**ADO.NET**

**ADO.NET是什么？**

ADO.NET是一组类库，这组类库能让我们通过c#代码的方式来访问、修改数据库中的数据，ADO.NET是数据库访问驱动

**ADO.NET中的连接池技术**

**执行过程**

当我们创建一次SqlConnection对象conn的时候，会创建一个连接池，当执行到conn.Open()打开数据库的连接的时候，conn会在连接池中寻找有没有内连接对象（可以自己设置连接池的一些初始值），如果有，则直接拿过来用，如果没有，则自己创建一个内连接对象，当我们执行到conn.Close()的时候，conn会把刚刚使用的内连接对象放回连接池中，以便第二个SqlConnection的使用，当我们创建第二个SqlConnection对象conn2的时候，当执行到conn2.Open()打开数据库的链接的时候，conn2会在连接池中寻找有没有内连接对象，因为刚刚conn已经创建了一个，所以就可以直接拿来用，当执行到conn2.Close()的时候，又会把conn创建的内连接对象放回连接池中去

**注意问题：**

1. SqlConnection对象就像是内连接对象的一层外衣，两者需区分开来
2. 当我们Close()的时候并不是释放的SqlConnection的资源，而只是把内连接对象放回连接池中去，当我们Dispose()的时候才是真正的释放了SqlConnection对象的资源
3. 当我们Open()的时候，才会在连接池中寻找有没有内连接对象，如果没有，则创建一个
4. 连接池根据我们的连接字符串创建连接池，不同的连接字符串创建不同的连接池，一个连接字符串对应一个连接池

**优点：**

1. 通过连接池技术，当我们频繁的对数据库进行打开关闭的操作的时候，极度的占用了内存空间，减少了运行的效率，因为我们一直在创建和关闭对象，当时用了连接池技术，我们就不用平凡的去创建对象，而是先去连接池中寻找有没有内连接对象，如果有的话就拿来用
2. 使用内连接对象可以提升连接对象重用的效率

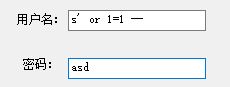
**SQL漏洞注入问题**

当我们在一个程序集中写登录模块那里直接同字符串拼接的方式写一个SQL脚本，就会存在SQL漏洞注入的问题

**那么，什么是SQL漏洞注入？如何实现的？**

我们先来看看我们的SQL脚本：

微信截图_20180316220023

当我们在填写用户名和密码的文本框中填写文本，当我们填写的一些文本如：，这时候程序通过字符串连接的方式，把微信截图_20180316220137连接进字符串的时候，其实真实的SQL脚本是这种效果：

微信截图_20180316220258

我们可以看到，or条件还有单引号，成功的把where的过滤条件变成一个永远为true的了，并且 后面的 -- 注释掉了后面的内容，这样就造成了我们可以把数据库中的所有内容查询出来

**那么如何解决？**

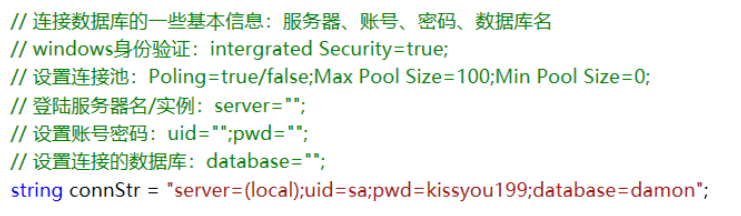
我们可以通过在SqlCommand所指定的SQL脚本中填写参数（通过 @参数名 ），然后使用SqlCommand的Parameters属性的AddWithValue函数来为参数赋值，SQL脚本中的参数的作用能为我们所指定的参数的值编码化，也就是说参数的值中出现的特殊符号能够编码成不被SQL所识别的字符，这样就能够防止SQL漏洞注入攻击

**ADO.NET中的一些常用类**

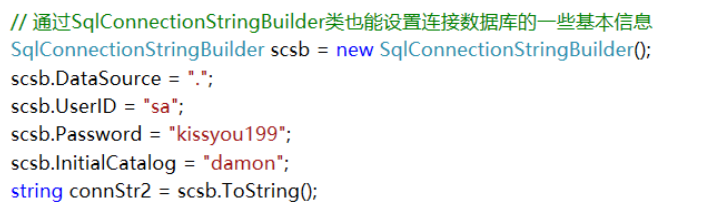
1. **SqlConnection类：该类用于建立一条连接数据库的通道，ADO.NET执行SQL语句的类需通过这条通道才能访问、修改数据库中的数据**

**->** **编写SQL连接字符串：**

1. 直接写一个string的字符串



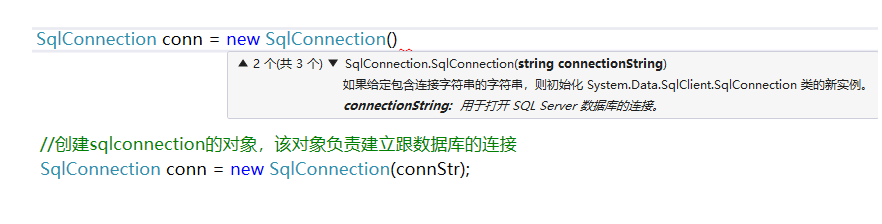
1. 通过SqlConnectionStringBuilder的属性的赋值，最后ToString(）



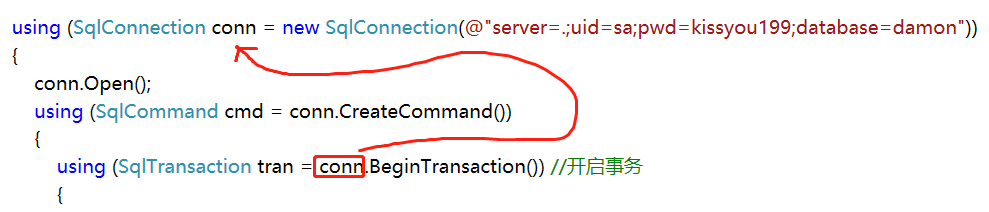
1. 通过配置文件 App.config **（推荐）**



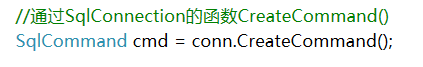
**->** **创建Sqlconnection的对象，构造函数需要SQL连接字符串作为参数：**



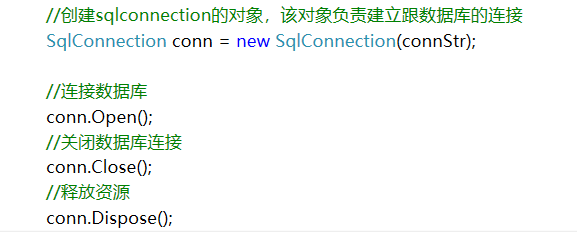
**->** **BeginTransaction函数：该函数表示 SQL事务的开启 ，并且能够返回一个SqlTransaction类型的对象，该对象能够在ADO.NET完成SQLServer中一件事务的相关操作**

****

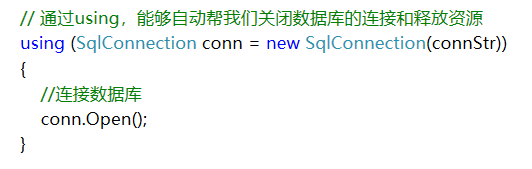
**->** **CreateCommand函数：该函数能够返回一个SqlCommand类型的对象，这个SqlCommand对象已经指定好了进行对数据库进行操作的通道为该函数的执行者所创建的通道**



**->** **手动的：打开数据库的连接/关闭数据库的连接/释放资源：**



**->** **通过using()来自动关闭数据库的连接和释放SqlConnection的资源**



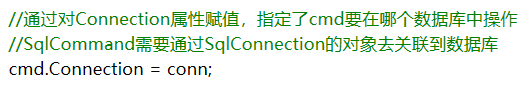
1. **SqlCommand类：该类用于执行SQL脚本，需要与SqlConnection类的对象进行关联，才能使用该类进行对数据库的操作**

**-> 创建SqlCommand类的对象，并且指定该对象所执行的SQL语句和需要通过哪个SqlConnection通道对数据库进行操作**

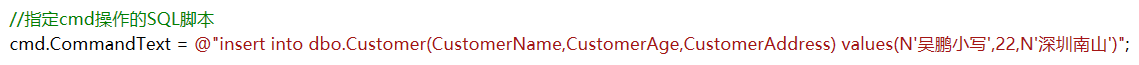
1. 通过构造函数来指定SqlCommand所执行的SQL语句和需要通过的SqlConnection通道



1. 通过给SqlCommand的属性赋值来制定SqlCommand所执行的SQL语句和需要通过的SqlConnection通道
2. connection属性：能够指定SqlCommand需要通过哪条SqlConnection通道执行数据库的操作



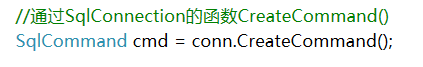
1. CommandText属性：指定SqlCommand**执行对数据库操作所需的SQL脚本**或**通过名字执行存储过程**
2. **执行对数据库操作所需的SQL脚本**



1. **执行存储过程**



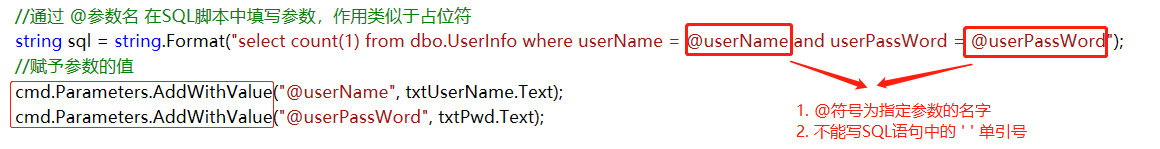
1. 通过SqlConnection的函数CreateCommand来指定SqlCommand的对象需要通过的连接通道



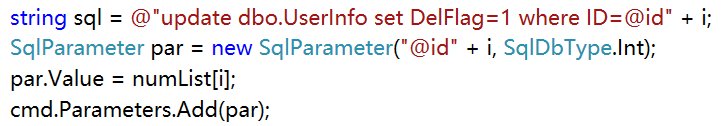
**-> Parameters属性的AddWithValue函数：指定在SQL脚本中所写的参数的值**

**普通的参数赋值和添加，两种方式：**

1. 方式一：Paramerters的AddWithValue()函数，指定参数和参数的值

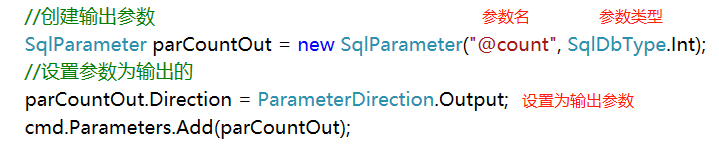


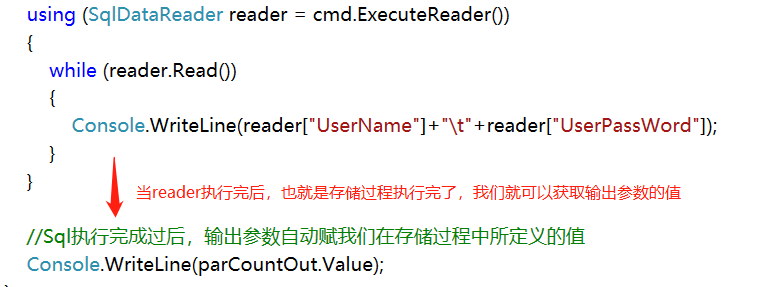
1. 方式二：创建Parameter的对象，通过构造函数指定参数和参数的类型，然后通过Parameter的Values属性指定参数的值，最后通过Parameters的Add()函数



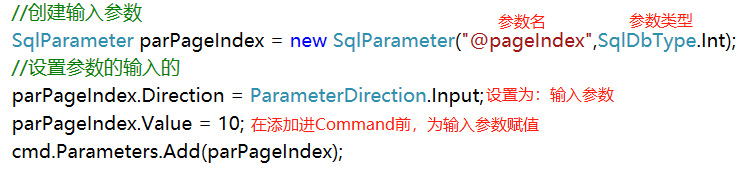
**存储过程中，输出参数、输入参数的添加：**

1. 输出参数：
2. 通过SqlParameter参数对象的Direction属性，赋值枚举值：ParameterDirection.Output来指定该参数类型为输出参数
3. 当存储过程执行完后，我们可以通过该输出参数对象的Value来取值
4. 我们不需要在添加进Command之前，赋值Value，因为是输出参数，在存储过程中必须对其赋值，所以在执行完后自动赋值该参数





1. 输入参数：
2. 通过SqlParameter参数对象的Direction属性，赋值枚举值：ParameterDirection.Input来指定该参数类型为输入参数
3. 我们需要在添加进Command之前，赋值Value，因为是输入类型的参数

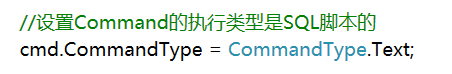


**->** **CommandType属性：该属性的赋值能够指定SqlCommand的执行类型**

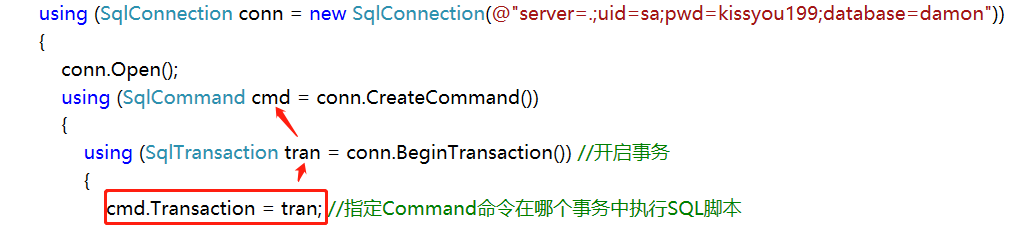
1. CommandType = Command.StoredProcedure：指定SqlCommand的执行类型为存储过程



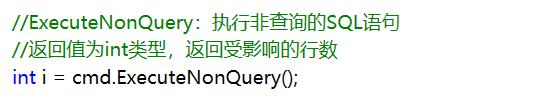
1. CommandType = Command.Text：指定SqlCommand的执行类型为SQL脚本



**-> Transaction属性：该属性的赋值使SqlCommand命令在我们所指定的一件事务中执行SQL脚本**

****

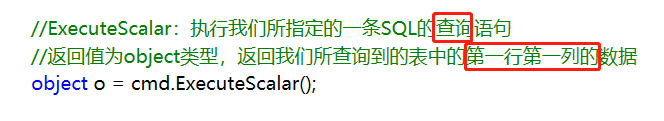
**->** **ExecuteNonQuery函数：执行非查询SQL语句，当我们指定所执行的SQL语句与使用哪个SqlConnection连接通道后才能使用**



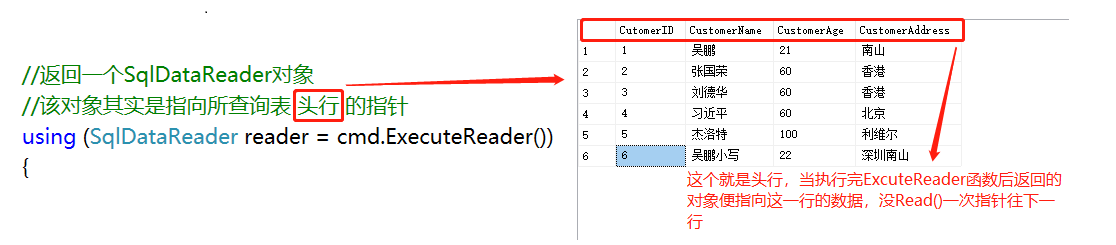
**->** **ExecuteScalar函数：执行一条SQL语句，当我们指定所执行的SQL语句与使用哪个SqlConnection连接通道后才能使用**

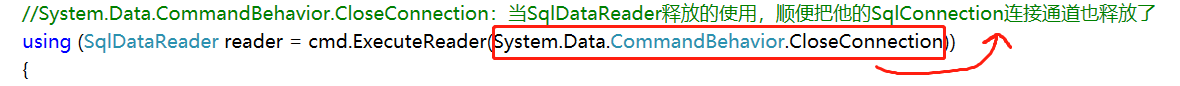
**注意**

1. **如果执行SQL语句为一条查询语句且返回表，则返回表中第一行第一列的数据**
2. **如果执行的SQL语句非查询，则返回的object为null，但是对数据库的操作还是可以正常执行**



**-> ExecuteReader函数：执行SQL的查询语句，返回SqlDataReader的对象，该对象其实就是指向我们所查询表中的行的指针**





**-> 通过using()来自动的释放SqlCommand的资源**



1. **SqlDataReader类：该类的对象为SqlCommand类中的ExcuteReader函数所返回指向SQL语句查询结果的一张表的指针**

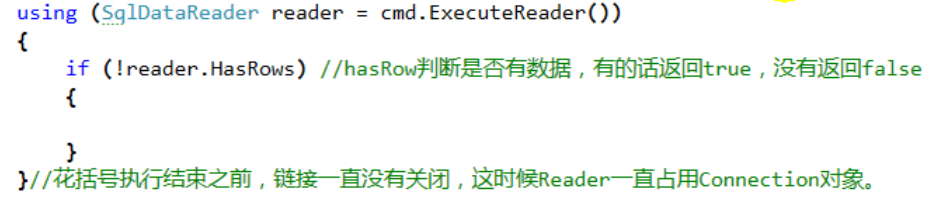
**注意：**

1. 当SqlDataReader对象正在使用且资源没有释放的时候，不允许使用其他的SqlCommand对象通过SqlConnection的连接通道对数据库的数据进行CRUD操作，因为这时候SqlDataReader的对象正在占用SqlConnection的连接通道来对数据库中的数据进行读取，**SqlDataReader必须独享一个SqlConnetcion**

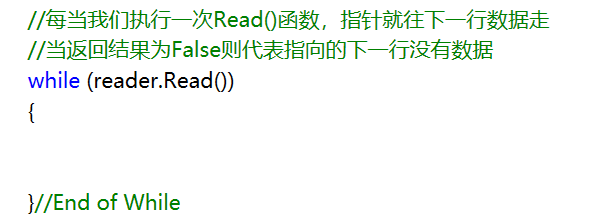
****

1. 当建立好了SqlDataReader连接数据库的连接后，SqlDataReader对象的指针其实一直是指向了数据库中指定表的行上，也就是说**一直是处于一个连接的状态**，并且数据库中指定表的数据一直都是在数据库中的，**只有当我们通过SqlDataReader对象的下标来访问数据的时候才会取走数据**，简而言之，**SqlDataReader对象在资源释放前只是一个一直指向数据库表中的行的指针**

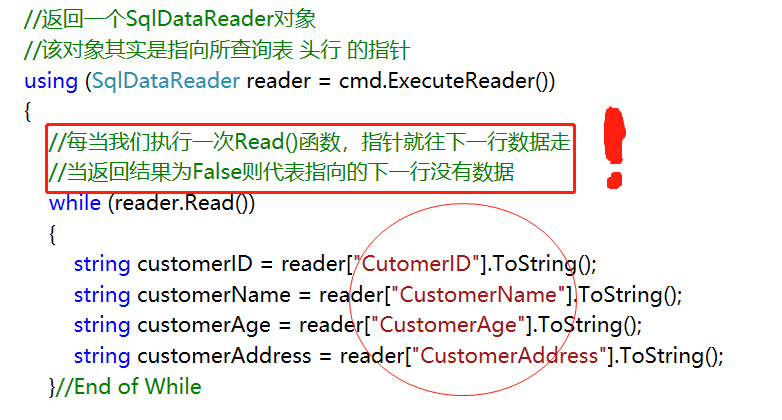
**-> HasRows()函数：判断表中是否有一行/多行的数据，如果有则返回true，没有则返回false**



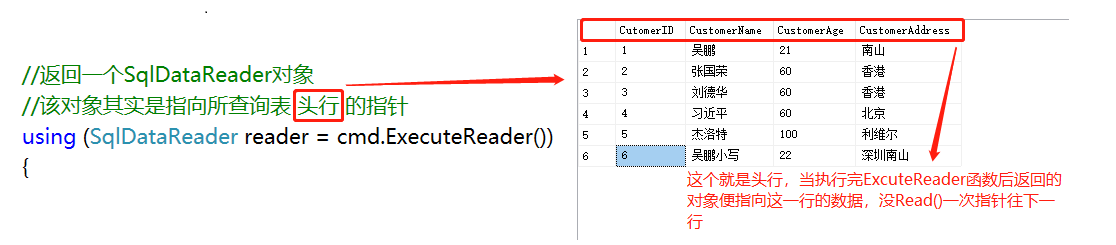
**-> Read()函数：让指向表的指针往下一行的数据走，返回一个bool值，当结果为False的时候代表指向的下一行没有数据，为True的时候代表指向下一行是有数据的**



**-> 填写表中的列名至SqlDataReader对象的下标，我们能够获取到当前SqlDataReader对象所指向的行中，我们填写的对应列名的值**



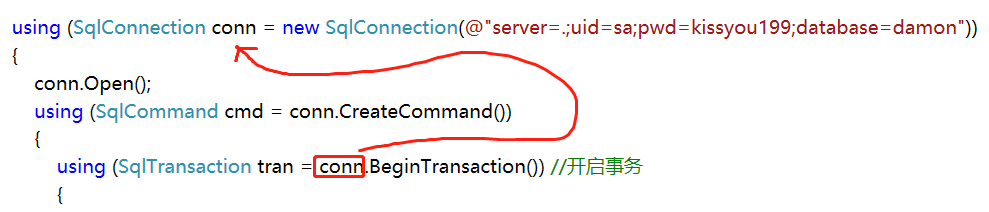
**-> 通过using()来自动释放SqlDataReader的资源**



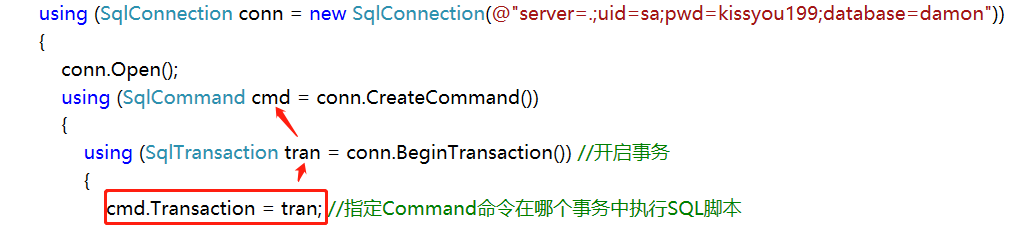
1. **SqlTransaction类/TransactionScope类：这两个类都能够帮助我们在ADO.NET中实现SQLServer中的事务**

**SqlTransaction类的使用：**

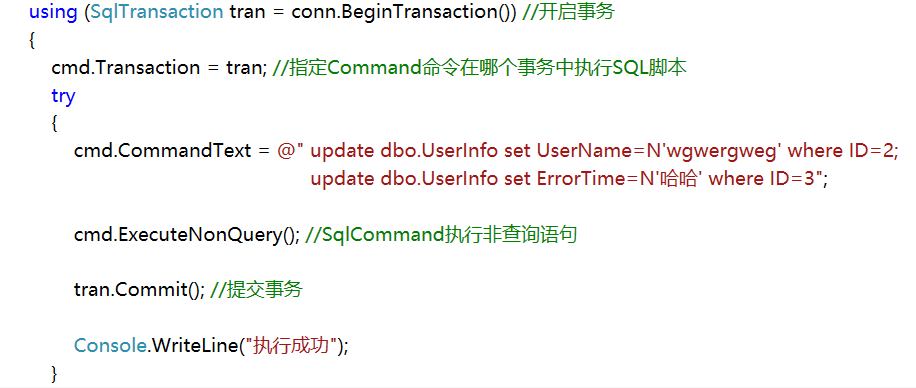
**-> 通过SqlConnection的BeginTransaction函数创建SqlTransaction类型的对象，并且让事务处于开启状态**

****

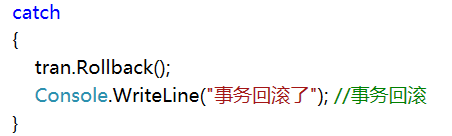
**-> 通过SqlCommand的Transaction属性的赋值，能够使SqlCommand对象在我们所指定的事务中去执行SQL脚本**

****

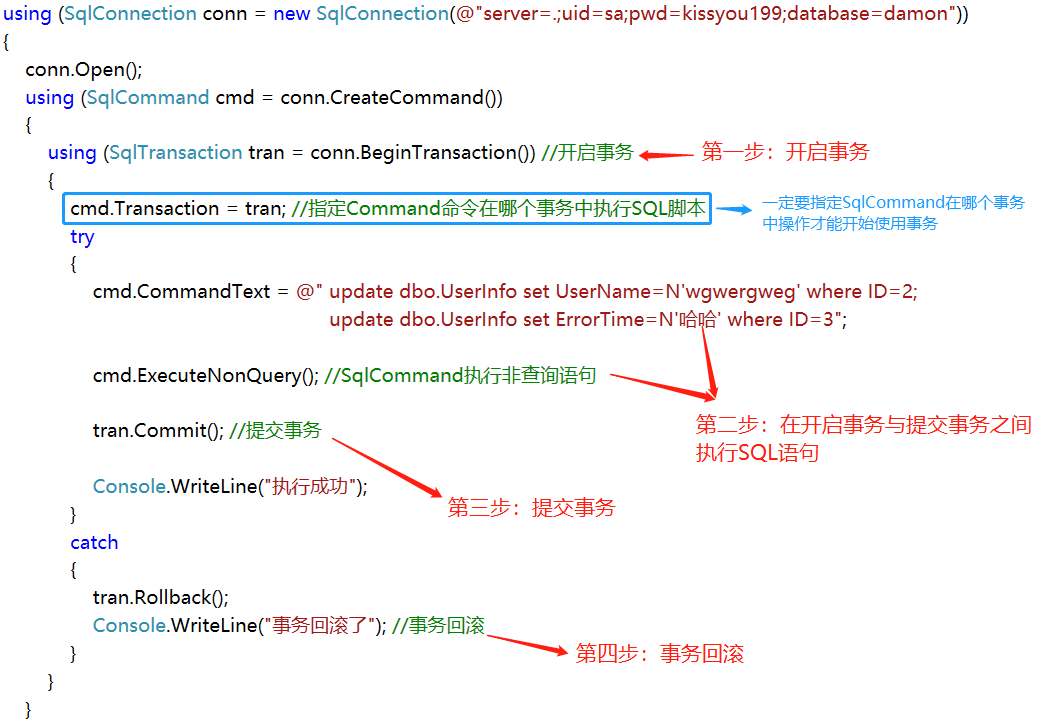
**-> Commit()函数：该函数能够提交事务，写在try中**

****

**-> Rollback()函数：该函数能够手动的使事务回滚，写在catch中**

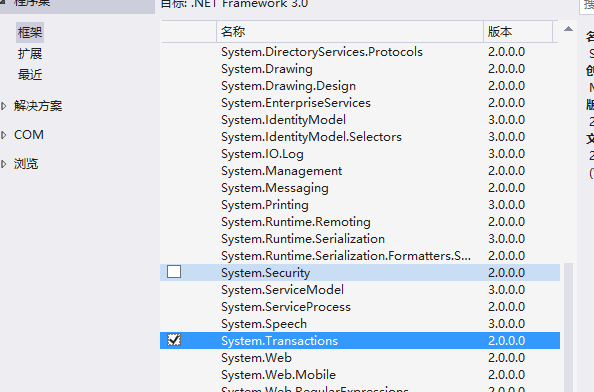
****

**-> 完整演示：**

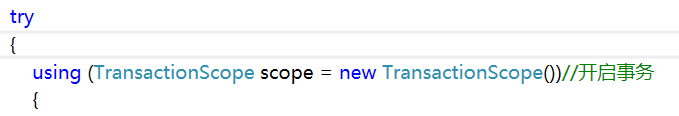
****

**TransactionScope类的使用：**

**-> 要想使用该类，必须先添加其引用，并导入其所在的NameSpace**

****

**-> 创建TransactionScope类的对象，当我们创建好对象后，事务便处于开启的状态**

****

**-> Complete()函数：该函数能够用于提交事务，当事务提交失败的时候，自动让事务回滚，不用在catch中写SqlTransaction中的Rollback()函数来让事务手动的进行回滚**

****

**-> 与SqlTransaction一样，开启事务和提交事务需要写在try中，回滚事务后的提示需要写在catch中**

****

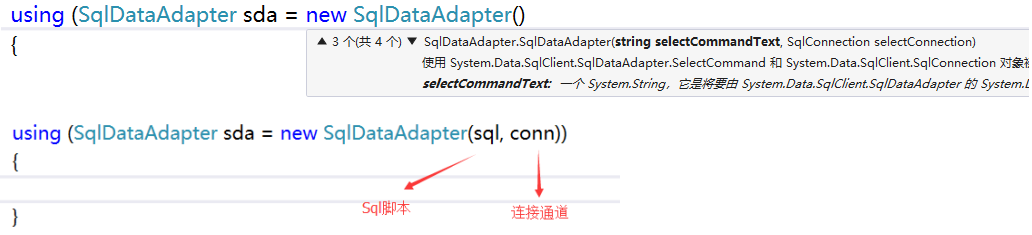
1. **SqlDataAdapter类：该类能够帮我们一次性读取指定数据库中的表的所有数据，然后加载到内存表DataTable当中，跟SqlDataReader区分**

**-> SqlDataAdapter数据访问适配器的实现原理**

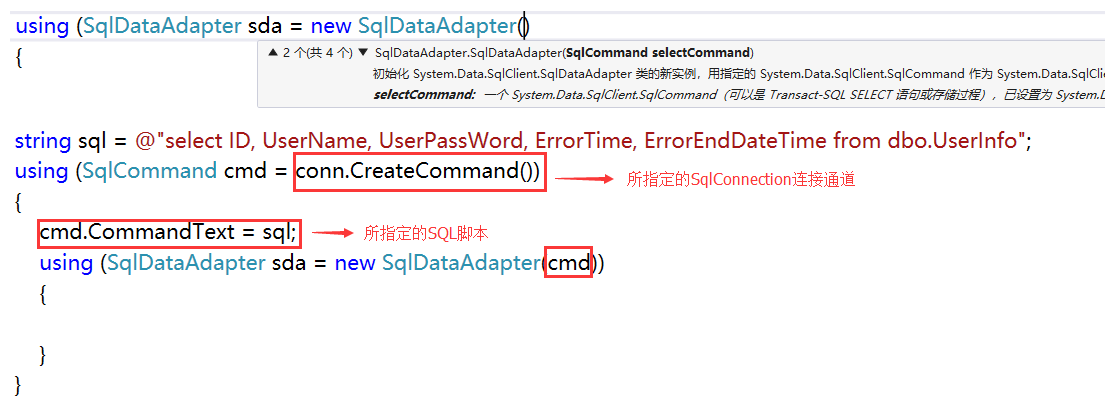
1. SqlDataAdapter帮我们连接上了数据库
2. 通过我们所指定的SelectCommand查询出数据库中表的数据，并把数据一次性全部加载到程序所占用的内存当中
3. 起到一个适配器的作用，把数据库中的数据的类型转换成c#能够识别的类型

**-> 创建SqlDataAdapter数据访问适配器的对象，通过构造函数指定了SqlDataAdapter所执行的SelectCommand，和需要通过的SqlConnection的连接通道**

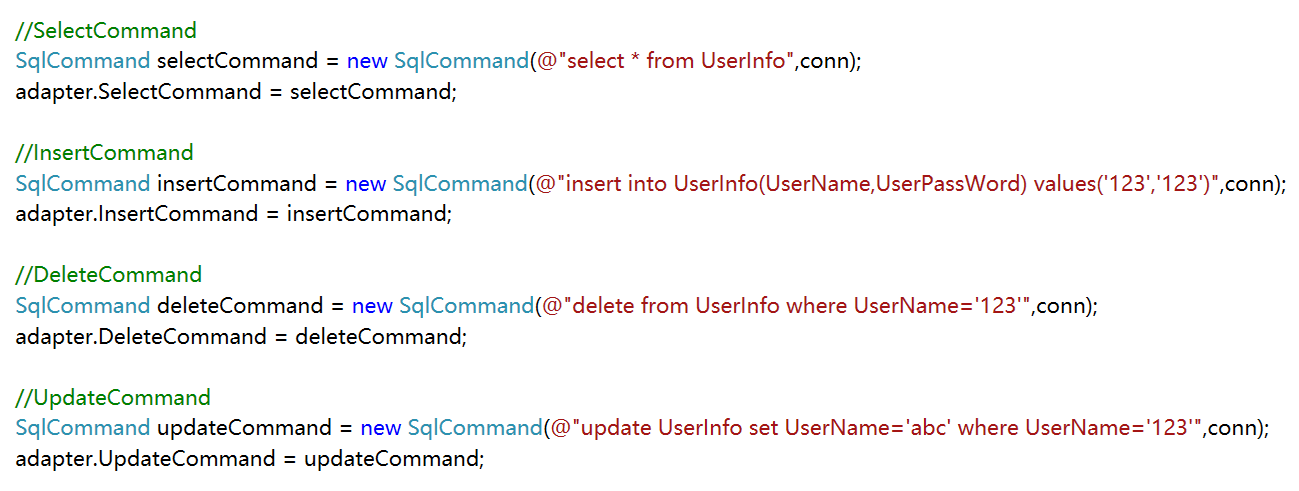
1. 通过在SqlDataAdapter中的构造函数指定查询的SQL脚本和SqlConnection连接通道来创建SqlDataAdapter对象



1. 通过一个已经指定好查询的SQL脚本和SqlConnection连接通道的SelectCommand对象来创建SqlDataAdapter对象



**-> DeleteCommand/InsertCommand/SelectCommand/UpdateCommand属性：这4个属性为Adapter对象执行相关函数如：Update()、Fill()…的前提条件，该属性指定了Adapter所执行的Command对象，和使用与Adapter对象函数相关的CRUD的SQL脚本**

****

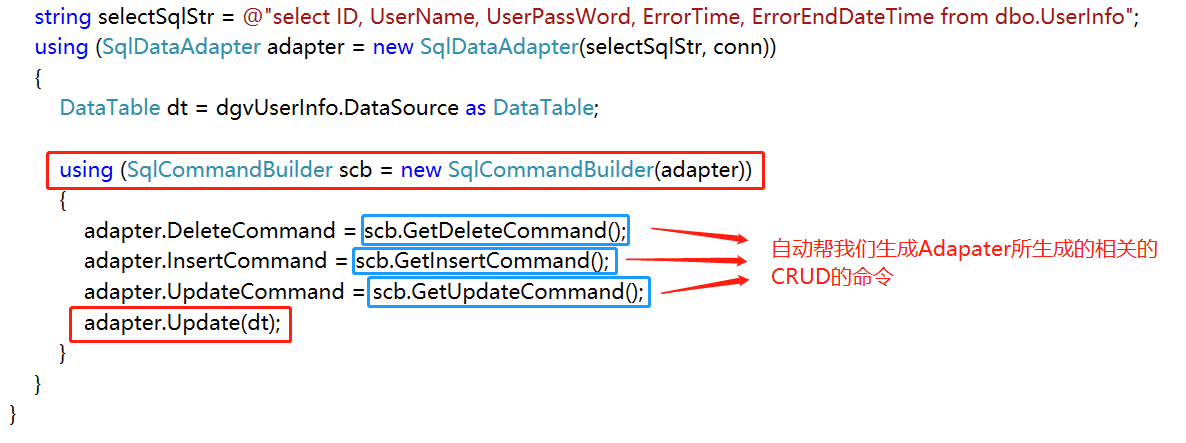
**-> Update()函数：该函数能把与数据库中的表的列与类型对应一致的DataTable内存表映射到数据库中去**

**该函数执行的注意事项：**

1. SqlDataAdapter对象必须已经指定好了CRUD的SqlCommand的对象以应对DataTable中的对数据的CRUD的操作的需求，通过指定了这四个Command我们才能使用该函数
2. 因为我们需要指定4个CRUD的SqlCommand对象给Adapter才能使用Update()函数，而不同的SqlCommand需要填写不同的SQL脚本，而我们可以通过**SqlCommandBuilder**来辅助Adapter生成相关的CRUD命令

**通过SqlCommandBuilder的类能够辅助Adapter生成所的相关的CRUD的命令**

1. **SqlCommandBuilder**类辅助Adapter生成相关的CRUD命令



**-> Fill()函数：该函数把Adapter从数据库中 一次性读取到的 放在程序所占用内存的 所有数据填充到一张DataTable内存表当中**

**该函数执行的注意事项：**

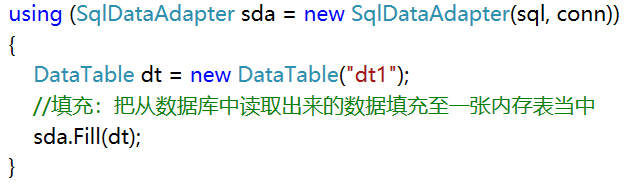
1. SqlDataAdapter对象必须已经指定好了一个SelectCommand才能执行该函数，Adapter需要查询到表的数据，才能填充到我们所定义的DataTable内存表

**当执行到该函数的时候再其内部干了4件事情：**

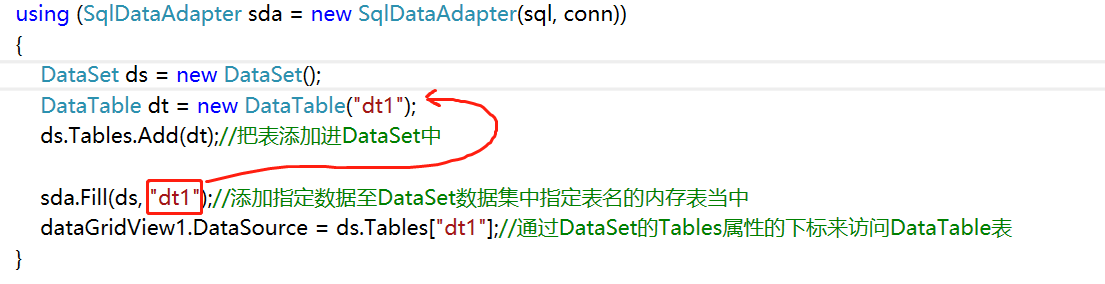
1. 先判断SqlConnection是否已初始化，如果没有Open()连接那么就自动帮我们打开，如果Open()了则无视
2. 初始化了一个SqlCommand对象，指定了该对象所执行的SQL脚本
3. 通过SqlCommand对象执行ExecuteReader()函数，通过该函数所返回的SqlDataReader指针读取数据库中我们所指定表的所有数据
4. 把SqlDataReader指针所读取到的所有数据填充到一张DataTable内存表当中

**该函数的使用：**

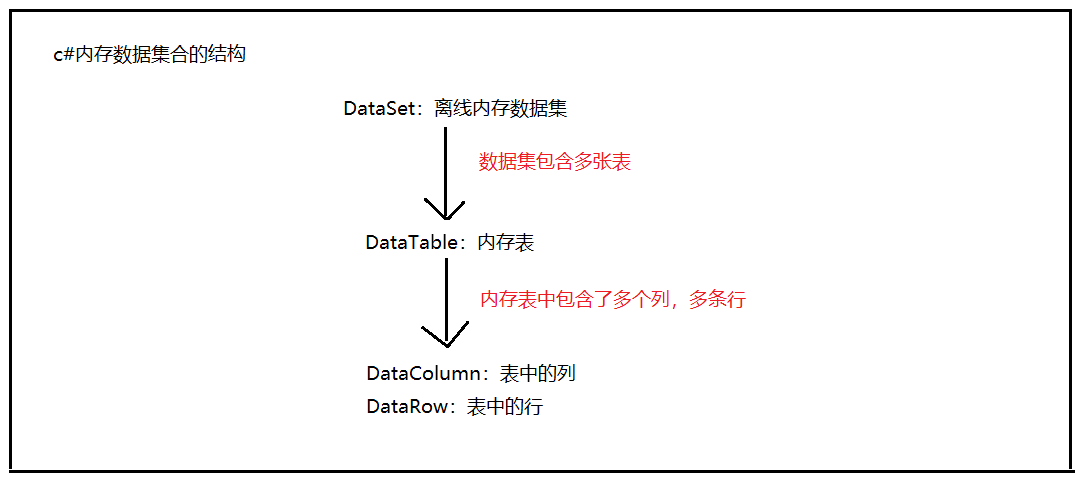
1. 通过Fill()填充一张DataTable内存表



1. 通过Fill()填充一个DataSet数据集中，指定名字的内存表DataTable

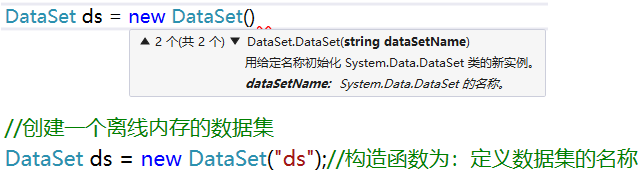


1. **DataSet离线内存数据集 -> DataTable内存表 -> DataColumn表的列/DataRow表的行**

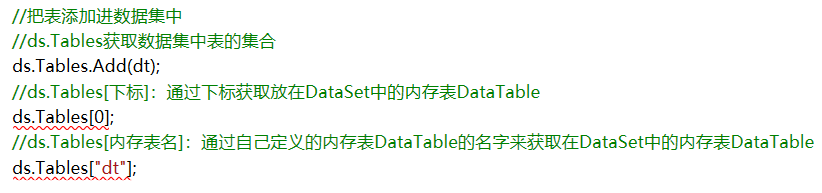
****

**-> DataSet类：该类为离线内存数据集，里面包含了多张内存表**

1. 创建DataSet的对象

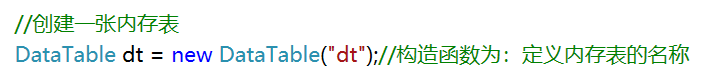


1. 把表添加进DateSet数据集中：Tables属性能够获取在DataSet中存放的内存表DataTable的集合

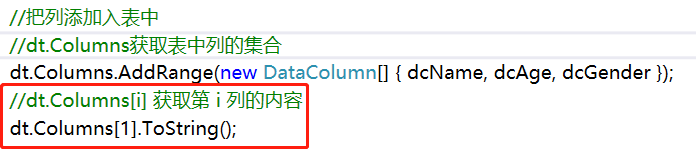


**-> DataTable类：该类为一张内存表，里面包含了表中的 列DataCoulmn/行DataRow**

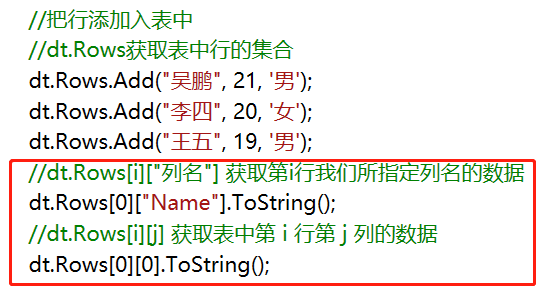
1. 创建DataTable的对象



1. 把列添加进DataTable内存表中：Columns属性能够获取内存表DataTable中列的集合



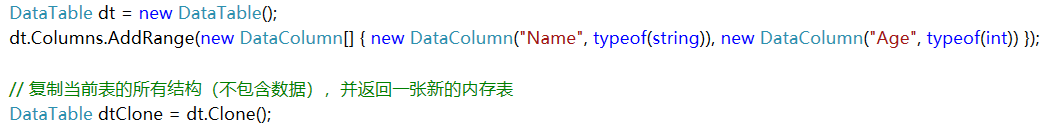
1. 把行添加进DataTable内存表中：Rows属性获取表中行的集合



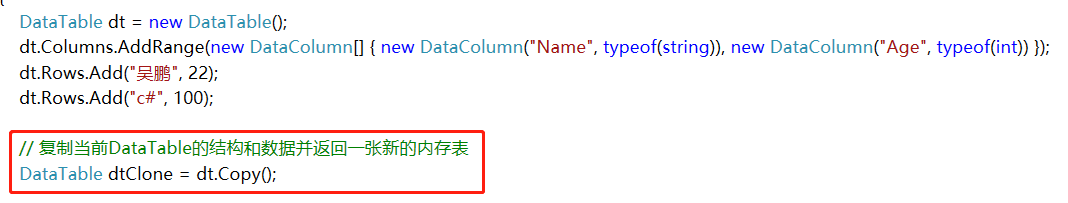
1. 把DataRow中的数据添加至DataTable中



1. 复制当前DataTable的结构(不包含数据)并返回一张新的内存表

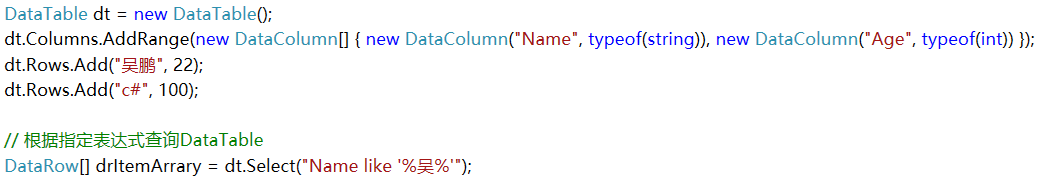


1. 复制当前DataTable的结构和数据并返回一张新的内存表



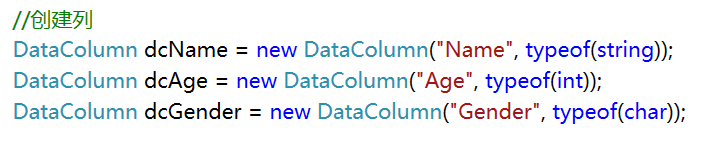
1. 通过一个查找表达式（类似于SQL语句中WHERE条件后面的部分）查找DataTable中的数据，并返回一个DataRow的集合

详情：<https://blog.csdn.net/jiankunking/article/details/17992217>

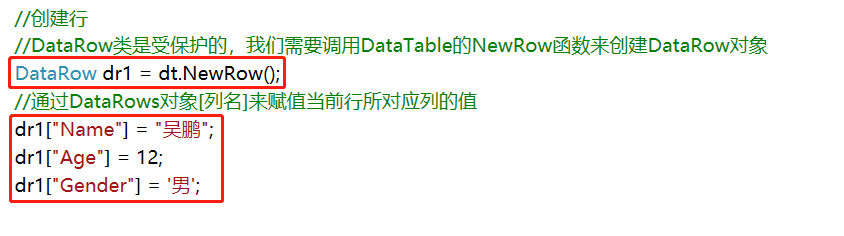


**-> DataColumn列/DataRow行：该类用于创建列和行的对象**

1. DataColumn：创建DataColumn的对象然后通过DataTable.Column.Add()函数添加进表中



1. DataRow：**DataRow不能创建对象，他是受保护的，所以我们只能通过以下方式来创建对象**



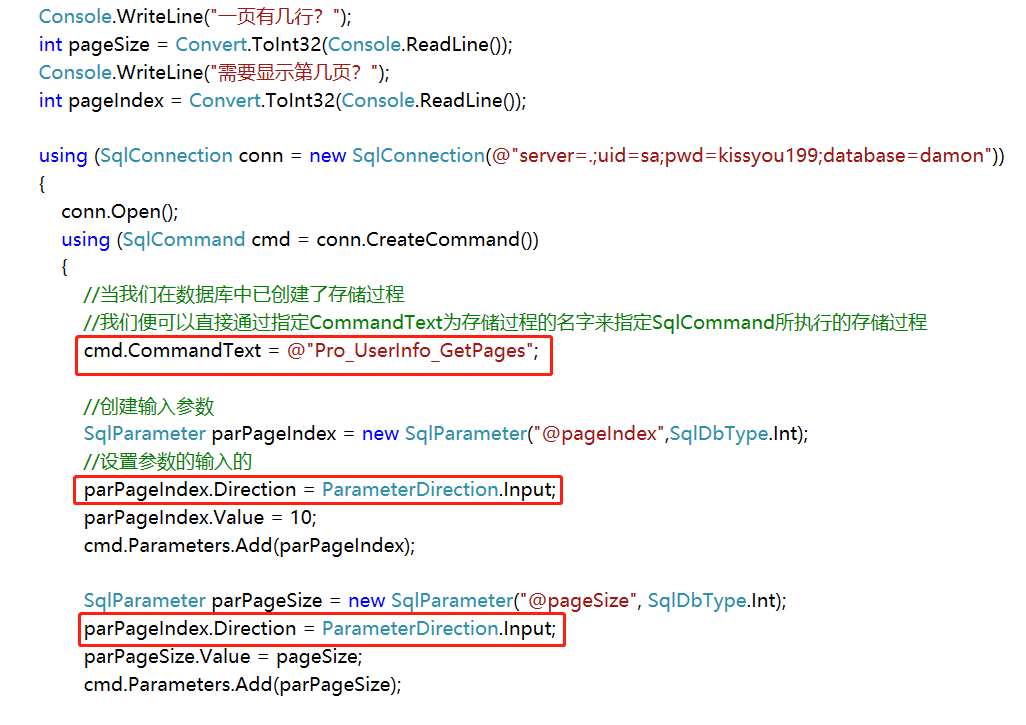
**-> 整合使用**



1. **在ADO.NET中执行SQL的存储过程**

**因为使用方法和普通的就一点点区别，不同点已画出，具体可以参照SqlCommand类中，里面有具体的详解**

**以下直接上代码：**

****

****