DB2 9 应用开发

CLI/ODBC 编程简介

　　什么是 CLI/ODBC？

　　结构化查询语言（Structured Query Language，SQL）是用于操纵数据库对象和它们包含的数据的一种标准语言。但是，由于 SQL 没有过程语言的性质，因此，通常是将高级编程语言的决策和顺序控制与 SQL 的数据存储、操纵和检索功能相结合来开发数据库应用程序。有一些方法可以将 SQL 与高级编程语言相结合，但最简单的方法是将 SQL 语句直接嵌入到用于创建应用程序的高级编程语言源代码文件中。这种技术被成为嵌入式 SQL 编程。

嵌入式 SQL 编程最大的缺点是所开发的应用程序缺乏互操作性。用嵌入式 SQL 为 DB2 开发的应用程序如果要与其他关系数据库管理系统（RDBMS）交互，必须进行修改（在某些情况下，甚至要完全重写）。由于不论为何种 RDBMS 编写的嵌入式 SQL 应用程序中都存在这样的限制，因此在 20 世纪 90 年代初，X/Open 公司和 SQL Access Group（SAG，现在属于 X/Open）就为可调用 SQL 接口联合开发了一种标准的规范。这种接口被称为 X/Open CLI。大部分 X/Open CLI 规范后来都被接受为 ISO CLI 国际标准的一部分。 X/Open CLI 的主要目的是通过允许数据库应用程序独立于任何一种数据库管理系统的编程接口，增加数据库应用程序的可移植性。

在 1992，Microsoft 为 Microsoft Windows [操作系统](http://tech.ddvip.com/os/index.html)开 发了一个名为 Open Database Connectivity (ODBC) 的可调用 SQL 接口。ODBC 基于 X/Open CLI 标准规范，它提供了 X/Open CLI 没有提供的扩展功能和能力。ODBC 位于一个操作环境之上，在此环境中，一个名为 ODBC Driver Manager 的组件在应用程序运行时动态地装载特定于数据源的 ODBC 驱动程序。每个特定于数据源的驱动程序负责实现 ODBC 规范中定义的一种或全部功能，并提供与该驱动程序所针对的特定数据源的交互。ODBC Driver Manager 提供了一个集中控制点；当一个 ODBC 应用程序执行时，发出的每个 ODBC 函数调用被发送到 ODBC Driver Manager，并从那里转发到适当的数据源驱动程序进行处理。通过驱动程序，可以将应用程序直接链接到一个 ODBC 驱动程序库，而不必链接到每种特定于产品的数据库本身。

　　DB2 Call Level Interface (DB2 CLI) 基于 ISO CLI 国际标准，它提供了 ODBC 规范中给出的大部分功能。使用 DB2 CLI 而不是 ODBC 的应用程序直接链接到 DB2 CLI 加载库，任何 ODBC Driver Manager 都可以加载这个库作为一个 ODBC 驱动程序。DB2 UDB 应用程序还可以独立地使用 DB2 CLI 装载库。但是，当以这种方式使用这个库时，应用程序本身不能与其他数据源[通信](http://tech.ddvip.com/comm/" \t "_blank)。

　　嵌入式 SQL 与 CLI/ODBC 的区别

　 　在本系列的第四篇教程中可以看到，嵌入式 SQL 应用程序是通过直接将 SQL 语句嵌入到使用高级编程语言编写的一个或多个源代码文件中而构建的。而 CLI/ODBC 应用程序则依赖于一组标准的应用程序编程接口（API）函数将 SQL 语句发送到 DB2 Database Manager 进行处理。嵌入式 SQL 应用程序和 CLI/ODBC 应用程序在以下方面也有不同之处：

　　CLI/ODBC 应用程序不需要显式地声明和使用主机变量；可以使用任何变量来发送数据或者从一个数据源检索数据。

　　CLI/ODBC 应用程序不必显式地声明游标。相反，每当执行 SQLExecute() 函数或 SQLExecDirect() 函数（稍后会更详细讨论这两个函数）时，会根据需要自动生成游标。

　　在 CLI/ODBC 应用程序中，不需要显式地打开游标；当生成游标时，会自动打开游标。

　 　CLI/ODBC 函数使用句柄 管理环境、连接和与 SQL 语句相关的信息。这种技术便于将那些信息当作抽象对象来对待。通过使用句柄，CLI/ODBC 应用程序就不必使用特定于数据库产品的数据结构，例如 DB2 SQL Communications Area (SQLCA) 和 SQL Descriptor Area (SQLDA) 数据结构。

　　CLI/ODBC 应用程序天生就具有建立到多个数据源或同一个数据源的多个连接的能力。（嵌入式 SQL 应用程序只有在使用 Type 2 连接的情况下，才能一次连接到多个数据源。）

　 　嵌入式 SQL 应用程序和 CLI/ODBC 应用程序虽然有这么多不同之处，但是它们在概念上也有一个重要的共同点：CLI/ODBC 应用程序可以执行任何能在嵌入式 SQL 应用程序中动态准备的任何 SQL 语句。这一点是有保障的，因为 CLI/ODBC 应用程序将所有的 SQL 语句直接传递到数据源，以便动态执行这些语句。（CLI/ODBC 应用程序还可以执行一些不能动态准备的 SQL 语句，例如复合 SQL 语句，但是通常上不支持静态 SQL。）

　　由于是由数据源来处理 CLI/ODBC 应用程序提交的所有 SQL 语句，这就保证了 CLI/ODBC 应用程序的可移植性。而嵌入式 SQL 并不总是如此，因为如果所使用的关系数据库产品不同，动态准备 SQL 语句的方式也会有所不同。另外，由于有些数据库产品（包括 DB2）可以动态准备 COMMIT 和 ROLLBACK SQL 语句，而有些数据库产品又不能，因此 CLI/ODBC 应用程序中通常不使用这些语句。相反，CLI/ODBC 应用程序依靠 SQLEndTran() 函数结束活动事务（不使用自动提交的情况下）。这种技术可保证不管使用何种数据库产品，CLI/ODBC 应用程序都能成功地结束事务。

　　构造 CLI/ODBC 应用程序

　　CLI/ODBC 应用程序的组成部分

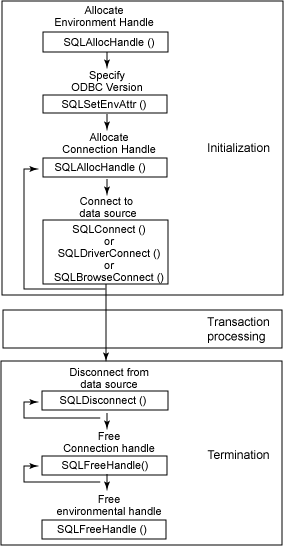
　　所有 CLI/ODBC 应用程序都要被构造来执行以下三个不同的任务：

　　初始化

　　事务处理

　　终止

　　每个任务涉及到的工作是通过调用一个或多个 CLI/ODBC API 函数来实现的。用于实现这些任务的很多 CLI/ODBC 函数必须按照特定的顺序来调用，否则会发生错误。图 1 标出了用于执行初始化和终止任务的一些基本的 CLI/ODBC 函数：

图 1. CLI/ODBC 应用程序执行的基本任务 

CLI/ODBC 应用程序还可以执行图 1 中列出的 3 个任务以外的任务，例如错误处理和消息处理。在后面的小节 诊断和错误处理 中，可以看到 CLI/ODBC 应用程序中如何处理错误。

　　分配资源

　 　在初始化期间，需要分配（和初始化）处理事务所需的资源，并建立到事务处理任务需要使用的数据源的连接。CLI/ODBC 应用程序使用的资源由通过惟一句柄标识的专用数据存储区组成。（句柄是一个简单的指针变量，它指向 DB2 CLI 或 ODBC Driver Manager 所控制的数据对象，该数据对象由 CLI/ODBC 函数调用引用。）通过使用数据存储区和句柄，CLI/ODBC 应用程序不必负责分配和管理全局变量和诸如嵌入式 SQL 应用程序中使用的 SQLCA 和 SQLDA 之类的数据结构。有四种不同类型的句柄：

　　环境句柄： 指向一个数据存储区的指针，该数据存储区包含特定于 CLI/ODBC 的全局信息。

　　连接句柄：指向一个数据存储区的指针，该数据存储区包含关于 CLI/ODBC 管理的数据源（数据库）连接的信息。

　　语句句柄：指向指向数据存储区域的指针，该数据存储区包含单个 SQL 语句有关的特定信息。

　　描述符句柄：指向一个数据存储区域的指针，该数据存储区包含一个元数据集合，这些元数据描述被绑定到 SQL 语句中的参数标记的应用程序变量，或者被绑定到查询结果数据集中列的应用程序变量。

　 　每个 CLI/ODBC 应用程序必须从分配一个环境句柄开始。通常每个应用程序只分配一个环境句柄，并且在分配任何其他句柄之前，那个环境句柄必须已经存在。所有其他句柄都是在 环境句柄使用的上下文环境中管理的。环境句柄可以通过调用 SQLAllocHandle() 函数并指定 SQL\_HANDLE\_ENV 选项来分配。用于分配环境句柄的源代码如下所示：

SQLHANDLE　EnvHandle　=　0;

...

SQLAllocHandle(SQL\_HANDLE\_ENV,　SQL\_NULL\_HANDLE,　&EnvHandle);

　 　在 CLI/ODBC 应用程序中，到数据源的连接是通过连接句柄实现的。因此，在建立到任何数据源的连接之前，必须存在一个连接句柄。可以通过调用 SQLAllocHandle() 函数并指定 SQL\_HANDLE\_DBC 选项和一个有效的环境句柄来分配连接句柄。用于分配连接句柄的源代码如下所示：

SQLHANDLE　ConHandle　=　0;

...

if　(EnvHandle　!=　0)

　　SQLAllocHandle(SQL\_HANDLE\_DBC,　EnvHandle,　&ConHandle);

　　CLI/ODBC 应用程序的真正骨干是语句句柄。语句句柄用于：

　　将应用程序变量绑定到 SQL 语句中使用的参数标记。

　　准备和提交 SQL 语句到适当的数据源，以便执行。

　　获得关于 SQL 语句产生的结果数据集的元数据。

　　将应用程序变量绑定到结果数据集中的列。

　　从结果数据集中检索（读取）数据。

　　当 SQL 未能执行时，获得诊断信息。

　 　CLI/ODBC 应用程序中编写的每条 SQL 语句都必须有它自己的语句句柄。每个语句句柄只能与一个连接句柄相关联。但是，一个连接句柄可以与任意数量的语句句柄相关联。语句句柄可以通过调用 SQLAllocHandle() 函数并指定 SQL\_HANDLE\_STMT 选项和一个有效的连接句柄来分配。因此，用于分配语句句柄的源代码如下所示：

SQLHANDLE　StmtHandle　=　0;

...

if　(ConHandle　!=　0)

　　SQLAllocHandle(SQL\_HANDLE\_STMT,　ConHandle,　&StmtHandle);

　 　每当分配一个语句句柄时，就会自动分配与语句句柄相关联的 4 个描述符句柄。分配后，这些描述符句柄继续与相应的语句句柄相关联，直到它被销毁（那时，描述符句柄也随之被销毁）。大多数 CLI/ODBC 操作都可以使用这些隐式定义的描述符句柄来执行。但是，也可以通过调用 SQLAllocHandle() 函数并指定 SQL\_HANDLE\_DESC 选项和有效的连接句柄来显式地分配描述符句柄。

　　声明应用程序 CLI/ODBC 版本

CLI 和 ODBC 使用特定于产品的驱动程序来与数据源（数据库）通信，大多数驱动程序都包含一组动态参数，通过修改这些参数，可以更改驱动程序的行为，以满足应用程序的需 求。这些参数被成为属性，每个环境句柄、连接句柄和语句句柄都有其自己的一组属性。（后面有一个小节描述用于获取和更改与环境句柄、连接句柄或语句句柄相 关联的属性。）环境属性的一个例子是 SQL\_ATTR\_ODBC\_VERSION 属性，在分配环境句柄之后，分配任何相应的连接句柄之前，必须为该属性赋予 SQL\_OV\_ODBC3 或 SQL\_OV\_ODBC2 值。这告诉 DB2 CLI 和 ODBC Driver Manager，应用程序打算遵从 CLI/ODBC 3.x 规范或 CLI/ODBC 2.0（或更早版本）规范。用于告诉 DB2 CLI 或 ODBC Driver Manager 应用程序打算遵从 CLI/ODBC 3.x（或更高版本）规范的代码如下所示：

...

SQLSetEnvAttr(EnvHandle,　SQL\_ATTR\_ODBC\_VERSION,　(SQLPOINTER)　SQL\_OV\_ODBC3,

　　SQL\_IS\_UINTEGER);

　　让 DB2 CLI 和 ODBC Driver Manager 知道应用程序遵从何种规范很重要，因为 CLI/ODBC 函数返回的很多返回码（或者被称为 SQLSTATE，SQLSTATEs 中会详细谈到）的值会因版本的不同而不同。此外，DB2 CLI 和 ODBC 的较高版本允许在某些函数参数中使用通配符，而较早版本则不允许。

　　建立数据源（数据库）连接

　　要对数据库执行任何类型的操作，必须首先建立到数据库的连接。对于 CLI/ODBC 应用程序，有 3 个函数可用于建立数据源（数据库）连接：

SQLConnect()

SQLDriverConnect()

SQLBrowseConnect()

　　应用程序可以使用这 3 个函数的任意组合连接到任意数目的数据源，但是某些数据源可能会限制所支持的活动连接的数量。（应用程序可以通过调用 SQLGetInfo() 函数并指定 SQL\_MAX\_DRIVER\_CONNECTIONS 信息类型来发现一个数据源支持多少活动连接。

　　 SQLConnect() 函数目前是最简单的 CLI/ODBC 连接函数。当使用 SQLConnect() 函数时，它假设建立连接所需的惟一信息就是一个数据源名称，可能还有一个用户 ID（授权 ID）和密码。（其他必要的信息存储在 db2cli.ini 文件的 [COMMON] 部分、ODBC.INI 文件的 [ODBC] 部分或系统注册表中的 ODBC 子键中。）该函数非常适用于只需要用户 ID 和口令来连接数据源的应用程序，以及需要提供其自己的连接界面或根本无需用户界面的应用程序。

　　而 SQLDriverConnect() 函数则允许应用程序使用一个连接字符串将连接信息发送到一个数据源驱动程序（而不是将该信息存储在 db2cli.ini 文件、ODBC.INI 文件或系统注册表中，然后允许驱动程序来检索）。连接字符串是一系列的键/值对，之间以分号隔开，其中包含建立到数据源连接时所需的信息。表 1 列出了一些较常用的键/值对。

表 1. 和 SQLDriverConnect() 一起使用的键/值对

|  |  |
| --- | --- |
| 键/值 | 用途 |
| DRIVER=DriverName | 指定建立到数据库的连接所需使用的 CLI/ODBC 驱动程序的名称 |
| DBALIAS=DatabaseAlias | 指定要与之建立连接的数据库的别名 |
| DSN=DataSourceName | 指定要与之建立连接的数据源的名称（与 SQLDataSources() 函数返回的值一致） |
| UID=UserID | 指定尝试建立连接的用户的用户 ID（授权 ID） |
| PWD=Password | 指定所指定用户 ID（授权 ID）对应的密码。如果所指定的 ID 不需要密码，则应该使用一个空的密码字符串（PWD=;） |
| NEWPWD=NewPassword | 指定为所指定用户 ID（授权 ID）分配的新密码。如果使用了 NEWPWD 但是没有提供新密码（NEWPWD=;），则 DB2 CLI 驱动程序提示用户提供一个新密码 |

　　因此，若要使用 SQLDriverConnect() 函数建立到一个已编目的、名为 PAYROLL 的、使用授权 ID db2admin 和密码 ibmdb2 的数据库的连接，可以使用如下代码：

...

char　ConnectStr[512];

...

sprintf(ConString,　"driver={IBM　DB2　ODBC　DRIVER};dbalias=payroll;

　　uid=db2admin;pwd=ibmdb2");

SQLDriverConnect(ConHandle,　NULL,　(SQLCHAR　\*)　ConnectStr,　SQL\_NTS,

　　NULL,　0,　NULL,　SQL\_DRIVER\_NOPROMPT);

...

　　您可能注意到，有一个名为 SQL\_NTS 的特殊代码被作为 SQLDriverConnect() 函数的一个参数值。接受字符串值作为参数的 CLI/ODBC 函数通常要求同时提供字符串的长度。可以使用 SQL\_NTS 值代替实际长度值，表明相应的字符串是以 null 终止的。

　　当被调用时，SQLDriverConnect() 函数使用指定的数据源名称解析连接字符串，并尝试从系统中获取附加的信息，以建立连接。然后，该函数使用这些信息登录到适当的[服务器](http://tech.ddvip.com/server/index.html" \t "_blank)上，并尝试连接到指定的数据源。使用 SQLDriverConnect() 函数的应用程序还可以让驱动程序提示用户提供所需的连接信息。例如，当用一个空连接字符串调用 SQLDriverConnect() 函数时，DB2 CLI 显示一个对话框，提示用户：

　　从 DB2 CLI 认识的数据源列表中选择一个数据源。

　　提供一个用户 ID 和相应的密码。

　　指定连接模式是独占的还是共享的。

是否显示该对话框由传递给 SQLDriverConnect() 函数的一个参数值控制：如果调用该函数时指定了 SQL\_DRIVER\_PROMPT、SQL\_DRIVER\_COMPLETE 或 SQL\_DRIVER\_COMPLETE\_REQUIRED 选项，那么，如果提供的连接字符串没有包括足够的信息来建立数据源连接，则显示该对话框。但如果指定了 SQL\_DRIVER\_NOPROMPT 选项，但是没有提供足够的信息，那么就会返回错误。

　　和 SQLDriverConnect() 函数一样，SQLBrowseConnect() 函数使用一个连接字符串将连接信息发送给驱动程序。SQLDriverConnect() 函数在编译时需要一个有效的连接字符串，而 SQLBrowseConnect() 函数可用于在应用程序运行时构造一个连接。这点差异使应用程序可以使用它自己的对话框提示用户提供连接信息，从而保留对其外观的控制。

　　事务处理

　 　CLI/ODBC 应用程序执行完适当的初始化之后，焦点就转移到事务处理上来。在此期间，通过不同的 CLI/ODBC 函数调用，将查询和操纵数据的 SQL 语句传递到适当的数据源（在这里通常是 DB2 数据库）进行处理。在事务处理期间，CLI/ODBC 应用程序依次执行以下五个步骤：

　　分配一个或多个语句句柄。

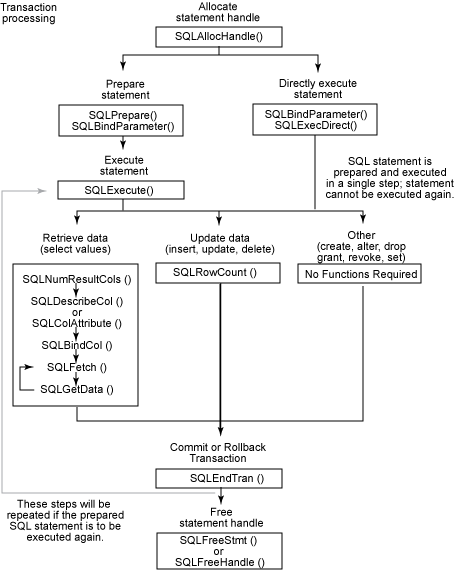
　　准备和执行一个或多个语句。

　　检索和处理产生的结果。

　　通过提交或回滚终止当前事务。

　　释放分配的所有语句句柄。

　　图 2 显示了事务处理期间执行的基本步骤，并给出了常用于执行每个步骤的 CLI/ODBC 函数调用：

图 2. 在 CLI/ODBC 应用程序中处理事务 

　　分配语句句柄

　　正如前面提到的，语句句柄引用一个数据对象，该对象包含单个 SQL 语句的有关信息。这些信息包括：

　　SQL 语句的文本

　　关于与语句相关联的游标的详细信息

　　所有 SQL 语句参数标记变量的绑定

　　所有结果数据集列变量的绑定

　　语句执行的返回码

　　状态信息

　　SQL 语句句柄是通过调用 SQLAllocHandle() 函数并指定 SQL\_HANDLE\_STMT 选项和一个有效的连接句柄来分配的。在 CLI/ODBC 应用程序执行任何 SQL 语句之前，必须至少分配一个语句句柄。

　　准备和执行 SQL 语句

　　分配好语句句柄后，就可以将一个 SQL 语句赋给它，处理 SQL 语句的方式有两种：

　 　准备和执行:这种方法将 SQL 语句的准备与执行分开，通常用于需要重复执行语句的情况。如果应用程序需要预知存在于执行 SQL 语句所产生的结果数据集中列的有关信息，也使用该方法。这种方式使用 CLI/ODBC 函数 SQLPrepare() 和 SQLExecute() 来处理 SQL 语句。

　　立即执行：该方法将 SQL 语句的准备和执行合并为一个步骤，通常用于语句只执行一次的情况。如果应用程序不需要关于 SQL 语句执行时所产生的结果数据集的附加信息，那么也可以使用该方法。这种方式使用 CLI/ODBC 函数 SQLExecDirect() 来处理 SQL 语句。

　　这两种方法都允许使用参数标记代替被处理的 SQL 语句中的常量和表达式。参数标记用问号（?）表示，用来指示当执行 SQL 语句时将在 SQL 语句中的何处替换一个或多个应用程序变量的当前值。当一个应用程序变量与 SQL 语句中一个参数标记关联时，就称该变量被“绑定”到该参数标记。这种绑定是通过调用 SQLBindParameter() 函数完成的，应用程序变量被绑定到参数标记之后，参数标记与该变量的关联就一直有效，直到它被重载或者相应的语句句柄被释放。虽然在 SQL 语句准备好之后可以随时进行绑定，但是只有到 SQL 语句执行的时候，才能真正从绑定变量中检索数据。

　　清单 1 展示了在一个用 C 语言编写的 CLI/ODBC 应用程序中，如何将应用程序变量绑定到简单的 SELECT SQL 语句中的参数标记。它还展示了在语句执行之前，将值以何种方式提供给绑定参数。

清单 1. 绑定参数标记

...

//　Define　A　SELECT　SQL　Statement　That　Uses　A　Parameter　Marker

strcpy((char　\*)　SQLStmt,　"SELECT　empno,　lastname　FROM　");

strcat((char　\*)　SQLStmt,　"employee　WHERE　JOB　=　?");

//　Prepare　The　SQL　Statement

RetCode　=　SQLPrepare(StmtHandle,　SQLStmt,　SQL\_NTS);

//　Bind　The　Parameter　Marker　Used　In　The　SQL　Statement　To

//　An　Application　Variable

RetCode　=　SQLBindParameter(StmtHandle,　1,

　　　　　　　SQL\_PARAM\_INPUT,　SQL\_C\_CHAR,　SQL\_CHAR,

　　　　　　　sizeof(JobType),　0,　JobType,

　　　　　　　sizeof(JobType),　NULL);

//　Populate　The　"Bound"　Application　Variable

strcpy((char　\*)　JobType,　"DESIGNER");

//　Execute　The　SQL　Statement

RetCode　=　SQLExecute(StmtHandle);

...

检索和处理结果

　　在准备并执行了 SQL 语句之后，需要检索和处理该语句所产生的结果。如果 SQL 语句是 SELECT 或 VALUES 以外的语句，那么在该语句执行之后，惟一需要做的是检查 CLI/ODBC 函数返回码，以确认该语句是否按预期执行。但是，如果执行的 SQL 语句是一个查询，那么可能还需要进行相应的处理来从产生的结果数据集中检索数据。

　　当查询返回多个行时，这些行被存储在与执行的 SQL 语句相关联的连接句柄和语句句柄所引用的数据存储区中。因此，可能需要执行以下步骤来从产生的结果数据集中检索数据：

　　确定产生的结果数据集的结构（即列的数量、列的数据类型、数据长度）。这可以通过执行 SQLNumResultCols()、SQLDescribeCol() 和 SQLColAttributes() 函数来完成。

　　使用 SQLBindCol() 函数将应用程序变量绑定到结果数据集中的列上（可选）。

　 　重复地从产生的结果数据集读取下一行，并将之复制到绑定的应用程序变量中。这通常是通过在一个循环中重复地调用 SQLFetch() 函数来实现的。（每当读取一个新行时，步骤 2 中未绑定到应用程序变量绑定的列值可以通过调用 SQLGetData() 函数获得。）

　 　在第一步中，首先分析所准备或执行的 SQL 语句，以确定产生的结果数据集的结构。如果 SQL 语句被硬编码到应用程序中，那么不需要这一步，因为产生的结果数据集的结构是已知的。但是，如果 SQL 语句是在应用程序运行时生成的，那么就必须查询该语句产生的结果数据集，以获得该信息。

　　知道结果数据集的结构之后，可以将一个或多个应 用程序变量绑定到结果数据集中特定的列上（就像将应用程序变量绑定到 SQL 语句参数标记一样）。在这种情况下，应用程序变量被用作输出参数，而不是输入参数，当调用 SQLFetch() 函数时，就会检索出数据并直接写入这些变量中。但是，由于可以使用 SQLGetData() 函数从结果数据集中检索数据，因此应用程序变量/列绑定是可选的。

　　在第三步中，通过重复调用 SQLFetch() 函数（通常在循环中），检索存储在结果数据集中的数据，直到数据被检索完。如果已经将应用程序变量绑定到结果数据集中的列上，那么每当调用 SQLFetch() 时，这些变量的值就会自动更新。另一方面，如果没有执行列绑定，那么可以使用 SQLGetData() 函数将数据从特定列中复制到适当的应用程序变量中。SQLGetData() 函数还可用于在多个小部分中检索大型可变长度的列数据值（当使用绑定应用程序变量时不能这样做）。结果数据集中存储的所有数据都可以使用这两种方法的任意 组合来检索。

　　清单 2 展示了在用 C 语言编写的 CLI/ODBC 应用程序中，如何将一个应用程序变量绑定到结果数据集中的列上。它还展示了通常如何通过重复地调用 SQLFetch()检索结果数据集中的数据。

清单 2. 绑定列

...

//　Bind　The　Columns　In　The　Result　Data　Set　Returned

//　To　Application　Variables

SQLBindCol(StmtHandle,　1,　SQL\_C\_CHAR,　(SQLPOINTER)

　　EmpNo,　sizeof(EmpNo),　NULL);

SQLBindCol(StmtHandle,　2,　SQL\_C\_CHAR,　(SQLPOINTER)

　　LastName,　sizeof(LastName),　NULL);

//　While　There　Are　Records　In　The　Result　Data　Set

//　Produced,　Retrieve　And　Display　Them

while　(RetCode　!=　SQL\_NO\_DATA)

{

　　RetCode　=　SQLFetch(StmtHandle);

　　if　(RetCode　!=　SQL\_NO\_DATA)

　　　　printf("%-8s　%sn",　EmpNo,　LastName);

}

...

　　管理事务

　　您也许还记得，事务（也称工作单元）是组合成一个单元的一个或多个 SQL 操作的序列，它们通常位于一个应用程序进程中。个给定的事务可以包含从一个单一操作到成百甚至上千个任意数目的 SQL 操作，主要取决于何为您的业务逻辑中所认为的单步。事务很重要，因为一个事务的开始和终止可以确定数据库中的数据一致性点。在一个事务中，要么将事务中执 行的所有操作的结果持久地应用到数据库中（提交），要么回改所有操作的结果（回滚），并使数据库返回到启动事务之前所处的状态。

　　从事务 处理的角度来看，可以将 CLI/ODBC 应用程序配置为以两种模式中的一种来运行：自动提交或手动提交。当使用自动提交模式时，每个 SQL 语句都被当作一个完整的事务，在 SQL 语句成功执行之后，每个事务都自动提交。对于除 SELECT SQL 语句以外的语句，在语句执行之后就立即发生提交操作。对于 SELECT 语句，当用于处理结果数据集的游标关闭后，就立即发生提交操作。（记住，CLI/ODBC 必要时会自动声明并打开一个游标。）自动提交模式是默认的提交模式，对于简单的 CLI/ODBC 应用程序通常就足够了。但是，对于一些大型应用程序，尤其是执行更新操作的那些应用程序，在建立一个数据源连接之后，应该立即切换为手动提交模式。用于切 换到手动提交模式的函数调用如下所示：

SQLSetConnectAttr(ConHandle,　SQL\_ATTR\_AUTOCOMMIT,　SQL\_AUTOCOMMIT\_OFF,　SQL\_IS\_UINTEGER);

　 　当使用手动提交模式时，会在应用程序首次访问一个数据源时隐式地开始事务，在调用 SQLEndTran() 函数时显式地结束事务。这个 CLI/ODBC 函数用于回滚或提交当前事务所做的所有更改。因此，从数据源第一次被访问开始，到调用 SQLEndTran() 函数为止，这段时间内在数据源上执行的所有操作都被当作一个事务。

　　释放语句句柄

　　当 SQL 语句的结果被处理完毕，并且不再需要事务处理开始时分配的 SQL 语句数据存储区时，应该释放为该数据存储保留的内存。要释放与特定语句句柄相关联的数据存储区，可以调用 SQLFreeHandle() 函数并指定 SQL\_HANDLE\_STMT 选项和适当的语句句柄（例如 SQLFreeHandle(SQL\_HANDLE\_STMT, StmtHandle)）来实现。当被调用时，CLI/ODBC 函数执行以下任务：

　　取消对所有之前绑定的参数应用程序变量的绑定

　　取消对所有之前绑定的列应用程序变量的绑定

　　关闭已打开的游标，并丢弃其结果

　　销毁指定的句柄，并释放所有相关资源

　　如果一个语句句柄没有被释放，那么它还可以用于处理其他 SQL 语句。

　　终止数据源连接

　　在 CLI/ODBC 应用程序终止，并且控制被返回到操作系统之前，应该终止已建立的所有数据源连接，并释放初始化期间分配的所有资源。（通常，这些资源由一个环境数据存储区和一个或多个连接数据存储区组成。）

　 　通过调用 SQLDisconnect() 函数并指定适当的连接句柄，终止已有的数据库连接。为了释放相应的连接数据存储区，用 SQL\_HANDLE\_DBC 选项调用 SQLFreeHandle() 函数并指定适当的连接句柄。当之前分配的所有连接数据存储区都被释放后，通过用 SQL\_HANDLE\_ENV 选项并指定适当的环境句柄来调用 SQLFreeHandle() 函数，释放与连接数据存储区相关联的环境数据存储区。

　　综述

　　检查了构成所有 CLI/ODBC 应用程序的基本组件之后，现在就让我们来看一看如何连接这些组件以产生与 DB2 数据库交互的 CLI/ODBC 应用程序。下面是用 C 语言编写的一个简单的 CLI/ODBC 应用程序，该应用程序获取并打印职位为 DESIGNER 的所有雇员的雇员编号和姓氏：

清单 3. 一个简单的 CLI/ODBC 应用程序

#include　<stdio.h>

#include　<string.h>

#include　<sqlcli1.h>

int　main()

{

　　//　Declare　The　Local　Memory　Variables

　　SQLHANDLE　EnvHandle　=　0;

　　SQLHANDLE　ConHandle　=　0;

　　SQLHANDLE　StmtHandle　=　0;

　　SQLRETURN　RetCode　=　SQL\_SUCCESS;

　　SQLCHAR　　SQLStmt[255];

　　SQLCHAR　　JobType[10];

　　SQLCHAR　　EmpNo[10];

　　SQLCHAR　　LastName[25];

　　/\*-----------------------------------------------------\*/

　　/\*　INITIALIZATION　　　　　　　　　　　　　　　　　　　\*/

　　/\*-----------------------------------------------------\*/

　　//　Allocate　An　Environment　Handle

　　SQLAllocHandle(SQL\_HANDLE\_ENV,　SQL\_NULL\_HANDLE,

　　　　&EnvHandle);

　　//　Set　The　ODBC　应用程序　Version　To　3.x

　　if　(EnvHandle　!=　0)

　　　　SQLSetEnvAttr(EnvHandle,　SQL\_ATTR\_ODBC\_VERSION,

　　　　　　(SQLPOINTER)　SQL\_OV\_ODBC3,　SQL\_IS\_UINTEGER);

　　//　Allocate　A　Connection　Handle

　　if　(EnvHandle　!=　0)

　　　　SQLAllocHandle(SQL\_HANDLE\_DBC,　EnvHandle,

　　　　　　&ConHandle);

　　//　Connect　To　The　Appropriate　Data　Source

　　if　(ConHandle　!=　0)

　　　　RetCode　=　SQLConnect(ConHandle,　(SQLCHAR　\*)　"SAMPLE",

　　　　　　　　　　　SQL\_NTS,　(SQLCHAR　\*)　"db2admin",

　　　　　　　　　　　SQL\_NTS,　(SQLCHAR　\*)　"ibmdb2",

　　　　　　　　　　　SQL\_NTS);

　　/\*-----------------------------------------------------\*/

　　/\*　TRANSACTION　PROCESSING　　　　　　　　　　　　　　　\*/

　　/\*-----------------------------------------------------\*/

　　//　Allocate　An　SQL　Statement　Handle

　　if　(ConHandle　!=　0　&&　RetCode　==　SQL\_SUCCESS)

　　　　SQLAllocHandle(SQL\_HANDLE\_STMT,　ConHandle,

　　　　　　&StmtHandle);

　　//　Define　A　SELECT　SQL　Statement　That　Uses　A　Parameter

　　//　Marker

　　strcpy((char　\*)　SQLStmt,　"SELECT　empno,　lastname　FROM　");

　　strcat((char　\*)　SQLStmt,　"employee　WHERE　job　=　?");

　　//　Prepare　The　SQL　Statement

　　RetCode　=　SQLPrepare(StmtHandle,　SQLStmt,　SQL\_NTS);

　　//　Bind　The　Parameter　Marker　Used　In　The　SQL　Statement　To

//　An　Application　Variable

　　RetCode　=　SQLBindParameter(StmtHandle,　1,

　　　　　　　　　SQL\_PARAM\_INPUT,　SQL\_C\_CHAR,　SQL\_CHAR,

　　　　　　　　　sizeof(JobType),　0,　JobType,

　　　　　　　　　sizeof(JobType),　NULL);

　//　Populate　The　"Bound"　Application　Variable

　　strcpy((char　\*)　JobType,　"DESIGNER");

　　//　Execute　The　SQL　Statement

　　RetCode　=　SQLExecute(StmtHandle);

　　//　If　The　SQL　Statement　Executed　Successfully,　Retrieve

　　//　The　Results

　　if　(RetCode　==　SQL\_SUCCESS)

　　{

　　　　//　Bind　The　Columns　In　The　Result　Data　Set　Returned

　　　　//　To　Application　Variables

　　　　SQLBindCol(StmtHandle,　1,　SQL\_C\_CHAR,　(SQLPOINTER)

　　　　　　EmpNo,　sizeof(EmpNo),　NULL);

　　　　SQLBindCol(StmtHandle,　2,　SQL\_C\_CHAR,　(SQLPOINTER)

　　　　　　LastName,　sizeof(LastName),　NULL);

　　　　//　While　There　Are　Records　In　The　Result　Data　Set

　　　　//　Produced,　Retrieve　And　Display　Them

　　　　while　(RetCode　!=　SQL\_NO\_DATA)

　　　　{

　　　　　　RetCode　=　SQLFetch(StmtHandle);

　　　　　　if　(RetCode　!=　SQL\_NO\_DATA)

　　　　　　　　printf("%-8s　%sn",　EmpNo,　LastName);

　　　　}

　　}

　　//　Commit　The　Transaction

　　RetCode　=　SQLEndTran(SQL\_HANDLE\_DBC,　ConHandle,　SQL\_COMMIT);

　　//　Free　The　SQL　Statement　Handle

　　if　(StmtHandle　!=　0)

　　　　SQLFreeHandle(SQL\_HANDLE\_STMT,　StmtHandle);

　　/\*-----------------------------------------------------\*/

　　/\*　TERMINATION　　　　　　　　　　　　　　　　　　　　　\*/

　　/\*-----------------------------------------------------\*/

　　//　Terminate　The　Data　Source　Connection

　　if　(ConHandle　!=　0)

　　　　RetCode　=　SQLDisconnect(ConHandle);

　　//　Free　The　Connection　Handle

　　if　(ConHandle　!=　0)

　　　　SQLFreeHandle(SQL\_HANDLE\_DBC,　ConHandle);

　　//　Free　The　Environment　Handle

　　if　(EnvHandle　!=　0)

　　　　SQLFreeHandle(SQL\_HANDLE\_ENV,　EnvHandle);

　　//　Return　Control　To　The　Operating　System

　　return(0);

}

控制 CLI/ODBC 驱动程序的行为

　　获得关于数据源的信息

　　由于 CLI/ODBC 应用程序可以连接到很多类型的数据源，因此有时候需要获取关于应用程序所连接到的特定数据源的信息。根据设计，所有 CLI/ODBC 驱动程序必须支持三个特定的函数，这些函数提供关于驱动程序本身的功能以及该驱动程序底层数据源功能的信息。通过使用这组函数，应用程序可以确定特定数据 源的功能和限制，并相应地调整其行为。第一个函数是 SQLGetInfo() ，该函数用于获取关于数据源的各种特征的信息。第二个函数是 SQLGetFunctions()，该函数告诉应用程序数据源/驱动程序是否支持特定的 CLI/ODBC 函数。最后一个函数是 SQLGetTypeInfo()，该函数为应用程序提供关于数据源所使用的本地数据类型的信息。在这三个函数当中，SQLGetInfo() 的功能可能最为强大。这个函数可单独提供 165 条以上的信息。

　　SQLGetInfo()、SQLGetFunctions() 和 SQLGetTypeInfo() 函数所返回的信息是静态的 —— 也就是说，这三个函数返回的关于数据源/驱动程序的特征的信息不能通过调用应用程序来修改。不过，大多数数据源驱动程序还包括一些可以更改的信息，通过更 改这些信息可以改变驱动程序为特定应用程序提供的行为方式。这种信息被称为驱动程序属性，有三种类型的驱动程序属性可用：

　　环境属性

　　连接属性

　　SQL 语句属性

　　我们将在以下几小节中讨论每种属性。

　　环境属性

　　环境属性可以影响在指定环境下运行的 CLI/ODBC 函数的行为。在任何时候，应用程序都可以通过调用 SQLGetEnvAttr() 函数获得环境属性的值，还可以通过调用 SQLSetEnvAttr() 函数改变环境属性的值。一些较为常见的环境属性有：

　　SQL\_ATTR\_ODBC\_VERSION: 确定 CLI/ODBC 的行为表现具有 ODBC 2.0 特征还是 ODBC 3.x 特征。

　　SQL\_ATTR\_OUTPUT\_NTS: 决定在将字符串数据值返回到应用程序之前，驱动程序是否将一个 null 终止符附加在这个值的后面。

　　值得注意的是，只有当还未给该环境分配连接句柄的时候，才可以更改环境属性。一旦分配了连接句柄，该环境的属性值就只能被检索，不能被修改。

　　连接属性

　　连接属性可以影响到数据源和驱动程序的连接的行为。在任何时候，应用程序都可以通过调用 SQLGetConnectAttr() 函数获得连接属性的值，还可以通过调用 SQLSetConnectAttr() 函数更改连接属性的值。一些较为常用的连接属性有：

　　SQL\_ATTR\_AUTOCOMMIT: 决定数据源/驱动程序是以自动提交模式还是手动提交模式运行。

　　SQL\_ATTR\_MAXCONN: 决定应用程序每次可以打开的并发数据源连接的最大个数。（这个属性的默认值为 0，意即一个应用程序每次可以打开任意数量的连接。）

　 　SQL\_ATTR\_TXN\_ISOLATION: 指定用于当前连接的隔离级别（SQL\_TXN\_SERIALIZABLE 为可重复读，SQL\_TXN\_REPEATABLE\_READ 为读稳定性，SQL\_TXN\_READ\_COMMITTED 为游标稳定性，SQL\_TXN\_READ\_UNCOMMITTED 为未提交读）。

　　和环境属性一样，设置连接属性的时机很重要。有些连 接属性可以随时进行设置，而有些属性只能在分配了连接句柄之后、建立到数据源的连接之前进行设置。有些连接属性只能在建立了到数据源的连接之后，并且还不 存在与打开的连接相关联的未解决的事务或打开的游标的时候进行设置。

　　语句属性

　　语句属性可以影响很多 SQL 语句级 CLI/ODBC 函数的行为。 在任何时候，应用程序都可以通过调用 SQLGetStmtAttr() 函数获得语句属性的值，还可以通过调用 SQLSetStmtAttr() 函数更改语句属性的值。一些常用的语句属性有：

　　SQL\_ATTR\_CONCURRENCY: 指定使用的游标并发级别（只读、低级锁定或比较值锁定）。

　　SQL\_ATTR\_CURSOR\_TYPE: 指定在处理结果数据集时使用的游标的类型（forward-only、static、keyset-driven 或 dynamic）。

　　SQL\_ATTR\_RETRIEVE\_DATA: 指定 SQLFetch() 和 SQLFetchScroll() 函数在放置游标之后，是否自动检索数据。

　　同样，设置语句属性的时机很重要。有些语句属性必须在与语句句柄相关联的 SQL 语句执行之前进行设置；有些语句属性虽可以在任何时候设置，但是只有当与语句句柄相关联的 SQL 语句被再次使用时才会被应用，而有些属性则可以在任何时候进行设置。