，必须将 BufferSize 设置为列的大小，如下所示:
	`Int32 iBufferSize = 10000;`
	注意，iBufferSize 的值 10,000 与分配给 CLOB 列 c1 的值 10K 相对应。
		所有

CommandBehavior.
SequentialAccess
	使用 OdbcDataReader.GetChars() 时，如果没有更多可读取的数据，那么 ODBC .NET Data Provider 将抛出以下异常:
	`NO_DATA - no error information available`
	`  at System.Data.Odbc.OdbcConnection.HandleError(`
	`  HandleRef hrHandle, SQL_HANDLE hType, RETCODE retcode)`
	`  at System.Data.Odbc.OdbcDataReader.GetData(`
	`  Int32 i, SQL_C sqlctype, Int32 cb)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(`
	`  Int32 i, Int64 dataIndex, Char[] buffer,`
	`  Int32 bufferIndex, Int32 length)`
		所有

CommandBehavior.
SequentialAccess
	使用 OdbcDataReader.GetChars() 时，可能无法使用较大的块大小，例如值 5000。当您尝试使用较大的块大小时，ODBC .NET Data Provider 将抛出以下异常:
	`Object reference not set to an instance of an object.`
	`  at System.Runtime.InteropServices.Marshal.Copy(Int32 source,`
	`  Char[] destination, Int32 startIndex, Int32 length)`
	`  at System.Data.Odbc.OdbcDataReader.GetChars(`
	`  Int32 i, Int64 dataIndex, Char[] buffer,`
	`  Int32 bufferIndex, Int32 length)`
	`  at OleRestrict.TestGetCharsAndBufferSize(IDbConnection con)`
		所有

连接合用	ODBC .NET Data Provider 不控制连接合用。连接合用由 ODBC 驱动程序管理器处理。有关连接合用的更多信息，请参阅 MSDN 库中的 ODBC Programmer's Reference，网址为: `http://msdn.microsoft.com/library`	所有

表 54. ODBC .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器
DataColumnMapping	源列名的大小写必须与系统目录表中使用的大小写（缺省为大写）匹配。	所有

Decimal 列	Decimal 列不支持参数标记。
	
	通常，如果目标 SQLType 是 Decimal 列，那么您将 OdbcType.Decimal 用于 OdbcParameter；但是，当 ODBC .NET Data Provider 检测到 OdbcType.Decimal 时，它使用 C 类型 SQL_C_WCHAR 以及 SQLType 值 SQL_VARCHAR 来绑定参数，这是无效的。
	
	例如:
	
	[C#]
	cmd.CommandText = "SELECT dec_col FROM MYTABLE WHERE dec_col > ? ";
	OdbcParameter p1 = cmd.CreateParameter();
	p1.DbType = DbType.Decimal;
	p1.Value = 10.0;
	cmd.Parameters.Add(p1);
	IDataReader rdr = cmd.ExecuteReader();
	
	您将接收到以下异常:
	
	ERROR [07006] [IBM][CLI Driver][SQLDS/VM]
	SQL0301N The value of input host variable or parameter
	number "" cannot be used because of its data type.
	SQLSTATE=07006
	
	解决方法: 请完全使用字面值而不是 OdbcParameter 值。
		DB2 for VM/VSE

键信息	用于限定表名的模式名（例如 MYSCHEMA.MYTABLE）必须与连接用户标识匹配。ODBC .NET Data Provider 无法检索任何其指定模式与连接用户标识不同的键信息。
	
	例如:
	
	CREATE TABLE USERID2.TABLE1(c1 INT NOT NULL PRIMARY KEY);
	
	[C#]
	// Connect as user bob
	odbcCon = new OdbcConnection("DSN=sample;UID=bob;PWD=mypassword");
	
	OdbcCommand cmd = odbcCon.CreateCommand();
	
	// Select from table with schema USERID2
	cmd.CommandText="SELECT * FROM USERID2.TABLE1";
	
	// Fails - No key info retrieved
	da.FillSchema(ds, SchemaType.Source);
	
	// Fails - SchemaTable has no primary key
	cmd.ExecuteReader(CommandBehavior.KeyInfo)
	
	// Throws exception because no primary key
	cbuilder.GetUpdateCommand();
		所有

表 54. ODBC .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器

键信息	同时打开了 IDataReader 时，ODBC .NET Data Provider 无法检索键信息。当 ODBC .NET Data Provider 打开 IDataReader 时，将在服务器上打开游标。如果已请求获取键信息，那么它接着将调用 SQLPrimaryKeys() 或 SQLStatistic() 以获取键信息，但这些模式函数不会打开另一个游标。因为 DB2 for VM/VSE 不支持挂起游标，所以接着将关闭第一个游标。因此，IDataReader.Read() 调用 IDataReader 将引起以下异常:
	`System.Data.Odbc.OdbcException: ERROR [HY010] [IBM][CLI Driver]`
	`  CLI0125E  Function sequence error. SQLSTATE=HY010`
	解决方法: 您需要先检索键信息，然后检索数据。例如:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	`// Use FillSchema to retrieve just the schema information`
	`da.FillSchema(ds, SchemaType.Source);`
	`// Use FillSchema to retrieve just the schema information`
	`da.Fill(ds);`
		DB2 for VM/VSE

键信息	在 SQL 语句中引用数据库对象时，使用的大小写必须与系统目录表中存储的数据库对象的大小写相同。缺省情况下，数据库对象以大写方式存储在系统目录表中，所以，您通常需要使用大写。 ODBC .NET Data Provider 将扫描 SQL 语句以检索数据库对象名并将其传递到模式函数（例如 SQLPrimaryKeys 和 SQLStatistics），这些模式函数将对系统目录表中的这些对象发出查询。数据库对象引用必须与这些对象在系统目录表中的存储方式完全匹配，否则，将返回空结果集。	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE
批处理非 SELECT SQL 语句的键信息	ODBC .NET Data Provider 无法检索未以“SELECT”开头的批处理语句的任何键信息。	DB2 for OS/390 DB2 for OS/400 DB2 for VM/VSE

表 54. ODBC .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器

LOB 列	ODBC .NET Data Provider 不支持 LOB 数据类型。因此，每当 DB2 服务器返回 SQL_CLOB (-99)、SQL_BLOB (-98) 或 SQL_DBCLOB (-350) 时，ODBC .NET Data Provider 将抛出以下异常:
	`"Unknown SQL type - -98"  (for Blob column)`
	`"Unknown SQL type - -99"  (for Clob column)`
	`"Unknown SQL type - -350"  (for DbClob column)`
	任何直接或间接访问 LOB 列的方法都将失败。
	解决方法: 将 CLI/ODBC LongDataCompat 关键字设置为 1。这样做会强制 CLI 驱动程序将以下数据类型映射至 ODBC .NET Data Provider 可识别的数据类型:
	• SQL_CLOB 至 SQL_LONGVARCHAR
	• SQL_BLOB 至 SQL_LONGVARBINARY
	• SQL_DBCLOB 至 SQL_WLONGVARCHAR
	要设置 LongDataCompat 关键字，请从客户机上的 DB2 命令窗口运行以下 DB2 命令:
	`db2 update cli cfg for section common using longdatacompat 1`
	通过按如下所示使用连接字符串，还可在应用程序设置此关键字:
	`[C#]`
	`OdbcConnection con =`
	`  new OdbcConnection("DSN=SAMPLE;UID=uid;PWD=mypwd;LONGDATACOMPAT=1;");`
	有关所有 CLI/ODBC 关键字的列表，请参阅“UID CLI/ODBC 配置关键字”（在《DB2 CLI 指南和参考》中。
		所有

OdbcCommand.Cancel	在运行 OdbcCommand.Cancel 之后执行语句可能会引起以下异常:
	`"ERROR [24000] [Microsoft][ODBC Driver Manager]`
	`  Invalid cursor state"`
		所有

OdbcCommandBuilder	对于不支持转义字符的服务器，OdbcCommandBuilder 无法生成命令。当 OdbcCommandBuilder 生成命令时，它首先调用 SQLGetInfo，从而请求获取 SQL_SEARCH_PATTERN_ESCAPE 属性。如果服务器不支持转义字符，那么将返回空字符串，这将导致 ODBC .NET Data Provider 抛出以下异常:
	`Index was outside the bounds of the array.`
	`  at System.Data.Odbc.OdbcConnection.get_EscapeChar()`
	`  at System.Data.Odbc.OdbcDataReader.GetTableNameFromCommandText()`
	`  at System.Data.Odbc.OdbcDataReader.BuildMetaDataInfo()`
	`  at System.Data.Odbc.OdbcDataReader.GetSchemaTable()`
	`  at System.Data.Common.CommandBuilder.BuildCache(`
	`  Boolean closeConnection)`
	`  at System.Data.Odbc.OdbcCommandBuilder.GetUpdateCommand()`
		DB2 for OS/390，仅适用于 DBCS 服务器；DB2 for VM/VSE，仅适用于 DBCS 服务器

表 54. ODBC .NET Data Provider 限制 (续)

类或功能	限制描述	受影响的 DB2 服务器

OdbcCommandBuilder	使用 OdbcCommandBuilder 来自动生成 UPDATE、DELETE 和 INSERT 语句时，区分大小写至关重要。缺省情况下，除非您在创建期间通过在数据库对象两旁添加引号以区分大小写方式显式创建这些对象，否则 DB2 以大写方式将模式信息（例如表名和列名）存储在系统目录表中。因此，SQL 语句必须与目录中存储的大小写（缺省为大写）匹配。
	
	例如，如果已使用以下语句来创建表:
	`"db2 create table mytable (c1 int) "`
	
	那么 DB2 将把表名“mytable”作为“MYTABLE”存储在系统目录表中。
	
	以下代码示例演示如何正确使用 OdbcCommandBuilder 类:
	`[C#]`
	`OdbcCommand cmd = odbcCon.CreateCommand();`
	`cmd.CommandText  = "SELECT * FROM MYTABLE";`
	`OdbcDataAdapter da = new OdbcDataAdapter(cmd);`
	`OdbcCommandBuilder cb = new OdbcCommandBuilder(da);`
	`OdbcCommand updateCmd = cb.GetUpdateCommand();`
	
	在此示例中，如果未使用大写字符来引用表名，那么您将接收到以下异常:
	`"Dynamic SQL generation for the UpdateCommand is not supported against a SelectCommand that does not return any key column information."`
		所有

OdbcCommandBuilder	当 SELECT 语句包含下面这些列数据类型时，OdbcCommandBuilder 生成的命令不正确:
	`REAL`
	`FLOAT or DOUBLE`
	`TIMESTAMP`
	
	在 SELECT 语句的 WHERE 子句中，不能使用这些数据类型。
		DB2 for OS/390
		DB2 for OS/400
		DB2 for VM/VSE

OdbcCommandBuilder.DeriveParameters	DeriveParameters() 方法映射到 SQLProcedureColumns，它使用 CommandText 属性作为存储过程的名称。由于 CommandText 未包含存储过程的名称（使用完整 ODBC 调用语法），因此将在根据 ODBC 调用语法标识过程名的情况下调用 SQLProcedureColumns。例如: `"{ CALL myProc(?) }"` 这将生成一个空的结果集，在此结果集中，找不到过程的列。	所有
OdbcCommandBuilder.DeriveParameters	要使用 DeriveParameters()，请在 CommandText 中指定存储过程名（例如 cmd.CommandText = "MYPROC"）。过程名的大小写必须与系统目录表中存储的大小写匹配。DeriveParameters() 将返回它在系统目录表中找到的过程名的所有参数。在执行语句前，请不要忘记将 CommandText 重新更改为完整的 ODBC 调用语法。	所有
OdbcCommandBuilder.DeriveParameters	对于 ODBC .NET Data Provider 而言，不会返回 ReturnValue 参数。	所有
OdbcCommandBuilder.DeriveParameters	DeriveParameters() 不支持标准存储过程名。例如，为 CommandText = "MYSCHEMA.MYPROC" 调用 DeriveParameters() 将失败。这里，不会返回任何参数。	所有
OdbcCommandBuilder.DeriveParameters	对于重载的存储过程，DeriveParameters() 无效。SQLProcedureColumns 将返回存储过程的所有版本的所有参数。	所有
OdbcConnection.ChangeDatabase	不支持 OdbcConnection.ChangeDatabase() 方法。	所有

表 54. *ODBC .NET Data Provider 限制 (续)*

类或功能	限制描述	受影响的 DB2 服务器
OdbcConnection. ConnectionString	• Server 关键字将被忽略。 • Connect Timeout 关键字将被忽略。CLI 不支持连接超时，所以设置此属性不会影响该驱动程序。 • 连接合用关键字将被忽略。具体而言，这将影响下列关键字: Pooling、Min Pool Size、Max Pool Size、Connection Lifetime 和 Connection Reset。	所有
OdbcDataReader. GetSchemaTable	ODBC .NET Data Provider 无法从不返回扩展描述信息的服务器检索扩展描述信息。因此，如果您正在连接到不支持扩展描述的服务器（受影响的服务器），那么 IDataReader.GetSchemaTable() 所返回的元数据表中的下列各列无效: • IsReadOnly • IsUnique • IsAutoIncrement • BaseSchemaName • BaseCatalogName	DB2 for OS/390 V7 或更低版本 DB2 for OS/400 DB2 for VM/VSE

存储过程	要调用存储过程，需要指定完整的 ODBC 调用语法。
	
	例如，要调用接受 VARCHAR(10) 作为参数的存储过程 MYPROC:
	[C#]
	OdbcCommand cmd = odbcCon.CreateCommand();
	cmd.CommandType = CommandType.Text;
	cmd.CommandText = "{ CALL MYPROC(?) }"
	OdbcParameter p1 = cmd.CreateParameter();
	p1.Value = "Joe";
	p1.OdbcType = OdbcType.NVarChar;
	cmd.Parameters.Add(p1);
	cmd.ExecuteNonQuery();
	
	注: 注意，即使您正在使用 CommandType.StoredProcedure，也必须使用完整的 ODBC 调用语法。 在 MSDN 中，有关 OdbcCommand.CommandText 属性的内容对此语法作了阐述。
		所有

存储过程: 没有结果集的列名	DB2 for OS/390 V6.1 服务器不返回从存储过程返回的结果集的列名。ODBC .NET Data Provider 将这些未命名列映射至其序号位置（例如，“1”、“2”和“3”）。这与 MSDN 中阐述的映射不同: "Column1"、"Column2"或"Column3"。	DB2 for OS/390 V6.1
唯一索引提升为主键	ODBC .NET Data Provider 将可空唯一索引提升为主键。这与 MSDN 文档中的陈述不同，该文档指出，不应将可空唯一索引提升为主键。	所有

附录 A. DB2 技术信息概述

DB2 技术信息以多种可以通过多种方法访问的格式提供。

您可以通过下列工具和方法获得 DB2 技术信息:

- DB2 信息中心
  - 主题（任务、概念和参考主题）
  - 样本程序
  - 教程
- DB2 书籍
  - PDF 文件（可下载）
  - PDF 文件（在 DB2 PDF DVD 中）
  - 印刷版书籍
- 命令行帮助
  - 命令帮助
  - 消息帮助

注: DB2 信息中心主题的更新频率比 PDF 书籍或硬拷贝书籍的更新频率高。要获取最新信息，请安装可用的文档更新或者参阅 ibm.com 上的 DB2 信息中心。

您可以在线访问 ibm.com 上的其他 DB2 技术信息，例如技术说明、白皮书和 IBM Redbooks® 出版物。请访问以下网址处的 DB2 信息管理软件资料库站点: http://www.ibm.com/software/data/sw-library/。

文档反馈

我们非常重视您对 DB2 文档的反馈。如果您想就如何改善 DB2 文档提出建议，请向 db2docs@ca.ibm.com 发送电子邮件。DB2 文档小组将阅读您的所有反馈，但无法直接给您答复。请尽可能提供具体的示例，这样我们才能更好地了解您所关心的问题。如果您要提供有关具体主题或帮助文件的反馈，请加上标题和 URL。

请不要使用以上电子邮件地址与 DB2 客户支持机构联系。如果您遇到文档无法解决的 DB2 技术问题，请与您当地的 IBM 服务中心联系以获得帮助。

硬拷贝或 PDF 格式的 DB2 技术库

下列各表描述 IBM 出版物中心（网址为 www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss）所提供的 DB2 资料库。可从 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 下载 PDF 格式的 DB2 V10.1 手册的英文版本和翻译版本。

尽管这些表标识书籍有印刷版，但可能未在您所在国家或地区提供。

每次更新手册时，表单号都会递增。确保您正在阅读下面列示的手册的最新版本。

注: *DB2 信息中心*的更新频率比 PDF 或硬拷贝书籍的更新频率高。

表 55. DB2 技术信息

书名	书号	是否提供印刷版	最近一次更新时间
Administrative API Reference	SC27-3864-00	是	2012 年 4 月
Administrative Routines and Views	SC27-3865-00	否	2012 年 4 月
Call Level Interface Guide and Reference Volume 1	SC27-3866-00	是	2012 年 4 月
Call Level Interface Guide and Reference Volume 2	SC27-3867-00	是	2012 年 4 月
Command Reference	SC27-3868-00	是	2012 年 4 月
数据库管理概念和配置参考	S151-1758-00	是	2012 年 4 月
Data Movement Utilities Guide and Reference	S151-1756-00	是	2012 年 4 月
数据库监视指南和参考	S151-1759-00	是	2012 年 4 月
数据恢复及高可用性指南与参考	S151-1755-00	是	2012 年 4 月
数据库安全性指南	S151-1753-01	是	2012 年 4 月
DB2 Workload Management Guide and Reference	SC27-3891-00	是	2012 年 4 月
开发 ADO.NET 和 OLE DB 应用程序	S151-1765-00	是	2012 年 4 月
开发嵌入式 SQL 应用程序	S151-1763-00	是	2012 年 4 月
Developing Java Applications	SC27-3875-00	是	2012 年 4 月
Developing Perl, PHP, Python, and Ruby on Rails Applications	SC27-3876-00	否	2012 年 4 月
开发用户定义的例程（SQL 和外部例程）	S151-1761-00	是	2012 年 4 月
数据库应用程序开发入门	G151-1764-00	是	2012 年 4 月
Linux 和 Windows 上的 DB2 安装和管理入门	G151-1769-00	是	2012 年 4 月
全球化指南	S151-1757-00	是	2012 年 4 月
安装 DB2 服务器	G151-1768-00	是	2012 年 4 月
安装 IBM Data Server Client	G151-1751-00	否	2012 年 4 月
消息参考第 1 卷	S151-1767-00	否	2012 年 4 月
消息参考第 2 卷	S151-1766-00	否	2012 年 4 月
Net Search Extender 管理和用户指南	S151-1078-00	否	2012 年 4 月

表 55. DB2 技术信息 (续)

书名	书号	是否提供印刷版	最近一次更新时间
分区和集群指南	S151-1754-00	是	2012 年 4 月
pureXML 指南	S151-1775-00	是	2012 年 4 月
Spatial Extender User's Guide and Reference	SC27-3894-00	否	2012 年 4 月
《SQL 过程语言: 应用程序启用和支持》	S151-1762-00	是	2012 年 4 月
SQL Reference Volume 1	SC27-3885-00	是	2012 年 4 月
SQL Reference Volume 2	SC27-3886-00	是	2012 年 4 月
Text Search Guide	SC27-3888-00	是	2012 年 4 月
故障诊断和调整数据库性能	S151-1760-00	是	2012 年 4 月
升级到 DB2 V10.1	S151-1770-00	是	2012 年 4 月
DB2 V10.1 新增内容	S151-1752-00	是	2012 年 4 月
XQuery 参考	S151-1774-00	否	2012 年 4 月

表 56. 特定于 DB2 Connect 的技术信息

书名	书号	是否提供印刷版	最近一次更新时间
DB2 Connect 安装和配置 DB2 Connect Personal Edition	S151-1773-00	是	2012 年 4 月
DB2 Connect 安装和配置 DB2 Connect 服务器	S151-1772-00	是	2012 年 4 月
DB2 Connect 用户指南	S151-1771-00	是	2012 年 4 月

从命令行处理器显示 SQL 状态帮助

DB2 产品针对可能充当 SQL 语句结果的条件返回 SQLSTATE 值。SQLSTATE 帮助说明 SQL 状态和 SQL 状态类代码的含义。

过程

要启动 SQL 状态帮助，请打开命令行处理器并输入:

```
? sqlstate or ? class code
```

其中，*sqlstate* 表示有效的 5 位 SQL 状态，*class code* 表示该 SQL 状态的前 2 位。例如，`? 08003` 显示 08003 SQL 状态的帮助，而 `? 08` 显示 08 类代码的帮助。

访问不同版本的 DB2 信息中心

您可以在 ibm.com® 上的不同信息中心中找到其他版本 DB2 产品的文档。

关于此任务

对于 DB2 V10.1 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v10r1。

对于 DB2 V9.8 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/。

对于 DB2 V9.7 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/。

对于 DB2 V9.5 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9r5。

对于 DB2 V9.1 主题，*DB2 信息中心* URL 是 http://publib.boulder.ibm.com/infocenter/db2luw/v9/。

对于 DB2 V8 主题，请转至 *DB2 信息中心* URL: http://publib.boulder.ibm.com/infocenter/db2luw/v8/。

更新安装在计算机或内部网服务器上的 DB2 信息中心

安装在本地的 DB2 信息中心必须定期进行更新。

开始之前

必须已安装 DB2 V10.1 信息中心。有关详细信息，请参阅安装 DB2 服务器中的“使用 DB2 安装向导来安装 DB2 信息中心”主题。所有适用于安装信息中心的先决条件和限制同样适用于更新信息中心。

关于此任务

可以自动或手动更新现有的 DB2 信息中心:

- 自动更新将更新现有的信息中心功能部件和语言。自动更新的一个优点是，与手动更新相比，信息中心的不可用时间较短。另外，自动更新可设置为作为定期运行的其他批处理作业的一部分运行。
- 可以使用手动更新方法来更新现有的信息中心功能部件和语言。自动更新可以缩短更新过程中的停机时间，但如果您想添加功能部件或语言，那么必须执行手动过程。例如，如果本地信息中心最初安装的是英语和法语版，而现在还要安装德语版；那么手动更新将安装德语版，并更新现有信息中心的功能和语言。但是，手动更新要求您手动停止、更新和重新启动信息中心。在整个更新过程期间信息中心不可用。在自动更新过程中，信息中心仅在更新完成后停止工作以重新启动信息中心。

此主题详细说明了自动更新的过程。有关手动更新的指示信息，请参阅“手动更新安装在您的计算机或内部网服务器上的 DB2 信息中心”主题。

过程

要自动更新安装在计算机或内部网服务器上的 DB2 信息中心:

1. 在 Linux 操作系统上，
   a. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 `/opt/ibm/db2ic/V10.1` 目录中。
   b. 从安装目录浏览至 `doc/bin` 目录。
   c. 运行 `update-ic` 脚本:

```
update-ic
```

2. 在 Windows 操作系统上，
   a. 打开命令窗口。
   b. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 <Program Files>\IBM\DB2 Information Center\V10.1 目录中，其中 <Program Files> 表示 Program Files 目录的位置。
   c. 从安装目录浏览至 doc\bin 目录。
   d. 运行 update-ic.bat 文件:

```
update-ic.bat
```

结果

DB2 信息中心将自动重新启动。如果更新可用，那么信息中心会显示新的以及更新后的主题。如果信息中心更新不可用，那么会在日志中添加消息。日志文件位于 doc\eclipse\configuration 目录中。日志文件名称是随机生成的编号。例如，1239053440785.log。

手动更新安装在计算机或内部网服务器上的 DB2 信息中心

如果您已在本地安装 DB2 信息中心，那么可从 IBM 获取文档更新并进行安装。

关于此任务

手动更新安装在本地的 *DB2 信息中心*要求您:

1. 停止计算机上的 *DB2 信息中心*，然后以独立方式重新启动信息中心。如果以独立方式运行信息中心，那么网络上的其他用户将无法访问信息中心，因而您可以应用更新。DB2 信息中心的工作站版本总是以独立方式运行。
2. 使用“更新”功能部件来查看可用的更新。如果有您必须安装的更新，那么请使用“更新”功能部件来获取并安装这些更新。

   注: 如果您的环境要求在一台未连接至因特网的机器上安装 *DB2 信息中心*更新，请使用一台已连接至因特网并已安装 *DB2 信息中心*的机器将更新站点镜像至本地文件系统。如果网络中有许多用户将安装文档更新，那么可以通过在本地也为更新站点制作镜像并为更新站点创建代理来缩短每个人执行更新所需要的时间。
   如果提供了更新包，请使用“更新”功能部件来获取这些更新包。但是，只有在单机方式下才能使用“更新”功能部件。
3. 停止独立信息中心，然后在计算机上重新启动 *DB2 信息中心*。

注: 在 Windows 2008、Windows Vista 和更高版本上，稍后列示在此部分的命令必须作为管理员运行。要打开具有全面管理员特权的命令提示符或图形工具，请右键单击快捷方式，然后选择**以管理员身份运行**。

过程

要更新安装在您的计算机或内部网服务器上的 *DB2 信息中心*:

1. 停止 *DB2 信息中心*。
   - 在 Windows 上，单击**开始 > 控制面板 > 管理工具 > 服务**。右键单击 **DB2 信息中心**服务，并选择**停止**。

- 在 Linux 上，输入以下命令:

  ```
  /etc/init.d/db2icdv10 stop
  ```

2. 以独立方式启动信息中心。
   - 在 Windows 上:
     a. 打开命令窗口。
     b. 浏览至信息中心的安装位置。缺省情况下，*DB2 信息中心*安装在 *Program_Files*\IBM\DB2 Information Center\V10.1 目录中，其中 *Program Files* 表示 Program Files 目录的位置。
     c. 从安装目录浏览至 doc\bin 目录。
     d. 运行 help_start.bat 文件:

        ```
        help_start.bat
        ```

   - 在 Linux 上:
     a. 浏览至信息中心的安装位置。缺省情况下，*DB2 信息中心*安装在 /opt/ibm/db2ic/V10.1 目录中。
     b. 从安装目录浏览至 doc/bin 目录。
     c. 运行 help_start 脚本:

        ```
        help_start
        ```

   系统缺省 Web 浏览器将打开以显示独立信息中心。
3. 单击**更新**按钮（  ）。（必须在浏览器中启用 JavaScript。）在信息中心的右边面板上，单击**查找更新**。将显示现有文档的更新列表。
4. 要启动安装过程，请检查您要安装的选项，然后单击**安装更新**。
5. 在安装进程完成后，请单击**完成**。
6. 要停止独立信息中心，请执行下列操作:
   - 在 Windows 上，浏览至安装目录中的 doc\bin 目录并运行 help_end.bat 文件:

     ```
     help_end.bat
     ```

     注: help_end 批处理文件包含安全地停止使用 help_start 批处理文件启动的进程所需的命令。不要使用 Ctrl-C 或任何其他方法来停止 help_start.bat。
   - 在 Linux 上，浏览至安装目录中的 doc/bin 目录并运行 help_end 脚本:

     ```
     help_end
     ```

     注: help_end 脚本包含安全地停止使用 help_start 脚本启动的进程所需的命令。不要使用任何其他方法来停止 help_start 脚本。
7. 重新启动 *DB2 信息中心*。
   - 在 Windows 上，单击**开始 > 控制面板 > 管理工具 > 服务**。右键单击 **DB2 信息中心**服务，并选择**启动**。
   - 在 Linux 上，输入以下命令:

     ```
     /etc/init.d/db2icdv10 start
     ```

结果

更新后的 *DB2 信息中心*将显示新的以及更新后的主题。

DB2 教程

DB2 教程帮助您了解 DB2 数据库产品的各个方面。这些课程提供了逐步指示信息。

开始之前

您可以在信息中心中查看 XHTML 版的教程: http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/。

某些课程使用了样本数据或代码。有关其特定任务的任何先决条件的描述，请参阅教程。

DB2 教程

要查看教程，请单击标题。

*pureXML 指南*中的『**pureXML®**』

设置 DB2 数据库以存储 XML 数据以及对本机 XML 数据存储器执行基本操作。

DB2 故障诊断信息

我们提供了各种各样的故障诊断和问题确定信息来帮助您使用 DB2 数据库产品。

DB2 文档

您可以在*故障诊断和调整数据库性能*或者 *DB2 信息中心*的"数据库基础"部分中找到故障诊断信息，这些信息包含以下内容:

- 有关如何使用 DB2 诊断工具和实用程序来隔离和确定问题的信息。
- 一些最常见问题的解决方案。
- 旨在帮助您解决 DB2 数据库产品使用过程中可能会遇到的其他问题的建议。

IBM 支持门户网站

如果您遇到问题并且希望得到帮助以查找可能的原因和解决方案，请访问 IBM 支持门户网站。这个技术支持站点提供了指向最新 DB2 出版物、技术说明、授权程序分析报告（APAR 或错误修订）、修订包和其他资源的链接。可搜索此知识库并查找问题的可能解决方案。

访问 IBM 支持门户网站: http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

信息中心条款和条件

如果符合以下条款和条件，那么授予您使用这些出版物的许可权。

适用性: 用户需要遵循 IBM Web 站点的使用条款及以下条款和条件。

个人使用: 只要保留所有的专有权声明，您就可以为个人、非商业使用复制这些出版物。未经 IBM 明确同意，您不可以分发、展示或制作这些出版物或其中任何部分的演绎作品。

商业使用: 只要保留所有的专有权声明，您就可以仅在企业内复制、分发和展示这些出版物。未经 IBM 明确同意，您不可以制作这些出版物的演绎作品，或者在您的企业外部复制、分发或展示这些出版物或其中的任何部分。

权利：除非本许可权中明确授予，否则不得授予对这些出版物或其中包含的任何信息、数据、软件或其他知识产权的任何许可权、许可证或权利，无论是明示的还是暗含的。

IBM 保留根据自身的判断，认为对出版物的使用损害了 IBM 的权益（由 IBM 自身确定）或未正确遵循以上指示信息时，撤回此处所授予权限的权利。

只有您完全遵循所有适用的法律和法规，包括所有的美国出口法律和法规，您才可以下载、出口或再出口该信息。

IBM 对这些出版物的内容不作任何保证。这些出版物“按现状”提供，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的关于适销和适用于某种特定用途的保证。

IBM Trademarks: IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml

附录 B. 声明

本信息是为在美国提供的产品和服务编写的。有关非 IBM 产品的信息是基于首次出版此文档时的可获信息且会随时更新。

IBM 可能在其他国家或地区不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。您可以用书面方式将许可查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节字符集 (DBCS) 信息的许可查询，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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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: International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。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 站点中的资料不是此 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他程序（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
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

只要遵守适当的条款和条件，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 国际软件许可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，在其他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、其出版说明或其他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、兼容性或任何其他关于非 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。

本信息可能包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、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:

本信息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，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。此样本程序“按现状”提供，且不附有任何种类的保证。对于使用此样本程序所引起的任何损坏，IBM 将不承担责任。

凡这些样本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.

商标

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other prod-

uct and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at 『Copyright and trademark information』 at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

索引

[B]

[C]

[D]

[F]

[G]

[J]

[K]

[L]

[M]

[数字]

A

C

D

E

I

J

M

N

O

R

S

T

U

V

X

[特别字符]

Printed in China

S151-1765-00

Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

开发 ADO.NET 和 OLE DB 应用程序