Linux/Unix下ODBC的安装、配置与编程

ODBC原理

ODBC 是Open Database Connect 即开放数据库互连的简称，它是由Microsoft 公司于1991 年提出的一个用于访问数据库的统一界面标准，是应用程序和数据库系统之间的中间件。它通过使用相应应用平台上和所需数据库对应的驱动程序与应用程序的交互来实现对数据库的操作，避免了在应用程序中直接调用与数据库相关的操作，从而提供了数据库的独立性。

ODBC 主要由驱动程序和驱动程序管理器组成。驱动程序是一个用以支持ODBC 函数调用的模块，每个驱动程序对应于相应的数据库，当应用程序从基于一个数据库系统移植到另一个时，只需更改应用程序中由ODBC 管理程序设定的与相应数据库系统对应的别名即可。驱动程序管理器可链接到所有ODBC 应用程序中，它负责管理应用程序中ODBC 函数与DLL 中函数的绑定。

ODBC 使用层次的方法来管理数据库，在数据库通信结构的每一层，对可能出现依赖数据库产品自身特性的地方，ODBC 都引入一个公共接口以解决潜在的不一致性，从而很好地解决了基于数据库系统应用程序的相对独立性，这也是ODBC 一经推出就获得巨大成功的重要原因之一。

从结构上分，ODBC 分为单束式和多束式两类。

1. 单束式驱动程序   
   单束式驱动程序介于应用程序和数据库之间，像中介驱动程序一样数据提供一个统一的数据访问方式。当用户进行数据库操作时，应用程序传递一个ODBC 函数调用给ODBC 驱动程序管理器，由ODBC API 判断该调用是由它直接处理并将结果返回还是送交驱动程序执行并将结果返回。由上可见，单束式驱动程序本身是一个数据库引擎，由它直接可完成对数据库的操作，尽管该数据库可能位于网络的任何地方。
2. 多束式驱动程序   
   多束式驱动程序负责在数据库引擎和客户应用程序之间传送命令和数据，它本身并不执行数据处理操作而用于远程操作的网络通信协议的一个界面。前端应用程序提出对数据库处理的请求，该请求转给ODBC 驱动程序管理器，驱动程序管理器依据请求的情况，就地完成或传给多束驱动程序，多束式驱动程序将请求翻译为特定厂家的数据库通信接口（如Oracle 的SQLNet）所能理解的形式并交于接口去处理，接口把请求经网络传送给服务器上的数据引擎，服务器处理完后把结果发回给数据库通信接口，数据库接口将结果传给多束式ODBC 驱动程序，再由驱动程序将结果传给应用程序。

很多程序员已经体会到了在Windows平台下的ODBC的益处，而在Linux/Unix下进行数据库编程的时候却不得不根据不同的数据库来选择特有的API进行编程，一旦数据库发生了改变，所有与这些API相关的程序都必须进行修改。其实在Linux/Unix下现在也有了自己的ODBC，可以使我们的数据库编程就像在Windows平台下一样简单。

下面我们开始介绍Linux/Unix下的ODBC：

Linux/Unix下ODBC的安装

方法一：   
先下载最新的unixODBC源码包（ <http://www.unixodbc.org/unixODBC-2.2.1.tar.gz>）放到/usr/local下，然后运行下述命令：

|  |
| --- |
| tar zxvf unixODBC-2.2.1.tar.gz  cd unixODBC-2.2.1  ./configure --prefix=/usr/local/unixODBC-2.2.1 --includedir=/usr/include  --libdir=/usr/lib -bindir=/usr/bin --sysconfdir=/etc  make  make install |

安装成功后，unixODBC所需的头文件都被安装到了/usr/inlucde下，编译好的库文件安装到了/usr/lib下，与unixODBC相关的可执行文件安装到了/usr/bin下，配置文件放到了/etc下。

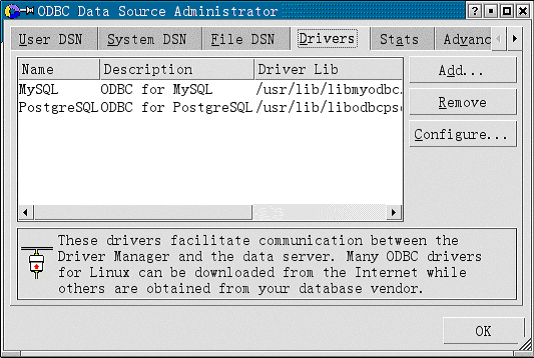
方法二：   
下载rpm包进行安装，我们这里以Red Hat 7.3为例：   
unixODBC-2.2.0-5 RPM for i386（安装包及源码包）   
（ <ftp://speakeasy.rpmfind.net/linux/redhat/7.3/en/os/i386/RedHat/RPMS/unixODBC-2.2.0-5.i386.rpm>、 <ftp://ftp.rpmfind.net/linux/redhat/7.3/en/os/i386/SRPMS/unixODBC-2.2.0-5.src.rpm>）   
unixODBC-devel-2.2.0-5 RPM for i386   
（ <ftp://speakeasy.rpmfind.net/linux/redhat/7.3/en/os/i386/RedHat/RPMS/unixODBC-devel-2.2.0-5.i386.rpm>）   
直接将unixODBC-2.2.0-5.i386.rpm和unixODBC-devel-2.2.0-5.i386.rpm装入系统就可以了，命令如下：

|  |
| --- |
| rpm -ivh unixODBC-2.2.0-5.i386.rpm  rpm -ivh unixODBC-devel-2.2.0-5.i386.rpm |

安装好以后，所需的各个部分与上面所列的位置相同。

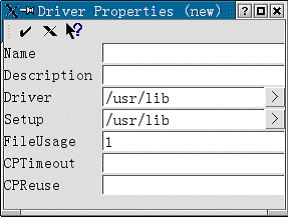
Linux/Unix下ODBC的配置

运行ODBCConfig程序（在/usr/bin下），如下图：

**图一：ODBCConfig主窗口**  


和Windows下的ODBC设置窗口是不是很像？我想大家都能看懂吧。

第一步：安装数据库的ODBC驱动程序   
Drivers这一栏中用来设置数据库的驱动程序，点击Add按钮，会出现下图：

**图二：ODBCConfig Driver Properties窗口**  


Name一栏填入数据库驱动的名称，Description是数据库驱动的描述，Driver是用来选择数据库驱动程序的，Setup是用来选择数据库驱动安装程序的，如果你是按照上述安装方法安装的，这些程序都放在/usr/lib下，下面是数据库驱动程序的列表：

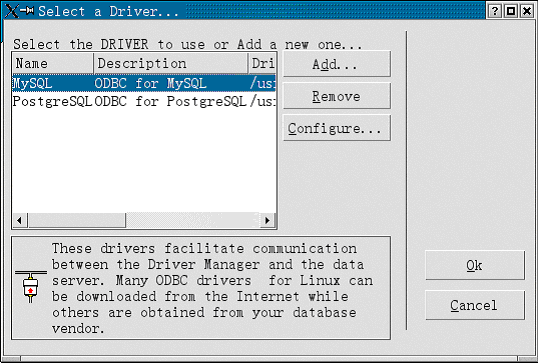
|  |  |  |
| --- | --- | --- |
| 数据库 | 数据库驱动程序 | 数据库驱动安装程序 |
| TXT | libodbctxt.so | libodbctxtS.so |
| NNTP | libnn.so | libodbcnnS.so |
| MiniSQL | libodbcmini.so | libodbcminiS.so |
| PostgreSQL | libodbcpsql.so | libodbcpsqlS.so |
| MySQL | （注释） | libodbcmyS.so |
| Sybase/MS SQL | （注释） | libtdsS.so |
| Oracle | （注释） | liboraodbcS.so |

注释：   
MySQL、Sybase/MS SQL和Oracle的数据库驱动可以在下列网址找到：   
MySQL　　　　　　　　　　 <http://www.unixodbc.org/myodbc.html>   
Sybase/MS SQL　　　　　　 <http://www.freetds.org>   
Oracle　　　　　　　　　　 <http://www.easysoft.org>

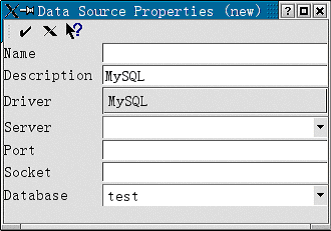
MySQL的驱动程序MyODBC-2.50.39-4 RPM for i386以及源码包：   
<ftp://speakeasy.rpmfind.net/linux/redhat/7.3/en/os/i386/RedHat/RPMS/MyODBC-2.50.39-4.i386.rpm>   
<ftp://ftp.redhat.com/pub/redhat/linux/7.3/en/os/i386/SRPMS/MyODBC-2.50.39-4.src.rpm>

选择好驱动程序之后，点击"√"保存退出。

第二步：设置DSN   
DSN分为User DSN、System DSN和File DSN三种，我们以System DSN为例。选中System DSN一栏以后，点击Add…按钮就会见到下图：

**图三：创建DSN选择数据库驱动**  


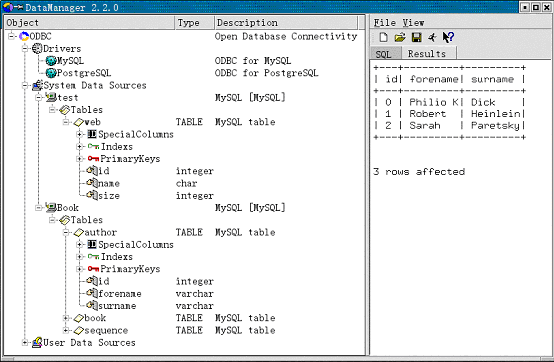
列表中会列出你已经安装好的数据库驱动程序，我这里只装了MySQL和PostgreSQL，然后选择你所要使用的驱动程序，然后点击OK就会出现下图：

**图四：DSN的设置**  


我这里使用的是MySQL的数据库驱动，不同的数据库，这个窗口的内容会有所不同。Name是数据源的名称，Description是描述，Server可以选择服务器，如果本机启动了MySQL就可以选择localhost，如果Port和Socket有特殊要求，再根据实际情况进行修改，Database是用来选择数据库的，下拉菜单不一定包含所有的数据库，你可以把自己已经创建好的数据库名称填写在这里。都配置好之后，点击"√"保存退出。

这样Linux/Unix下的ODBC数据源就已经设置好了，大家还可以在ODBCConfig程序的Status栏中查看ODBC的使用情况，在Advanced栏中设置是否做日志或者启动连接池，在About栏中，有一个Linux/Unix ODBC的示意图，在Credits按钮中可以看到所有开发者的名字的列表。 ODBCConfig程序中所有有关数据库驱动程序的信息被放在odbcinst.ini（在/etc下）文件中，有关DSN的信息被放在odbc.ini（在/etc下）文件中，大家有兴趣的话，可以自己去观察一下。

第三步：使用DataManager程序浏览数据库   
运行DataManager程序之后就可以查看Drivers、System DSN和User DSN这几项内容，，在浏览数据库的时候，可以在右面的SQL栏中输入SQL语句，然后点击人形按钮就可以运行SQL语句，运行结果会在Results一栏中显示出来，具体情况可以见下图：

**图五：使用DataManager浏览数据库**  


第四步：使用isql程序查看数据库   
unixODBC还提供了命令台下查看数据库的程序，这就是isql，用法如下：

isql DSN [UID [PWD]] [options]

DSN 数据源名称

UID 用户ID

PWD 用户密码

Options：

-b 批处理，没有提示符的模式

-dx 设置列之间的分隔符为x

-w 将查询结果输出为HTML格式

-c 第一行输出列名

--version 输出isql的版本号

Linux/Unix下ODBC的编程

**1、使用unixODBC提供的ODBC API进行编程：**   
在进行编程之前，我们来看一下ODBC API中的常用数据类型与我们在C语言中使用的数据类型的对应关系：

|  |  |  |
| --- | --- | --- |
| **类型标识符** | **ODBC数据类型** | **C数据类型** |
| SQL\_C\_CHAR | SQLCHAR \* | unsigned char \* |
| SQL\_C\_SSHORT | SQLSMALLINT | short int |
| SQL\_C\_USHORT | SQLUSMALLINT | unsigned short int |
| SQL\_C\_SLONG | SQLINTEGER | long int |
| SQL\_C\_FLOAT | SQLREAL | float |
| SQL\_C\_DOUBLE | SQLDOUBLE, SQLFLOAT | double |
| SQL\_C\_BINARY | SQLCHAR \* | unsigned char \* |
| SQL\_C\_TYPE\_DATE | SQL\_DATE\_STRUCT | struct tagDATE\_STRUCT {SQLSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; } DATE\_STRUCT; |
| SQL\_C\_TYPE\_TIME | SQL\_TIME\_STRUCT | struct tagTIME\_STRUCT {SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; } TIME\_STRUCT; |

我们这里使用的数据库名称为test（DSN），这个DSN使用的用户名是root，密码为空，表的名称是web，字段情况如下：

|  |  |
| --- | --- |
| **字段名** | **数据类型** |
| id | integer |
| name | char(40) |
| size | integer |

第一：设定ODBC环境句柄并设置参数   
首先我们需要声明一个ODBC环境句柄（SQLHENV），它可以用来获得有关的ODBC环境信息，我们需要调用SQLAllocHandle ( SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &V\_OD\_Env )来获得这个句柄，V\_OD\_Env就是要分配的SQLHENV类型的环境句柄。分配好句柄之后，你给它需要设定所使用的ODBC版本，你可以调用SQLSetEnvAttr ( V\_OD\_Env, SQL\_ATTR\_ODBC\_VERSION, (void\*)SQL\_OV\_ODBC3, 0 )，SQL\_ATTR\_ODBC\_VERSION是存放你定义的ODBC版本号的变量，SQL\_OV\_ODBC3则说明你的程序使用的是ODBC 3.0。

第二：设定连接句柄并设置超时参数   
我们需要声明一个连接句柄（SQLHDBC），用来存放数据库连接信息的，调用SQLAllocHandle ( SQL\_HANDLE\_DBC, V\_OD\_Env, &V\_OD\_hdbc )获得连接句柄，V\_OD\_hdbc就是要分配的SQLHDBC类型的连接句柄。分配好之后，我们可以调用SQLSetConnectAttr ( V\_OD\_hdbc, SQL\_LOGIN\_TIMEOUT, (SQLPOINTER \*)5, 0 )来设定连接超时参数。

第三：连接数据库   
调用SQLConnect ( V\_OD\_hdbc, (SQLCHAR\*) "Test", SQL\_NTS, (SQLCHAR\*) "root", SQL\_NTS, (SQLCHAR\*) "", SQL\_NTS )连接我前面提到的数据库，需要设定三个参数，就是数据库名称、用户名和密码（因为我的数据库密码为空，所以这里的密码也为空），后面的SQL\_NTS的位置应该写入这些参数的长度，如果写的是SQL\_NTS就是让SQLConnect来决定参数的长度。

第四：分配SQL语句的句柄并进行查询：   
需要声明一个SQL语句的句柄（SQLHSTMT），用来存放SQL语句信息的，调用SQLAllocHandle ( SQL\_HANDLE\_STMT, V\_OD\_hdbc, &V\_OD\_hstmt )来获得这个句柄，V\_OD\_hstmt就是我们要分配的SQLHSTMT类型的SQL语句句柄。

我们的查询语句是：

|  |
| --- |
| SELECT name, id FROM web ORDER BY id |

执行这条查询语句之后，查询结果可能有很多行，但每行只有两列，分别对应name和id，它们的数据类型为integer和char\*，在ODBC中的数据类型标识符为SQL\_C\_ULONG和SQL\_C\_CHAR。我们需要先声明这样的两个变量来存贮查询结果：

|  |
| --- |
| SQLINTEGER V\_OD\_id;  char V\_OD\_buffer[200]; |

然后我们需要使用SQLBindCol函数把查询结果和我们定义的变量进行绑定：

|  |
| --- |
| SQLBindCol(V\_OD\_hstmt,1,SQL\_C\_CHAR, &V\_OD\_buffer,150,&V\_OD\_err);  SQLBindCol(V\_OD\_hstmt,2,SQL\_C\_ULONG,&V\_OD\_id,150,&V\_OD\_err); |

这里的V\_OD\_err是用来存放错误信息编号的变量，类型也是SQLINTEGER。   
接下来，我们调用SQLExecDirect来进行查询：

|  |
| --- |
| SQLExecDirect ( V\_OD\_hstmt, "SELECT dtname,iduser FROM web order by iduser", SQL\_NTS ); |

我们可以用SQLNumResultCols ( V\_OD\_hstmt, &V\_OD\_colanz )来获得结果的列数，也可以用SQLRowCount( V\_OD\_hstmt, &V\_OD\_rowanz )来获得结果的条数，V\_OD\_colanz和V\_OD\_rowanz分别存储相应的结果，类型分别为SQLSMALLINT和SQLINTEGER。   
在读取结果之前，我们需要调用SQLFetch ( V\_OD\_hstmt )语句，这个语句可以用来获得第一条结果也可以用来都下一条，有点像next的感觉。然后我们就可以在V\_OD\_id和V\_OD\_buffer里面获得每条记录的结果了。

第五：关于关闭连接和释放句柄   
关闭数据库的连接，调用SQLDisconnect ( V\_OD\_hdbc )就可以了，但在关闭数据库之前需要先释放SQL语句的句柄，而且在关闭数据库之后应该释放连接句柄和ODBC环境句柄，语句如下（按正常的顺序）：

|  |
| --- |
| SQLFreeHandle(SQL\_HANDLE\_STMT,V\_OD\_hstmt);  SQLDisconnect(V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_DBC,V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env); |

第六：关于上述情况中的错误信息处理   
我们需要定义两个变量：

|  |
| --- |
| long V\_OD\_erg;  SQLINTEGER V\_OD\_err; |

SQLAllocHandle、SQLSetEnvAttr、SQLSetConnectAttr、SQLConnect、SQLExecDirect、SQLNumResultCols和SQLRowCount的调用结果都可以用V\_OD\_erg来存储，V\_OD\_err可以获得SQLBindCol中的错误信息。

第七：获得本机的DSN信息   
我们可以在声明SQLHENV句柄之后，使用SQLDataSources函数来获得本机的DSN信息。程序如下：

|  |
| --- |
| void OD\_ListDSN(void)  {  char l\_dsn[100],l\_desc[100];  short int l\_len1,l\_len2,l\_next;  l\_next=SQL\_FETCH\_FIRST;  while( SQLDataSources(V\_OD\_Env,l\_next,l\_dsn, sizeof(l\_dsn),  &l\_len1, l\_desc, sizeof(l\_desc), &l\_len2) == SQL\_SUCCESS)  {  printf("Server=(%s) Beschreibung=(%s)\n",l\_dsn,l\_desc);  l\_next=SQL\_FETCH\_NEXT;  }  } |

l\_next变量是用来指定我们所要获得的DSN的类别：

|  |  |
| --- | --- |
| SQL\_FETCH\_FIRST | 设定SQLDataSources()函数找到第一个可用的数据源（可以是User DSN，也可以是Systerm DSN） |
| SQL\_FETCH\_FIRST\_USER | 设定SQLDataSources()函数找到第一个User DSN |
| SQL\_FETCH\_FIRST\_SYSTEM | 设定SQLDataSources()函数找到第一个System DSN |
| SQL\_FETCH\_NEXT | 找到下一个数据源，至于数据源类型则要根据前面的定义 |

到这里，我们在Unix的C语言下面进行ODBC编程已经讲完，上述ODBC API需要引用以下几个头文件（这些文件已经安装到/usr/include下了）：

|  |
| --- |
| #include <sql.h>  #include <sqlext.h>  #include <sqltypes.h> |

另外如果大家使用GTK进行编程，由于到目前为止GTK还没有加入专门处理数据库的部件，所以大家可以在GTK中调用上述的ODBC API即可。

这里附上例程供大家参考学习：

|  |
| --- |
| /\* odbc.c  testing unixODBC  \*/  #include <stdlib.h>  #include <stdio.h>  #include <odbc/sql.h>  #include <odbc/sqlext.h>  #include <odbc/sqltypes.h>  SQLHENV V\_OD\_Env; // Handle ODBC environment  long V\_OD\_erg; // result of functions  SQLHDBC V\_OD\_hdbc; // Handle connection  char V\_OD\_stat[10]; // Status SQL  SQLINTEGER V\_OD\_err,V\_OD\_rowanz,V\_OD\_id;  SQLSMALLINT V\_OD\_mlen,V\_OD\_colanz;  char V\_OD\_msg[200],V\_OD\_buffer[200];  int main(int argc,char \*argv[])  {  // 1. allocate Environment handle and register version  V\_OD\_erg=SQLAllocHandle(SQL\_HANDLE\_ENV,SQL\_NULL\_HANDLE,&V\_OD\_Env);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Error AllocHandle\n");  exit(0);  }  V\_OD\_erg=SQLSetEnvAttr(V\_OD\_Env, SQL\_ATTR\_ODBC\_VERSION, (void\*)SQL\_OV\_ODBC3, 0);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Error SetEnv\n");  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  // 2. allocate connection handle, set timeout  V\_OD\_erg = SQLAllocHandle(SQL\_HANDLE\_DBC, V\_OD\_Env, &V\_OD\_hdbc);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Error AllocHDB %d\n",V\_OD\_erg);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  SQLSetConnectAttr(V\_OD\_hdbc, SQL\_LOGIN\_TIMEOUT, (SQLPOINTER \*)5, 0);  // 3. Connect to the datasource "web"  V\_OD\_erg = SQLConnect(V\_OD\_hdbc, (SQLCHAR\*) "Test", SQL\_NTS,  (SQLCHAR\*) "root", SQL\_NTS,  (SQLCHAR\*) "", SQL\_NTS);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Error SQLConnect %d\n",V\_OD\_erg);  SQLGetDiagRec(SQL\_HANDLE\_DBC, V\_OD\_hdbc,1,  V\_OD\_stat, &V\_OD\_err,V\_OD\_msg,100,&V\_OD\_mlen);  printf("%s (%d)\n",V\_OD\_msg,V\_OD\_err);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  printf("Connected !\n");  V\_OD\_erg=SQLAllocHandle(SQL\_HANDLE\_STMT, V\_OD\_hdbc, &V\_OD\_hstmt);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Fehler im AllocStatement %d\n",V\_OD\_erg);  SQLGetDiagRec(SQL\_HANDLE\_DBC, V\_OD\_hdbc,1, V\_OD\_stat,&V\_OD\_err,V\_OD\_msg,100,&V\_OD\_mlen);  printf("%s (%d)\n",V\_OD\_msg,V\_OD\_err);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  SQLBindCol(V\_OD\_hstmt,1,SQL\_C\_CHAR, &V\_OD\_buffer,150,&V\_OD\_err);  SQLBindCol(V\_OD\_hstmt,2,SQL\_C\_ULONG,&V\_OD\_id,150,&V\_OD\_err);    V\_OD\_erg=SQLExecDirect(V\_OD\_hstmt,"SELECT dtname,iduser FROM web order by iduser",SQL\_NTS);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Error in Select %d\n",V\_OD\_erg);  SQLGetDiagRec(SQL\_HANDLE\_DBC, V\_OD\_hdbc,1, V\_OD\_stat,&V\_OD\_err,V\_OD\_msg,100,&V\_OD\_mlen);  printf("%s (%d)\n",V\_OD\_msg,V\_OD\_err);  SQLFreeHandle(SQL\_HANDLE\_STMT,V\_OD\_hstmt);  SQLFreeHandle(SQL\_HANDLE\_DBC,V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  V\_OD\_erg=SQLNumResultCols(V\_OD\_hstmt,&V\_OD\_colanz);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  SQLFreeHandle(SQL\_HANDLE\_STMT,V\_OD\_hstmt);  SQLDisconnect(V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_DBC,V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  printf("Number of Columns %d\n",V\_OD\_colanz);  V\_OD\_erg=SQLRowCount(V\_OD\_hstmt,&V\_OD\_rowanz);  if ((V\_OD\_erg != SQL\_SUCCESS) && (V\_OD\_erg != SQL\_SUCCESS\_WITH\_INFO))  {  printf("Number of RowCount %d\n",V\_OD\_erg);  SQLFreeHandle(SQL\_HANDLE\_STMT,V\_OD\_hstmt);  SQLDisconnect(V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_DBC,V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  exit(0);  }  printf("Number of Rows %d\n",V\_OD\_rowanz);  V\_OD\_erg=SQLFetch(V\_OD\_hstmt);  while(V\_OD\_erg != SQL\_NO\_DATA)  {  printf("Result: %d %s\n",V\_OD\_id,V\_OD\_buffer);  V\_OD\_erg=SQLFetch(V\_OD\_hstmt);  } ;  SQLFreeHandle(SQL\_HANDLE\_STMT,V\_OD\_hstmt);  SQLDisconnect(V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_DBC,V\_OD\_hdbc);  SQLFreeHandle(SQL\_HANDLE\_ENV, V\_OD\_Env);  return(0);  } |

**2．QT下进行ODBC编程**   
QT 3.0提供了Data Table、Data Browser和Data View三个与数据库相关的控件。你可以在QT的Project设置你要连接的数据库，Driver一栏中选择QODBC3即可，其它选项你一看就明白了。上述的三个数据库控件的使用方法可以参见QT中相应文档，也很好使用的。

参考资料

1. 微软的ODBC主页： <http://www.microsoft.com/data/odbc/>；
2. UnixODBC的主页： <http://www.unixodbc.org>；
3. FreeODBC的主页： <http://www.jepstone.net/FreeODBC/>；
4. EasySoft的主页： <http://www.easysoft.com>；
5. TrollTech的QT 3.0文档主页： <http://doc.trolltech.com/3.0/>。