\_RecordsetPtr的open与\_CommandPtr 的Execute区别

为了取得结果记录集，我们定义一个指向Recordset对象的指针:\_RecordsetPtr m\_pRecordset;

并为其创建Recordset对象的实例: m\_pRecordset.CreateInstance("ADODB.Recordset");

SQL命令的执行可以采用多种形式，下面我们一进行阐述。

(1)利用Connection对象的Execute方法执行SQL命令

Execute方法的原型如下所示:

\_RecordsetPtr Connection15::Execute ( \_bstr\_t CommandText, VARIANT \* RecordsAffected, long Options ) 其中CommandText是命令字串，通常是SQL命令。参数RecordsAffected是操作完成后所影响的行数, 参数Options表示CommandText中内容的类型，Options可以取如下值之一：

adCmdText:表明CommandText是文本命令

adCmdTable:表明CommandText是一个表名

adCmdProc:表明CommandText是一个存储过程

adCmdUnknown:未知

Execute执行完后返回一个指向记录集的指针，下面我们给出具体代码并作说明。 \_variant\_t RecordsAffected;

//执行SQL命令：CREATE TABLE创建表格users,users包含四个字段:整形ID,字符串username,整形old,日期型birthday

m\_pConnection->Execute("CREATE TABLE users(ID INTEGER,username TEXT,old INTEGER,birthday DATETIME)",&RecordsAffected,adCmdText);

//往表格里面添加记录

m\_pConnection->Execute("INSERT INTO users(ID,username,old,birthday) valueS (1, null,25, null)",&RecordsAffected,adCmdText);

//将所有记录old字段的值加一

m\_pConnection->Execute("UPDATE users SET old = old+1",&RecordsAffected,adCmdText);

//执行SQL统计命令得到包含记录条数的记录集

m\_pRecordset = m\_pConnection->Execute("SELECT COUNT(\*) FROM users",&RecordsAffected,adCmdText);

\_variant\_t vIndex = (long)0;

\_variant\_t vCount = m\_pRecordset->GetCollect(vIndex);//取得第一个字段的值放入vCount变量

m\_pRecordset->Close();///关闭记录集

CString message;

message.Format("共有%d条记录",vCount.lVal);

AfxMessageBox(message);///显示当前记录条数

(2)利用Command对象来执行SQL命令

\_CommandPtr m\_pCommand;

m\_pCommand.CreateInstance("ADODB.Command");

\_variant\_t vNULL;

vNULL.vt = VT\_ERROR;

vNULL.scode = DISP\_E\_PARAMNOTFOUND;//定义为无参数

m\_pCommand->ActiveConnection = m\_pConnection;//非常关键的一句，将建立的连接赋值给它

m\_pCommand->CommandText = "SELECT \* FROM users";//命令字串

m\_pRecordset = m\_pCommand->Execute(&vNULL,&vNULL,adCmdText);//执行命令，取得记录集

在这段代码中我们只是用Command对象来执行了SELECT查询语句，Command对象在进行存储过程的调用中能真正体现它的作用。下次我们将详细介绍。

ADO库包含三个基本接口:\_ConnectionPtr接口、\_CommandPtr接口和\_RecordsetPtr接口。

\_ConnectionPtr接口返回一个记录集或一个空指针。

通常使用它来创建一个数据连接或执行一条不返回任何结果的SQL语句，如一个存储过程。

使用\_ConnectionPtr接口返回一个记录集不是一个好的使用方法。

通常同Cdatabase一样，使用它创建一个数据连接，然后使用其它对象执行数据输入输出操作。

\_CommandPtr接口返回一个记录集。

它提供了一种简单的方法来执行返回记录集的存储过程和SQL语句。

在使用\_CommandPtr接口时，你可以利用全局\_ConnectionPtr接口，也可以在\_CommandPtr接口里直接使用连接串。

如果你只执行一次或几次数据访问操作，后者是比较好的选择。

但如果你要频繁访问数据库，并要返回很多记录集，那么，你应该使用全局\_ConnectionPtr接口创建一个数据连接，

然后使用\_CommandPtr接口执行存储过程和SQL语句。

\_RecordsetPtr是一个记录集对象。

与以上两种对象相比，它对记录集提供了更多的控制功能，如记录锁定，游标控制等。同\_CommandPtr接口一样，它不一定要使用一个已经创建的数据连接，可以用一个连接串代替连接指针赋给\_RecordsetPtr的connection成员变量，让它自己创建数据连接。

如果你要使用多个记录集，最好的方法是同Command对象一样使用已经创建了数据连接的全局\_ConnectionPtr接口，然后使用\_RecordsetPtr执行存储过程和SQL语句。

基本流程

(1)初始化COM库，引入ADO库定义文件

(2)用Connection对象连接数据库

(3)利用建立好的连接，通过Connection、Command对象执行SQL命令，或利用Recordset对象取得结果记录集进行查询、处理。

(4)使用完毕后关闭连接释放对象。

COM库的初始化

我们可以使用AfxOleInit()来初始化COM库，这项工作通常在CWinApp::InitInstance()的重载函数中完成，请看如下代码:

BOOL CADOTest1App::InitInstance()

{

AfxOleInit();

或

可以调用CoInitialize(0)来初始化;

用CoUninitialize()释放;

......

}

用#import指令引入ADO类型库

我们在stdafx.h中加入如下语句：

#import "c:\program files\common files\system\ado\msado15.dll" no\_namespace rename("EOF","adoEOF")

其最终作用同我们熟悉的#include类似,编译的时候系统会为我们生成msado15.tlh,ado15.tli两个C++头文件来定义ADO库。

创建Connection对象并连接数据库

首先我们需要添加一个指向Connection对象的指针:

\_ConnectionPtr m\_pConnection;

BOOL CADOTest1Dlg::OnInitDialog()

{

CDialog::OnInitDialog();

try

{

HRESULT hr = m\_pConnection.CreateInstance("ADODB.Connection");//创建Connection对象

if(SUCCEEDED(hr))

{

hr = m\_pConnection->Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb","","",adModeUnknown);//连接数据库

//上面一句中连接字串中的Provider是针对ACCESS2000环境的，对于ACCESS97,需要改为:Provider=Microsoft.Jet.OLEDB.3.51; }

}

}

catch (\_com\_error e) //COM错误取得,当执行COM功能的时候，如果出错，可以捕捉到\_com\_error的异常

{

CString strComError;

strComError.Format("错误编号: %08lx\n错误信息: %s\n错误源: %s\n错误描述: %s",

e.Error(), // 错误编号

e.ErrorMessage(), // 错误信息

(LPCSTR) e.Source(), // 错误源

(LPCSTR) e.Description()); // 错误描述

::MessageBox(NULL,strComError,"错误",MB\_ICONEXCLAMATION);

}

}

也可以使用UDL文件进行连接。

try

{

m\_pConnection.CreateInstance(\_\_uuidof(Connection));

m\_pConnection->ConnectionString ="File Name=e.udl";

m\_pConnection->Open("","","",NULL);

}

catch(\_com\_error e)

{....}

◆在这段代码中我们是通过Connection对象的Open方法来进行连接数据库的，下面是该方法的原型

HRESULT Connection15::Open (\_bstr\_t ConnectionString, \_bstr\_t UserID, \_bstr\_t Password, long Options )

ConnectionString 为连接字串,

UserID 是用户名,

Password 是登陆密码,

Options 是连接选项,用于指定Connection对象对数据的更新许可权,

Options可以是如下几个常量:

adModeUnknown: 缺省。当前的许可权未设置

adModeRead: 只读

adModeWrite: 只写

adModeReadWrite: 可以读写

adModeShareDenyRead: 阻止其它Connection对象以读权限打开连接

adModeShareDenyWrite: 阻止其它Connection对象以写权限打开连接

adModeShareExclusive: 阻止其它Connection对象打开连接

adModeShareDenyNone: 允许其它程序或对象以任何权限建立连接

◆常用的数据库连接方法：

(1)通过JET数据库引擎对ACCESS2000数据库的连接

m\_pConnection->Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\test.mdb","","",adModeUnknown);

(2)通过DSN数据源对任何支持ODBC的数据库进行连接:

m\_pConnection->Open("Data Source=adotest;UID=sa;PWD=;","","",adModeUnknown);

//m\_pConnection->Open("DSN=test;","","",0); //连接叫作test的ODBC数据源

(3)不通过DSN对SQL SERVER数据库进行连接：

m\_pConnection->Open("driver={SQL Server};Server=127.0.0.1;DATABASE=vckbase;UID=sa;PWD=139","","",adModeUnknown);

其中Server是SQL服务器的名称，DATABASE是库的名称

◆先介绍Connection对象中两个有用的属性ConnectionTimeOut与State

ConnectionTimeOut用来设置连接的超时时间，需要在Open之前调用，例如:

m\_pConnection->ConnectionTimeout = 5; //设置超时时间为5秒

m\_pConnection->Open("Data Source=adotest;","","",adModeUnknown);

State属性指明当前Connection对象的状态，0表示关闭，1表示已经打开，我们可以通过

读取这个属性来作相应的处理，例如:

if(m\_pConnection->State)

m\_pConnection->Close(); //如果已经打开了连接则关闭它

执行SQL命令并取得结果记录集

为了取得结果记录集，我们定义一个指向Recordset对象的指针:

\_RecordsetPtr m\_pRecordset;

并为其创建Recordset对象的实例:

m\_pRecordset.CreateInstance("ADODB.Recordset");

SQL命令的执行可以采用多种形式，下面我们一进行阐述。

◆(1)利用Connection对象的Execute方法执行SQL命令

Execute方法的原型如下所示:

\_RecordsetPtr Connection15::Execute (\_bstr\_t CommandText, VARIANT \* RecordsAffected, long Options )

其中

CommandText 是命令字串，通常是SQL命令。

RecordsAffected 是操作完成后所影响的行数,

Options 表示CommandText中内容的类型，Options可以取如下值之一：

adCmdText: 表明CommandText是文本命令

adCmdTable: 表明CommandText是一个表名

adCmdProc: 表明CommandText是一个存储过程

adCmdUnknown: 未知

Execute执行完后返回一个指向记录集的指针，下面我们给出具体代码并作说明。

try

{

\_variant\_t ra;

m\_pConnection->Execute("CREATE TABLE student(Sno INTEGER,Sname TEXT,Ssex TEXT,Sage INTEGER,Sdept TEXT)",&ra,adCmdText);

m\_pConnection->Execute("INSERT INTO 学生信息(学号,姓名,年龄,生日) VALUES (112105, '程红秀',22,'1982-08-16')",&ra,adCmdText);//往表格里面添加记录

m\_pRecordset = m\_pConnection->Execute("SELECT COUNT(\*) FROM 学生信息",&ra,adCmdText); //执行SQL统计命令得到包含记录条数的记录集

\_variant\_t vCount = m\_pRecordset->GetCollect((\_variant\_t)(long)(0)); //取得第一个字段的值放入vCount变量

m\_pRecordset->Close();

CString message;

message.Format("共有%d条记录",vCount.lVal);

AfxMessageBox(message);

}

catch (\_com\_error e)

{ ...}

◆(2)利用 Command对象 来执行SQL命令

try

{

\_CommandPtr m\_pCommand;

m\_pCommand.CreateInstance("ADODB.Command");

m\_pCommand->ActiveConnection = m\_pConnection; //关键的一句，将建立的连接赋值给它

m\_pCommand->CommandText="INSERT INTO 学生信息(学号,姓名,年龄,生日) VALUES (112105, '程红秀',22,'1982-08-16')";

m\_pCommand->Execute(NULL,NULL,adCmdText);

m\_pCommand->CommandText="SELECT COUNT(\*) FROM 学生信息";

m\_pRecordset=m\_pCommand->Execute(NULL,NULL,adCmdText);

\_variant\_t vCount = m\_pRecordset->GetCollect((\_variant\_t)(long)0); //取得第一个字段的值

CString str;

str.Format("共有%d条记录",vCount.lVal);

AfxMessageBox(str);

m\_pRecordset->Close();

}

catch (\_com\_error e) {...}

在这段代码中我们只是用Command对象来执行了SELECT查询语句，

Command对象在进行存储过程的调用中能真正体现它的作用。下次我们将详细介绍。

◆(3)直接用Recordset对象进行查询取得记录集

例如

m\_pRecordset->Open("SELECT \* FROM 学生信息",\_variant\_t((IDispatch \*)m\_pConnection,true),adOpenStatic,adLockOptimistic,adCmdText);

Open方法的原型是这样的:

HRESULT Recordset15::Open ( const \_variant\_t & Source, const \_variant\_t & ActiveConnection, enum CursorTypeEnum CursorType, enum LockTypeEnum LockType, long Options )

其中：

①Source是数据查询字符串

②ActiveConnection是已经建立好的连接（我们需要用Connection对象指针来构造一个\_variant\_t对象)

③CursorType光标类型，它可以是以下值之一,请看这个枚举结构:

enum CursorTypeEnum

{

adOpenUnspecified = -1, //不作特别指定

adOpenForwardOnly = 0, //前滚静态光标。这种光标只能向前浏览记录集，比如用MoveNext向前滚动,这种方式可以提高浏览速度。但诸如BookMark,RecordCount,AbsolutePosition,AbsolutePage都不能使用

adOpenKeyset = 1, //采用这种光标的记录集看不到其它用户的新增、删除操作，但对于更新原有记录的操作对你是可见的。

adOpenDynamic = 2, //动态光标。所有数据库的操作都会立即在各用户记录集上反应出来。

adOpenStatic = 3 //静态光标。它为你的记录集产生一个静态备份，但其它用户的新增、删除、更新操作对你的记录集来说是不可见的。

};

④LockType锁定类型，它可以是以下值之一，请看如下枚举结构：

enum LockTypeEnum

{

adLockUnspecified = -1, //未指定

adLockReadOnly = 1, //只读记录集

adLockPessimistic = 2, //悲观锁定方式。数据在更新时锁定其它所有动作，这是最安全的锁定机制

adLockOptimistic = 3, //乐观锁定方式。只有在你调用Update方法时才锁定记录。在此之前仍然可以做数据的更新、插入、删除等动作

adLockBatchOptimistic = 4， //乐观分批更新。编辑时记录不会锁定，更改、插入及删除是在批处理模式下完成。

};

⑤Options请参考本文中对Connection对象的Execute方法的介绍

记录集的遍历、更新

根据我们刚才通过执行SQL命令建立好的 学生信息 表，它包含四个字段:学号，姓名，年龄，生日

以下的代码实现：打开记录集，遍历所有记录，删除第一条记录，添加三条记录，移动光标到第二条记录，更改其年龄，保存到数据库。

try

{

\_variant\_t vUsername,vBirthday,vID,vOld;

\_RecordsetPtr m\_pRecordset;

m\_pRecordset.CreateInstance("ADODB.Recordset");

m\_pRecordset->Open("SELECT \* FROM 学生信息",\_variant\_t((IDispatch\*)m\_pConnection,true),adOpenStatic,adLockOptimistic,adCmdText);

while(!m\_pRecordset->adoEOF)

{

vID = m\_pRecordset->GetCollect(\_variant\_t((long)0)); //取得第1列的值,从0开始计数，你也可以直接给出列的名称，如下一行

vUsername = m\_pRecordset->GetCollect("姓名"); //取得姓名字段的值

vOld = m\_pRecordset->GetCollect("年龄");

vBirthday = m\_pRecordset->GetCollect("生日");

TRACE("id:%d,姓名:%s,年龄:%d,生日:%s\r\n",

vID.lVal,

(LPCTSTR)(\_bstr\_t)vUsername,

vOld.lVal,

(LPCTSTR)(\_bstr\_t)vBirthday); //在DEBUG方式下的OUTPUT窗口输出记录集中的记录

m\_pRecordset->MoveNext(); //移到下一条记录

}

m\_pRecordset->MoveFirst(); //移到首条记录

m\_pRecordset->Delete(adAffectCurrent); //删除当前记录

for(int i=0;i<3;i++) //添加三条新记录并赋值

{

m\_pRecordset->AddNew(); //添加新记录

m\_pRecordset->PutCollect("学号",\_variant\_t((long)(i+10)));

m\_pRecordset->PutCollect("姓名",\_variant\_t("王斌年"));

m\_pRecordset->PutCollect("年龄",\_variant\_t((long)21));

m\_pRecordset->PutCollect("生日",\_variant\_t("1930-3-15"));

}

m\_pRecordset->Move(1,\_variant\_t((long)adBookmarkFirst)); //从第一条记录往下移动一条记录,即移动到第二条记录处

m\_pRecordset->PutCollect(\_variant\_t("年龄"),\_variant\_t((long)45)); //修改其年龄

m\_pRecordset->Update(); //保存到库中

} catch (\_com\_error e){}

关闭记录集与连接

记录集或连接都可以用Close方法来关闭

m\_pRecordset->Close(); //关闭记录集

m\_pConnection->Close(); //关闭连接

在stdafx.h中进行宏定义:

#if !defined CATCH\_ERROR

#define CATCH\_ERROR \

{ \

CString strComError; \

strComError.Format("错误编号: %08lx\n错误信息: %s\n错误源: %s\n错误描述: %s", \

e.Error(), \

e.ErrorMessage(), \

(LPCSTR) e.Source(), \

(LPCSTR) e.Description()); \

::MessageBox(NULL,strComError,"错误",MB\_ICONEXCLAMATION); \

}

#endif

使用方法：

try

{ ...}

catch(\_com\_error e)

{

CATCH\_ERROR;

}

如何正确理解和使用Command、Connection和 Recordset三个对象？

我知道它们都是连接数据库的“好手”，但在编程的具体应用中应该如何正确地使用它们呢？

下面我们简要介绍一下它们的功能和应用范围，以及相互之间的关系：

Connection对象

我们用ADO 提供的Connection对象来和数据库源建立连接，可以使用它建立和管理应用程序和 ODBC 数据库之间的连接（必须先应创建 Connection 对象）。Connection 对象具有各种属性和方法，我们还可以用它们打开和关闭数据库连接，并且发出查询请求来更新信息。

Recordset对象

尽管 Connection 对象简化了连接数据库和查询任务，但检索和显示数据库信息的 Connection 对象不能用于创建脚本，必须在知道已对数据库作出的更改后，才能使用查询实现更改。因此，对于检索数据、检查结果、更改数据库，ADO 提供了Recordset 对象。它可以根据我们的查询限制，检索并且显示一组数据库的记录，并且，Recordset 对象保持查询返回的记录的位置，允许一次一项逐步扫描结果。

我们还可以根据 Recordset 对象的指针类型属性设置来滚动和更新记录。数据库指针可以在一组记录中定位到特定的项，还可以用于检索和检查记录，然后在这些记录的基础上执行操作。如可以用CursorType和CursorLocation属性设置指针的类型，将结果返回给客户端应用程序（结果通常保留在数据库服务器上）并显示其他用户对数据库的最后一次更改。

可以看出，一个成功的数据库应用程序需要用 Connection 对象建立连接，并使用 Recordset 对象处理返回的数据才能完成。这样，我们几乎可以开发出执行任何数据处理任务的数据库应用程序。

另外，如果我们需要在连接尝试失败之前指定等待的时间，可以使用Connection 对象来设置属性。但如果仅仅想用 ADO 默认的连接属性来建立连接，我们就应该使用Recordset对象的Open方法来建立，这时，必须使用Connection 对象来保证连接安全。

Command对象

Command 对象用来改善查询。通过它，我们可以象用上面两个对象那样执行查询。它们唯一的不同之处，在于用Command对象，可以在数据库源上准备、编译我们的查询，并且可以反复使用一组不同的值来发出查询。显然，这种方式的编译查询的长处是可最大程度地减少向现有查询重复发出修改的请求所需的时间。

另，我们还可以在执行之前通过查询的可变部分的选项使SQL查询保持局部未定义，使我们不必在每次重新发出查询时重新建立查询。

**Connection,recordset,command 这三个在执行SQL语句**(2007-12-25 9:48:00)

Open 方法 (ADO Connection)   
   
   
 打开到数据源的连接。   
   
 语法   
   
 connection.Open ConnectionString, UserID, Password, Options   
   
 参数   
   
 ConnectionString 可选，字符串，包含连接信息。参阅 ConnectionString 属性可获得有效设置的详细信息。   
   
 UserID 可选，字符串，包含建立连接时所使用用户名。   
   
 Password 可选，字符串，包含建立连接时所使用密码。   
   
 Options 可选，ConnectOptionEnum 值。决定该方法是在连接建立之后（异步）还是连接建立之前（同步）返回。可以是如下某个常量：   
   
 常量 说明   
 adConnectUnspecified （默认）同步打开连接。   
 adAsyncConnect 异步打开连接。ConnectComplete 事件可以用于决定连接何时可用。   
   
   
   
 说明   
   
 使用 Connection 对象的 Open 方法可建立到数据源的物理连接。在该方法成功完成后连接是活跃的，可以对它发出命令并且处理结果。   
   
 使用可选的 ConnectionString 参数指定连接字符串，它包含由分号分隔的一系列 argument = value 语句。ConnectionString 属性自动继承用于 ConnectionString 参数的值，因此可在打开之前设置 Connection 对象的 ConnectionString 属性，或在 Open 方法调用时使用 ConnectionString 参数设置或覆盖当前连接参数。   
   
 如果在 ConnectionString 参数和可选的 UserID 及 Password 参数中传送用户和密码信息，那么 UserID 和 Password 参数将覆盖 ConnectionString 中指定的值。   
   
 在对打开的 Connection 的操作结束后，可使用 Close 方法释放所有关联的系统资源。关闭对象并非将它从内存中删除；可以更改它的属性设置并在以后再次使用 Open 方法打开它。要将对象完全从内存中删除，可将对象变量设置为 Nothing。   
   
 远程数据服务用法 当在客户端的 Connection 对象上使用 Open 方法时，在 Connection 对象上打开 Recordset 之前 Open 方法其实并未建立到服务器的连接。   
   
 /////////////////////////////////////////////////////////////////////////////////   
   
 Open 方法 (ADO Recordset)   
   
   
 打开游标。   
   
 语法   
   
 recordset.Open Source, ActiveConnection, CursorType, LockType, Options   
   
 参数   
   
 Source 可选，变体型，计算 Command 对象的变量名、SQL 语句、表名、存储过程调用或持久 Recordset 文件名。   
   
 ActiveConnection 可选。变体型，计算有效 Connection 对象变量名；或字符串，包含 ConnectionString 参数。   
   
 CursorType 可选，CursorTypeEnum 值，确定提供者打开 Recordset 时应该使用的游标类型。可为下列常量之一（参阅 CursorType 属性可获得这些设置的定义）。   
   
 常量 说明   
 AdOpenForwardOnly （默认值）打开仅向前类型游标。   
 AdOpenKeyset 打开键集类型游标。   
 AdOpenDynamic 打开动态类型游标。   
 AdOpenStatic 打开静态类型游标。   
   
   
   
 LockType 可选。确定提供者打开 Recordset 时应该使用的锁定（并发）类型的 LockTypeEnum 值，可为下列常量之一（参见 LockType 属性可获得详细信息）。   
   
 常量 说明   
 AdLockReadOnly （默认值）只读 — 不能改变数据。   
 AdLockPessimistic 保守式锁定（逐个） — 提供者完成确保成功编辑记录所需的工作，通常通过在编辑时立即锁定数据源的记录。   
 AdLockOptimistic 开放式锁定（逐个） — 提供者使用开放式锁定，只在调用Update 方法时才锁定记录。   
 AdLockBatchOptimistic 开放式批更新—用于批更新模式（与立即更新模式相对）。   
   
   
   
 Options 可选，长整型值，用于指示提供者如何计算 Source 参数（如果它代表的不是 Command 对象），或从以前保存 Recordset 的文件中恢复 Recordset。可为下列常量之一（参见 CommandType 属性可获得该列表中前五个常量的详细说明）。   
   
 常量 说明   
 adCmdText 指示提供者应该将 Source 作为命令的文本定义来计算。   
 adCmdTable 指示 ADO 生成 SQL 查询以便从在 Source 中命名的表中返回所有行。   
 adCmdTableDirect 指示提供者更改从在 Source 中命名的表中返回所有行。   
 adCmdStoredProc 指示提供者应该将 Source 视为存储过程。   
 adCmdUnknown 指示 Source 参数中的命令类型为未知。   
 adCmdFile 指示应从在 Source 中命名的文件中恢复保留（保存的）Recordset。   
 adAsyncExecute 指示应异步执行 Source。   
 adAsyncFetch 指示在提取 Initial Fetch Size 属性中指定的初始数量后，应该异步提取所有剩余的行。如果所需的行尚未提取，主要的线程将被堵塞直到行重新可用。   
 adAsyncFetchNonBlocking 指示主要线程在提取期间从未堵塞。如果所请求的行尚未提取，当前行自动移到文件末尾。   
   
   
   
 说明   
   
 使用 Recordset 对象的 Open 方法可打开代表基本表、查询结果或者以前保存的 Recordset 中记录的游标。   
   
 使用可选的 Source 参数指定使用下列内容之一的数据源：Command 对象变量、SQL 语句、存储过程、表名或完整的文件路径名。   
   
 如果 Source 是文件路径名，它可以是完整路径（“c:\dir\file.rst”）、相对路径（“..\file.rst”）或 URL（“http://files/file.rst”）。   
   
 ActiveConnection 参数对应于 ActiveConnection 属性，并指定在哪个连接中打开 Recordset 对象。如果传送该参数的连接定义，则 ADO 使用指定的参数打开新连接。可以在打开 Recordset 之后更改该属性的值以便将更新发送到其他提供者。或者可以将该属性设置为 Nothing（在 Microsoft Visual Basic 中）以便将 Recordset 与所有提供者断开。   
   
 对于直接对应于 Recordset 对象属性的参数（Source、CursorType 和 LockType），参数和属性的关系如下：   
   
 在 Recordset 对象打开之前属性是读/写。   
   
   
 除非在执行 Open 方法时传送相应的参数，否则将使用属性设置。如果传送参数，则它将覆盖相应的属性设置，并且用参数值更新属性设置。   
   
   
 在打开 Recordset 对象后，这些属性将变为只读。   
 注意 对于其 Source 属性被设置为有效 Command 对象的 Recordset 对象，即使 Recordset 对象没有打开，ActiveConnection 属性也是只读的。   
   
 如果在 Source 参数中传送 Command 对象并且同时传递 ActiveConnection 参数，那么将产生错误。Command 对象的 ActiveConnection 属性必须已经设置为有效的 Connection 对象或者连接字符串。   
   
 如果在 Source 参数中传送的不是 Command 对象，那么可以使用 Options 参数优化对 Source 参数的计算。如果没有定义 Options 则性能将会降低，原因是 ADO 必须调用提供者以确定参数是否为 SQL 语句、存储过程或表名。如果已确定所用的 Source 类型，则可以设置 Options 参数以指示 ADO 直接跳到相关的代码。如果 Options 参数不匹配 Source 类型，将产生错误。   
   
 如果不存在与 Recordset 关联的连接，Options 参数的默认值将为 adCmdFile。这是持久 Recordset 对象的典型情况。   
   
 如果数据源没有返回记录，那么提供者将 BOF 和 EOF 属性同时设置为 True，并且不定义当前记录位置。如果游标类型允许，仍然可以将新数据添加到该空 Recordset 对象。   
   
 在打开的 Recordset 对象上完成操作时，可使用 Close 方法释放任何相关的系统资源。关闭对象并非将它从内存中删除，可以更改它的属性设置并且在以后使用 Open 方法再次将其打开。要将对象从内存中完全删除，可将对象变量设置为 Nothing。   
   
 在设置 ActiveConnection 属性之前调用不带操作数的 Open，可通过将字段追加到 Recordset Fields 集合创建 Recordset 的实例。   
   
 如果已经将 CursorLocation 属性设置为 adUseClient，就可以采用两种途径之一异步检索行。建议使用的方法是将 Options 设置为 adAsyncFetch。或者，可以使用在 Properties 集合中的“异步行集合处理”动态属性，但如果未将 Options 参数设置为 adAsyncFetch，则可能丢失相关的被检索事件。   
   
 注意 在 MSRemote 提供者中的背景提取仅能通过 Open 方法的 Options 参数得到支持。   
Execute 方法 (ADO Command)   
   
   
 执行在 CommandText 属性中指定的查询、SQL 语句或存储过程。   
   
 语法   
   
 对于按行返回的 Command：   
   
 Set recordset = command.Execute( RecordsAffected, Parameters, Options )   
   
 对于非按行返回的 Command：   
   
 command.Execute RecordsAffected, Parameters, Options   
   
 返回值   
   
 返回 Recordset 对象引用。   
   
 参数   
   
 RecordsAffected 可选，长整型变量，提供者向其返回操作所影响的记录数目。RecordsAffected 参数仅应用于操作查询或存储过程。RecordsAffected 不返回由返回结果的查询或存储过程所返回的记录数目。详细信息，请使用 RecordCount 属性。   
   
 Parameters 可选，变体型数组，使用 SQL 语句传送的参数值。（用该参数传送时输出参数将不返回正确值。）   
   
 Options 可选，长整型值，指示提供者如何计算 Command 对象的 CommandText 属性。该值可为下列常量之一：   
   
 常量 说明   
 AdCmdText 指示提供者应按命令的文本定义（如 SQL 语句）来计算 CommandText。   
 AdCmdTable 指示 ADO 应生成 SQL 查询以便从 CommandText 命名的表中返回所有行。   
 AdCmdTableDirect 指示提供者应从 CommandText 命名的表中返回所有行。   
 AdCmdStoredProc 指示提供者应按存储过程计算 CommandText。   
 AdCmdUnknown 指示 CommandText 中的命令类型未知。   
 adAsyncExecute 指示命令应异步执行。   
 adAsyncFetch 指示对由 CacheSize 属性指定的初始数量之后的剩余行应使用异步提取。   
   
   
   
 本列表中前 4 个常量的详细说明请参见 CommandType 属性。   
   
 说明   
   
 使用 Command 对象的 Execute 方法可执行在对象的 CommandText 属性中指定的查询。如果 CommandText 属性指定按行返回查询，执行所产生的任何结果都将存储在新的 Recordset 对象中。如果该命令不是按行返回查询，则提供者返回关闭的 Recordset 对象。某些应用程序语言允许忽略该返回值（如果不需要任何 Recordset）。   
 如果查询带有参数，将使用 Command 对象中参数的当前值，除非通过 Execute 调用传送的参数覆盖它们。可以在调用 Execute 方法时通过省略某些参数的新值来覆盖参数子集。指定参数的次序与其在方法中被传送的次序相同。例如，如果有 4 个（或更多）参数并且希望只为第一个和第四个参数传送新值，则可以将 Array(var1,,,var4) 作为 Parameters 参数传送。   
   
 注意 在 Parameters 参数中传送时输出参数将不返回正确的值。   
   
 该操作结束后将发出 ExecuteComplete 事件   
   
 ///////////////////////////////////////////////////////////////////////   
   
 Execute 方法 (ADO Connection)   
   
   
 执行指定的查询、SQL 语句、存储过程或特定提供者的文本等内容。   
   
 语法   
   
 对于非按行返回的命令字符串：   
   
 connection.Execute CommandText, RecordsAffected, Options   
   
 对于按行返回的命令字符串：   
   
 Set recordset = connection.Execute (CommandText, RecordsAffected, Options)   
   
 返回值   
   
 返回 Recordset 对象引用。   
   
 参数   
   
 CommandText 字符串，包含要执行的 SQL 语句、表名、存储过程或特定提供者的文本。   
   
 RecordsAffected 可选，长整型变量，提供者向其返回操作所影响的记录数目。   
   
 Options 可选，长整型值，指示提供者应如何计算 CommandText 参数，可为下列值：   
   
 常量 说明   
 AdCmdText 指示提供者应按命令的文本定义计算 CommandText。   
 AdCmdTable 指示 ADO 应生成 SQL 查询以便从 CommandText 命名的表中返回所有行。   
 AdCmdTableDirect 指示提供者应从 CommandText 命名的表中返回所有行。   
 AdCmdTable 指示提供者应按表名计算 CommandText。   
 AdCmdStoredProc 指示提供者应按存储过程计算 CommandText。   
 AdCmdUnknown 指示 CommandText 参数中的命令类型未知。   
 adAsyncExecute 指示命令应该异步执行。   
 adAsyncFetch 指示对在 CacheSize 属性指定的初始数量之后的剩余行使用异步提取。   
   
   
   
 本列表中前 4 个常量的详细说明请参见 CommandType 属性。   
   
 说明   
   
 使用 Connection 对象的 Execute 方法，可执行任何在指定连接的 CommandText 参数中传送给方法的查询。如果 CommandText 参数指定按行返回的查询，执行产生的任何结果将存储在新的 Recordset 对象中。如果命令不是按行返回的查询，则提供者返回关闭的 Recordset 对象。   
返回的 Recordset 对象始终为只读、仅向前的游标。如需要具有更多功能的 Recordset 对象，应首先创建具有所需属性设置的 Recordset 对象，然后使用 Recordset 对象的 Open 方法执行查询并返回所需游标类型。   
   
 CommandText 参数的内容对提供者是特定的，并可以是标准的 SQL 语法或提供者支持的任何特殊命令格式。   
   
 该操作完成后将产生 ExecuteComplete 事件。

自己理解为

command执行存储过程