结构化查询语句：SQL（Structured Query Language）

ADO.NET基本概念:  
（作为.NET框架的重要组成部分，ADO.NET类封装在System.Data.dll中，并且与System.Xml.dll中的XML类集成。当编译使用System.Data命名空间的代码时，需要引用System.Data.dll和System.Xml.dll）

特色:

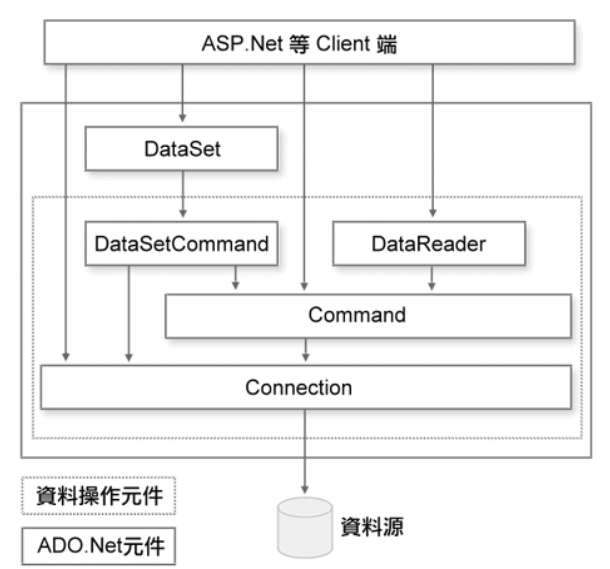
ADO对象是继ODBC之后微软主推存取数据的最新技术,ADO对象是程序开发平台用来和OLEDB沟通的媒介,ADO目前的最新版本是ADO.NET。ADO.NET不像以前的ADO版本是站在**为了存取数据库**的观点二设计的,ADO.NET是为了因应广泛的数据控制而设计,所有用起来比以前的ADO更灵活有弹性,也提供了更多的功能.ADO.NET的出现是要提供更有效率的数据存取.ADO.NET的架构及新功能是为了能满足广泛的数据交换需求所产生出来的新技术,这就是ADO.NET

ADO.NET架构:

ADO.NET对象可以让我们快速简单的来存取各种数据.传统的主从式应用程序在执行时,都会保持和数据源的联机.但是在某些情况下和数据库一直保持联机是不需要的,而且一直保持和数据源的联机会浪费系统资源.有些时候我们只需要很单纯的将数据取回,这时候就不需要保持对数据源的联机.ADO.NET被设计成对于数据处理不一直保持联机的架构,应用程序只有在要取数据或是更新数据的时候才对数据源进行联机的工作,所以应用程序所要管理的连接减少:数据源就不用一直和应用程序保持联机,负载减轻了效能自然也就提升. 不过我们的应用程序也有些情况需要和数据源一直保持联机,例如在线订位系统;此时我们还是可以使用ADO对象和数据源随时保持联机的状态.

ADO.NET对象模型

ADO.NET对象模型中有五个主要的组件,分别是Connection对象,Command对象,DataSetCommand,DataSet以及DataReader.这些组件中负责联机和数据操作的部分我们称为**数据操作组件**,分别是Connection对象,Command对象,DataSetCommand对象和DataReader对象所组成.数据操作组件最主要是当做DataSet对象以及数据源之间的桥梁,负责将数据源中的数据取出后植入DataSet对象中，以及将数据存回数据源的工作。ADO.NET对象模型图如下所示：



Connection组件

Connection对象主要是开启程序和数据库之间的连接。没有利用连接对象将数据库打开，是无法从数据库中取得数据的。这个物件在ADO.NET的最底层，我们可以自己产生这个对象，或是由其他的对象自动产生。

Command组件

Command组件主要是用来对数据库发出一些指令，例如可以对数据库下达查询、新增、修改、删除数据等指令，以及**呼叫存在数据库中的预存程序等**。这个对象是架构在Connection对象上，也就是Command对象是透过连接到数据源的Connection对象来下命令的：所以Connection连接到哪个数据库，Command对象的命令就下到哪里。

DataSetCommand物件

DataSetCommand对象主要是在数据源以及DataSet之间执行数据传输的工作，它可以透过Command对象下达命令后，并将取得数据放入DataSet对象中。这个对象是架构在Command对象上，并提供了许多配合DataSet使用的功能。在如今的版本中该物件会更名为DataAdapter。

DataSet物件

DataSet这个对象可以视为一个暂存区（Cache），可以把从数据库中所查询到的数据保留起来，甚至可以将整个数据库显示出来。DataSet的能力不只是可以存储多个Table而已，还可以透过DataSetCommand对象取得一些例如主键等的数据表结构，并可以记录数据表间的关联。DataSet对象可以说是ADO.NET中重量级的对象，这个对象架构在DataSetCommand对象上，本身不具备和数据源沟通的能力：也就是说我们将DataSetCommand对象当做DataSet对象以及数据源之间传输数据的桥梁。

DataReader物件

当我们只需要循序的读取数据而不需要其他操作时，可以使用DataReader对象。DataReader对象只是一次一笔向下循序的读取数据源中的数据，而且这些数据是只读的，并不允许其他的操作。因为DataReader在读取数据的时候限制了每次只读取一笔，而且只能只读，所以使用起来不但节省资源而且效率很好。使用DataReader对象除了效率较好之外，因为不用把数据全部传回，故可以降低网络的负载。

（**以前的用法**）

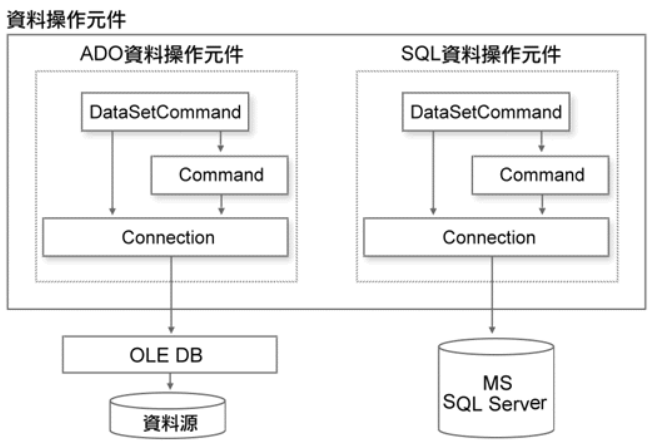
ADO.NET的数据操作组件（Managed Providers）

ADO.NET的数据存取和之前的版本不一样。前版的ADO存取数据的方式只有一种，那就是透过OLEDB来存取数据；而现在的ADO.NET则分为两种，

一种是直接存取SQL Server中的数据，

另一种是透过OLEDB来存取其他数据库中的数据。

前面我们提过：要存取数据源中的数据，要透过数据操控组件。由于我们可以选择OLEDB和资料源联机，或者是和SQL Server直接联机，所以ASP.NET提供了两组数据操作组件，分别 为ADO数据操作组件以及SQL数据操作组件。

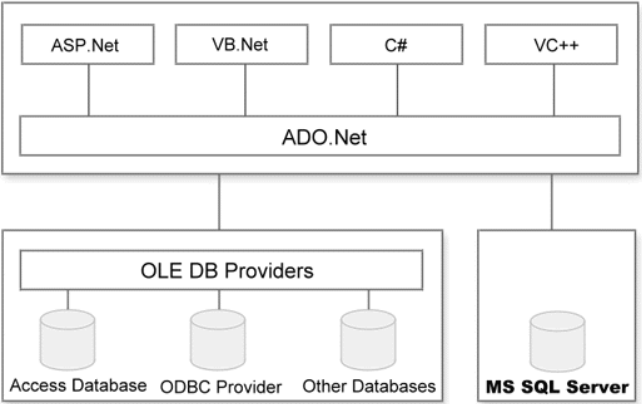


每组数据操作组件内都有Connection对象、Command对象、DataSetCommand对象及DataReader对象。为了容易分别这两组数据控制对象，我们将这四个对象分别加上前缀ADO以及SQL，如下表所示：



这两种数据操作组件虽然针对的数据源不一样，但是这些对象的架构都是一样的。（直接通过SQL类别对象来存取SqlServer中的数据效率相比于通过OLEDB来存取SqlServer中的资料更好。）

ADO.NET对于这两种数据存取方式所使用的对象完全不一样，在使用时必须特别注意。



OLEDB简介：

我们要开启如Access数据库中的数据，必须用ADO.NET透过OLEDB来开启。ADO.NET利用OLEDB来取得数据，这是因为OLEDB了解如何和许多种数据源做沟通，所以对OLEDB有相当程度的了解很重要。OLEDB最主要是由三个部分组合而成的：

Data Provider 数据提供者

凡是透过OLEDB将数据提供出来的，就是数据提供者。例如SqlServer数据库中的数据表，或是Access数据库档案等，都是Data Provider

Data Consumer 数据使用者

凡是使用OLEDB提供数据的程序或组件，都是OLEDB的数据使用者。换句话说，凡是使用ADO的应用程序或网页都是OLEDB的数据使用者。

Service Components 服务组件

数据服务组件可以执行数据提供者以及数据使用者之间数据传递的工作，数据使用者要向数据提供者要求数据时，是透过OLEDB服务组件的查询处理器执行查询的工作，而查询的结果则由指针引擎来管理。

System.Data名称地址

我们要使用ADO.NET，必须要先宣告System.Data这个名称地址。因为System.Data这个名称地址中包含大部分组成ADO.NET架构的基础对象类别。例如DataSet对象，数据表，字段，关联等。

System.Data.ADO 命名空间

我们要使用ADO数据操作组件来存取数据，必须宣告System,Data.ADO 这个名称地址。ADO数据操作组件是透过OLEDB和数据源联机，故System.Data.ADO这个命名空间定义了ADO数据操作组件的对象类别，例如ADOConnection对象，ADOCommand对象等。要使用ADO数据操作组件就一定要声明命名空间System.Data.ADO。

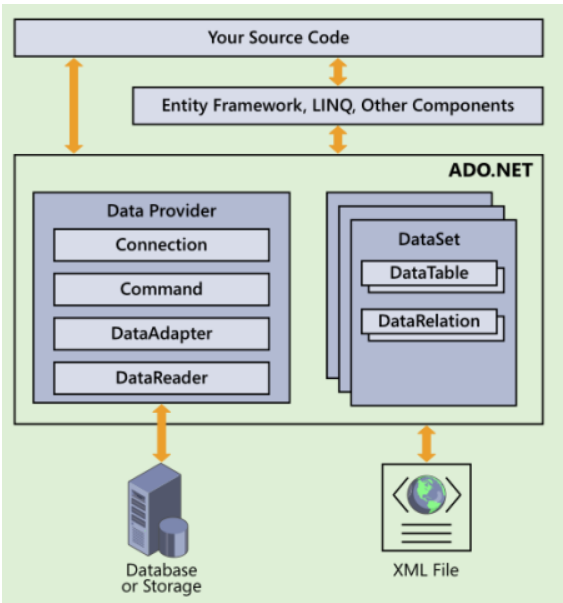
System.Data.SQL

同上一致，用于使用SQL操作组件的。

（**现在的用法**）

ADO.NET最核心的组成部分

System.Data命名空间提供了不同的ADO.NET类，它们既分工明确，又互相协作的提供表格数据的访问服务。该类库包括两组重要的类：一组负责处理软件内部的实际数据（DataSet），一组负责与外部数据系统通信（Data Provider）。具体架构如下图所示：



**DataSet**是ADO.NET的非连接（断开）结构的核心组件。DataSet的设计目的很明确：**为了实现独立于任何数据源的数据访问**。因此，ADO.NET结构可以用于多种不同的数据源，用于XML数据，或用于管理应用程序本地的数据。DataSet包含了一个或多个DataTable对象的结合，这些对象由数据行和数据列以及主键、外键、约束和有关DataTable对象中数据的关系信息组成。

ADO.NET结构的另一个核心元素是.NET数据提供程序（Data Provider）。具体包括：

* Connection对象提供与数据源的连接。
* Command对象使你能够访问用于返回数据、修改数据、运行存储过程以及发送或检索参数信息的数据库命令。
* DataReader对象从数据源中提供快速的，只读的数据流。
* DataAdapter对象提供连接DataSet对象和数据源之间的桥梁。DataAdapter使用Command对象在数据源中执行SQL命令，便将数据加载到DataSet中，并使对DataSet中数据的更改与数据源保持一致。

ADO.NET扩展

提供一致的数据访问，是使用ADO.NET的一个关键的优势。但是对于开发人员来说，更大的优势是通过ADO.NET**将管理的数据作为对象来做处理**。表中每个字段都是强类型成员，与.NET通用类型系统（Common Type System）完全兼容。个别的字段甚至可以作为局部变量来使用。数据行或者其他的数据集对象是标准的.NET集合（Collections），可以用标准的迭代方法处理。

Entity Framework和LINQ是微软为了提高ADO.NET核心功能而建立的两个新的工具。需要注意的是，它们**并不是ADO.NET的基本组成部分**。

Entity Framework利用了抽象化数据结构的方式，将每个数据库对象都转换成应用程序对象（entity），而数据字段都转换为属性（property），关系则转换为结合属性（association），让数据库的E/R模型完全转换为对象模型。而在抽象化的结构之下，则是高度集成与对应结构的概念层、对应层、和储存层，以及支持Entity Framework的数据提供者（provider），让数据访问的工作得以顺利与完整的进行。

LINQ允许C#或者VB代码以查询数据库相同的方式操作内存数据。LINQ是一个通用的数据工具，可以让你非常容易的融合不同数据源的数据，并得到单一的数据结果集。

.NET数据提供程序

我们知道ADO.NET的两大核心组件分别是Data Provider和 Dataset。Data Provider提供了外部访问数据数据源的可能性。

1. 什么是.NET数据提供程序

.NET Framework数据提供程序**用于连接数据库、执行命令和检索结果**。这些结果将被直接处理，放置在DataSet中以便根据需要向用户公开、与多个源中的数据组合，或在层之间进行远程处理。.NET Framework 数据提供程序是轻量的，它在数据源和代码之间创建最小的分层，并在不降低功能性的情况下提高性能。

下表列出了.NET Framework中包含的数据提供程序。



1. .NET 数据提供程序的核心对象

在上述文字中，我们知道Connection对象、Command对象、DataReader对象以及DataAdapter对象构成了.NET数据提供程序的骨架。这四个对象非常重要。

1. 其他重要的对象
   1. Parameter对象

简单的讲，**Parameter对象定义了命令和存储过程的输入、输出和返回值参数**。

先看这样一段代码，

strSQL = “SELECT \* FROM users WHERE (name = ‘ ” + username + ” ’) and (pw = ‘ ” + password + ” ’);”

这是一个简单的登陆查询验证代码。

但是如果用户（一些自称“黑客”的高级用户）填入

username = “ ‘ OR ‘1’ = ‘1 ”;

passWord = “ ‘ OR ‘1’ = ‘1 ”;

接下来网站居然通过验证并登陆成功了。

分析：

我们将这两个变量填入命令后可以看到

strSQL = “ SELECT \* FROM users WHERE (name = ‘’ OR ‘1’ = ‘1’) and (pw = ‘’ OR ‘1’=’1’); ”

也就是实际上运行的SQL命令会变成下面的样子：

strSQL = “ SELECT \* FROM users ;”

这样这个验证语句就形同虚设了。上面的情况，用专业术语来说就是一个简单的SQL注入（SQL injection）。有SQL注入的出现，因此就有参数化查询的出现（Parameterized Query）。

**参数化查询是指在设计与数据库连接并存取资料时，在需要填入数值或资料的地方，使用参数（Parameter）来给值**，这个方法目前已被视为最有效可预防SQL注入的攻击方法的防御模式。

在使用参数化查询的情况下，数据库服务器不会将参数的内容视为SQL指令的一部分来处理，而是在数据库完成SQL指令的编译后，才套用参数执行，因此就算参数中含有破坏性的指令，也不会被数据库执行。

Parameter对象有两个非常重要的属性：DBType和Value。DBType用来设置或获取参数的类型，Value则用来设置或获取参数的值。

下例为用Parameter对象来改写的简单登录验证代码：

1 strSQL = "SELECT \* FROM users WHERE Name = @Name and Password = @Password";  
2 SqlParamter[] paras = new SqlParamter[]{//参数数组  
3 new SqlParamter("@Name",SqlDBType.Varchar,**50**)  
4 new SqlParamter("@Password",SqlDBType.Varchar,**50**)};  
5 paras[0].value = userName;//绑定用户名  
6 paras[1].value = password;//绑定用户密码

* 1. ConnectionStringBuilder和CommandBuilder
* ConnectionStringBuilder：它提供了一种用于创建和管理由Connection对象使用的连接字符串的内容的简单方法。所有ConnectionStringBuilder对象的基类均为DBConnectionStringBuilder类。
* CommandBuilder：它自动生成DataAdapter的命令属性或从存储过程中派生参数信息，并填充Command对象的Parameters集合。所有CommandBuilder对象的基类均为DBCommandBuilder类。

1. 理解.NET 数据提供程序
   1. 用于SQL Server的.NET Framework数据提供程序（SqlClient）

用于SQL Server的.NET Framework数据提供程序使用自己的协议与SQL Server进行通信。它是轻量级的且性能良好，可直接访问SQL Server，而无需添加OLEDB或ODBC层。用于OLEDB的.NET Framework数据提供程序通过OLEDB组件（它提供连接池和事务服务）和用于数据源的OLEDB访问接口与OLEDB数据源进行通信。

若要使用用于SQL Server的.NET Framework数据提供程序，你必须具有对SQL Server7.0或更高版本的访问权限。用于SQL Server类的.NET Framework数据提供程序位于System.Data.SqlClient命名空间中。

用于SQL Server的.NET Framework数据提供程序支持本地事务和分布式事务。对于分布式事务，默认情况下，用于SQL Server的.NET Framework数据提供程序会自动登记在事务中，并自动从Windows组件服务或System.Transcations获取事务详细信息。

* 1. 用于OLEDB的.NET Framework数据提供程序

OleDb通过COM互操作使用本机的OLEDB来启用数据访问。用于OLEDB的数据提供程序支持本地事务和分布式事务。

* 1. 用于ODBC的.NEY Framework数据提供程序

Odbc使用本机ODBC驱动程序管理器来启用数据访问。

* 1. 用于Oracle的.NET Framework 数据提供程序

用于Oracle的.NET Framework数据提供程序（OracleClient）通过Oracle客户端连接软件启用对Oracle数据源的数据访问。该数据提供程序支持Oracle客户端软件8.1.7版或更高。用于Oracle的.NET Framework数据提供程序要求系统上安装有Oracle客户端软件，才能链接到Oracle数据源。

1. 应用程序或者数据源不同，我们就需要选择不同的.NET 数据提供程序。



**连接字符串**

ADO.NET强大的优势在于对不同的数据源提供一致的访问。这与连接字符串的存在少不了关系。

1. 什么是连接字符串

我们已经知道，ADO.NET类库为不同的外部数据源提供了一致的访问。这些数据源可以是本地的数据文件（如Excel，TXT，Access，甚至是sqlite），也可以是远程的数据库服务器（如SQL Server、MySQL、Oracle等）。数据源似乎琳琅满目。

ADO.NET如何能够准确而又高效的访问到不同的数据源呢？

你或许已经清楚，ADO.NET已经为不同的数据源编写了不同的数据提供程序。但是这个前提是，我们得访问到正确的数据。否则的话，只会“张冠李戴”。就好比，我们SQL Server数据提供程序去处理Excel数据源，结果肯定是不行的。

连接字符串，就是这样一组被格式化的键值对：它告诉ADO.NET数据源在哪里，需要什么样的数据格式，提供什么样的访问信任级别以及其他任何包括连接的相关信息。

1. 理解语法格式

连接字符串由一组元素组成，一个元素包含一个键值对，元素之间由“；”分开。

key1=value1;key2=value2;key3=value3…

典型的元素（键值对）应当包含这些信息：

* 数据源是基于文件的还是基于网络的数据库服务器
* 是否需要账号密码来访问数据源
* 超时的限制是多少，以及其他相关的配置信息

1. 列举几种典型连接字符串
   1. SQL Server连接字符串
2. 标准的安全连接



说明：

Data Source：需要连接的服务器。需要注意的是，如果使用的是Express版本的SQL Server需要在服务器名后加\SQLEXPRESS 例如，连接本地的SQL Server2008 Express版本的数据库服务器，可以写成Data Source = (local)\SQLEXPRESS

Initial Catalog: 默认使用的数据库名称

User ID:数据库服务器账号。

PassWord：数据库服务器密码。

或者也可以写成这样：



1. 可信连接



Intergrate Security：使用存在的Windows安全证书访问数据库。

也可以这样写：



* 1. Access连接字符串



* 1. MySQL连接字符串



* 1. DB2连接字符串



* 1. Oracle连接字符串



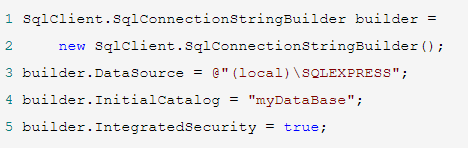
1. 如何构造连接字符串？

连接字符串本质就是一个字符串，所以我们完全可以用



来构造一个连接字符串。实际上，ADO.NET有一个专门的类来处理连接字符串：DBConnectionStringBuilder。

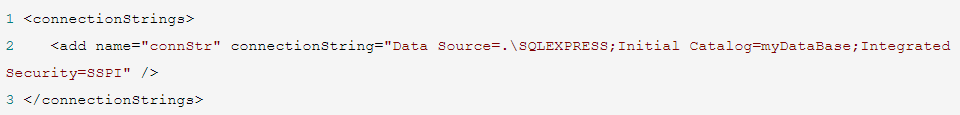
DbConnectionStringBuilder类为强类型连接字符串生成基类。之所以有这样一个类，是因为它更加安全友好。以SQL Server为例，可以构建一个连接字符串。



1. 在配置文件中存储连接字符串

在我们实际开发中，不会把连接字符串直接写在代码中，而是存储在配置文件中。把连接字符串写死在代码中，不便于维护，每次修改字符串时，还得重新编译代码。

以ASP.NET应用程序为例，我们一般把连接字符串写在web.config配置文件的<connectionstrings>节点。



因此，我们只需要在程序中添加相应代码来获取配置文件中的值即可。

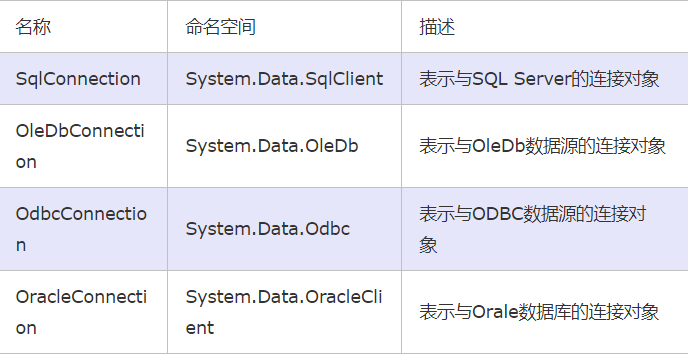


当然，你可以在配置文件中的<appsettings>节点下添加连接字符串，获取字符串值的方法类似。

**Connection对象**

1. 理解Connection

Connection对象，顾名思义，表示与特定数据源的连接。如果把数据源比作大门，那么连接字符串则是钥匙，而连接对象则是拿着钥匙开门的人。对于ADO.NET而言，不同的数据源，都对应着不同的Connection对象。具体Connection对象如下表所示：



不管哪种连接对象，它都继承与DBConnection类。我们看看DBConnection类的实现结构：



从上面，可以看出DbConnection是抽象基类，并且继承Component，IDbConnection，IDisposable类。由于DbConnection类是抽象基类，因此它不能实例化。DbConnection类封装了很多重要的方法和属性。

1. 必须掌握的几个方法

Open：使用ConnectionString所指定的设置打开数据库连接

Dispose：释放由Component使用的所有资源。

Close：关闭与数据库的连接。此方法是关闭任何已打开连接的首选方法。Close方法回滚任何挂起的事务。然后，他将连接释放到连接池，或在连接池被禁用的情况下关闭连接。

1. 必须掌握的几个属性：

DataBase：在连接打开之后获取当前数据库的名称，或者在连接打开之前获取连接字符串中指定的数据库名。

DataSource：获取要连接的数据库服务器的名称。

ConnectionTimeOut：获取在建立连接时终止尝试并生成错误之前所等待的时间

ConnectionString：获取或设置用于打开连接的字符串

State：获取描述连接状态的字符串

1. 说说ConnectionState

State属性描述了与数据源的连接的当前状态。ConnectionState是一个枚举类型。它包括以下成员：

Closed：连接处于关闭状态。

Open：连接处于打开状态。

Connecting：连接对象正在与数据源连接。

Executing：连接对象正在执行命令。

Fetching：连接对象正在检索数据。

Broken：与数据源的连接中断。

1. 实例：连接SQL Server的SqlConnection对象
2. **编写优雅而又安全的代码**
3. 添加try。。。catch代码

我们知道连接数据库时，可能会出现异常，因此要添加异常捕获的代码。对于C#来说，典型的异常处理是添加try…catch代码块。Finally是可选的。异常我们应当确保打开连接后，无论是否出现异常，都应该关闭连接和释放资源。所以，在finally语句块中调用close方法关闭数据库连接。



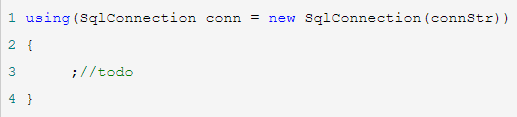
1. 使用using语句

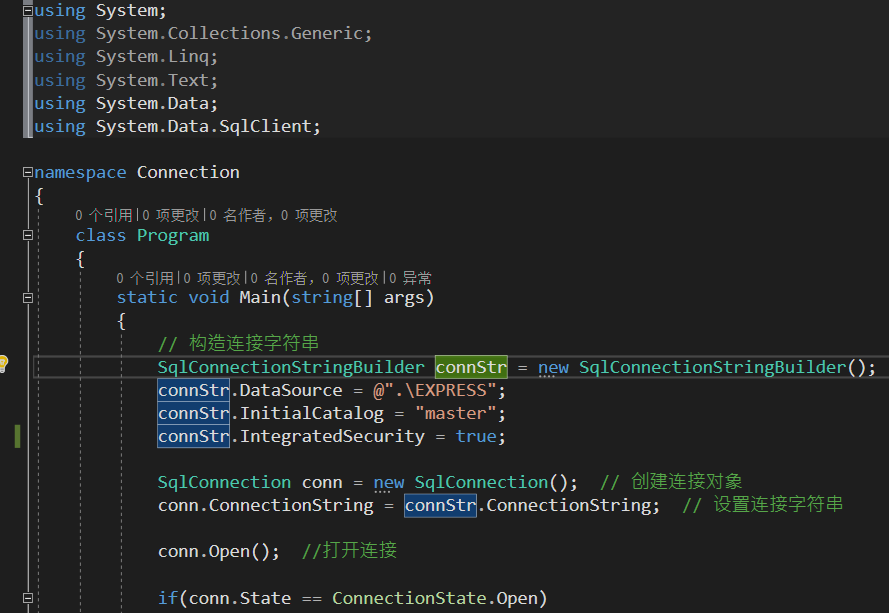
另外一种优雅的方法，则是使用using语句。**Using语句的作用是确保资源使用后，并很快释放它们**。Using语句帮助减少意外的运行时错误带来的潜在问题，它整洁地包装了资源的使用。

具体来说，它执行了一下内容：

* 分配资源
* 把statement放进try块
* 创建资源的Dispose方法，把它放进finally块。

因此，上面的语句等同于：







外记：

Using实质:

在程序编译阶段，编译器会自动将using语句生成try-finally语句，并在finally块中调用对象的Dispose方法，来清理资源。所以，using语句等效于try-finally语句：

例如：

using( Font f2 = new Font(“Arial”, 10, FontStyle.Bold))

{

font2.F();

}

被编译器编译为：

Font f2 = new Font(“Arial”, 10, FontStyle.Bold);

try

{

font2.F();

}

finally

{

If (f2 != null) ((IDisposable)f2).Dispose();

}

由上可知：

using只能用于实现了IDisposable接口的类型，禁止为不支持IDisposable接口的类型使用using语句，否则会出现编译错误。

Using语句适用于清理单个非托管资源的情况，而多个非托管对象的清理最好以try-finally来实现，因为嵌套的using语句可能存在bug。

**数据库连接池**

1. 什么是连接池

建立一个数据库连接是一件非常耗时（消耗时间）耗力（消耗资源）的事情。之所以会这样，是因为连接到数据库服务器需要经历几个漫长的过程：建立物理通道（例如套接字或命名管道），与服务器进行初次握手，分析连接字符串信息，由服务器对连接进行身份验证，运行检查以便在当前事务中登记等等。既然新建一条连接那么痛苦，那么为什么不重复利用已有的连接呢。

实际上，ADO.NET已经为我们提供了名为连接池的优化方法。

连接池就是这样一个容器：它存放了一定数量的与数据库服务器的物理连接。

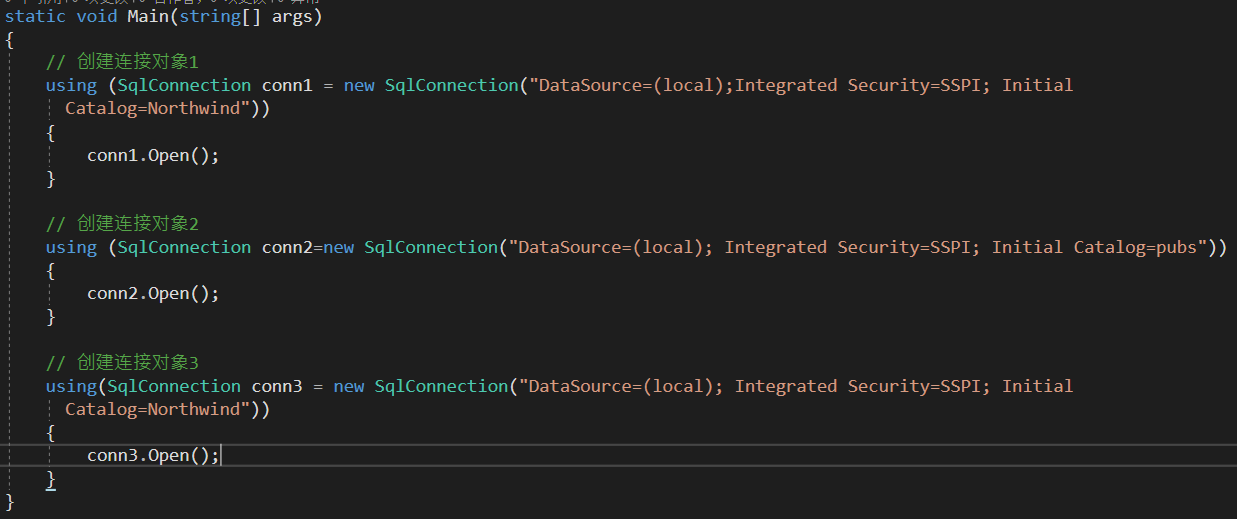
因此，当我们需要连接数据库服务器的时候，只需去池（容器）中取出一条空闲的连接，而不是新建一条连接。这样的话，我们就可以大大减少连接数据库的开销，从而提高了应用程序的性能。

1. 连接池的工作原理
   1. 创建连接池

需要说明的是，连接池是具有类别区分的。也就是说，同一时刻同一程序域可以有多个不同类型的连接池。那么，连接池是如何标识区分的？细致的讲，是由进程，应用程序域，连接字符串以及Windows标识（在使用集成的安全性时）共同组成签名来标识区分的。

但是对于同一应用程序来说，一般只由连接字符串来标识区分。当打开一条连接时，如果该条连接的类型签名与现有的连接池类型不匹配，则创建一个新的连接池。反之，则不创建新的连接池。

一个典型的创建连接的实例：



上面实例中创建了三个SqlConnection对象，但是管理时只需要两个连接池。因为conn1与conn3的连接字符串相同，所以可以共享一个连接池，而conn2与conn3不同，所以需要创建新的连接池。

* 1. 分配空闲连接

当用户创建连接请求或者说调用Connection对象的Open时，连接池管理器首先需要根据连接请求的类型签名找到匹配类型的连接池，然后尽力分配一条空闲连接。具体情况如下：

* 如果池中有空闲连接可用，返回该连接。
* 如果池中连接都已用完，创建一个新连接添加到池中。
* 如果池中连接已达到最大连接数，请求进入等待队列直到有空闲连接可用。
  1. 移除无效连接

无效连接，即不能正确链接到数据库服务器的连接。对于连接池来说，存储的与数据库服务器的连接的数量是有限的。因此，对于无效连接，如果不及时移除，将会浪费连接池的空间。其实你不用担心，连接池管理器已经很好的帮我们处理了这些问题。如果连接长时间空闲，或检测到与服务器的联机已断开，连接池管理器会将该连接从连接池中移除。

* 1. 回收使用完的连接

**当我们使用完一条连接时，应当及时关闭或释放连接，以便连接可以返回池中重复利用**。我们可以通过Connection对象的Close或Dispose方法，也可以通过C#的using语句来关闭连接。

1. 说说几个重要的属性

**连接池的行为可以通过连接字符串来控制**。主要包括四个重要的属性：

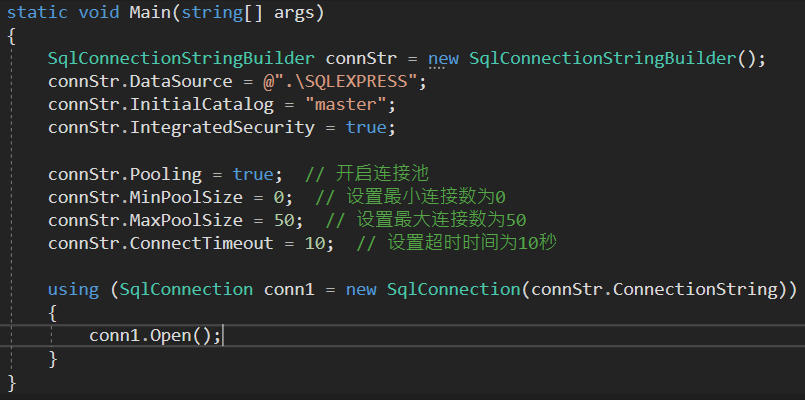
**Connection Timeout：**连接请求等待超时时间。默认15秒，以秒为单位

**Max Pool Size：**连接池中最大连接数。默认为100

**Min Pool Size：**连接池中最小连接数。默认为0

**Pooling：**是否启用连接池。**ADO.NET默认是启用连接池的。**你也可以手动关闭连接池。

下面来看一个实例来理解连接池属性：



1. 连接异常与处理方法

当用户打开一个连接而没有正确或者及时的关闭时，经常会引发“连接泄漏”问题。泄漏的连接，会一直保持打开的状态，直到调用Dispose方法，垃圾回收器（GC）才关闭和释放连接。与ADO不同，ADO.NET需要手动的关闭使用完的连接。一个重要的**误区**是：当连接对象超出局部作用域范围时，就会关闭连接。实际上，当超出作用域时，释放的只是连接对象而非连接资源。

因此必须牢记使用完的连接应当尽快的正确的关闭和释放。

1. 高效的使用连接池的基本原则

用好链接池会 大大提高应用程序的性能。相反，如果使用不当的话，则百害而无一利。一般来说，应当遵循以下原则：

* 在最晚的时刻申请链接，在最早的时候释放连接
* 关闭连接时先关闭相应用户定义的事务。
* 确保并维持连接池至少一个打开的连接
* 尽力避免池碎片的产生。主要包括集成安全性产生的池碎片以及使用许多数据库产生的池碎片。

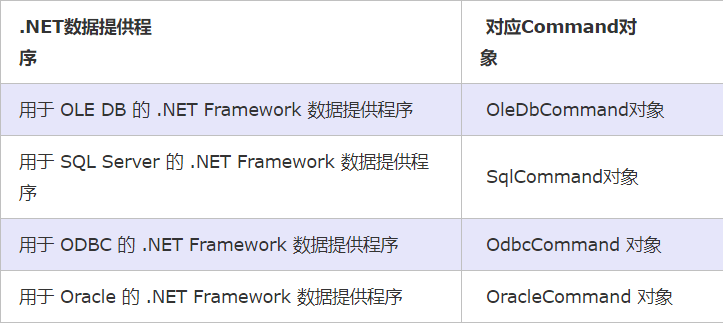
池碎片：是许多web应用程序中的一个常见的问题，应用程序可能会创建大量在进程退出后才会释放的池。这样，将打开大量的连接，占用许多内存，从而导致性能降低。

Command对象和数据检索

1. 什么是Command对象

我们知道ADO.NET最主要的目的是对外部数据源提供统一的访问。而访问数据源数据，就少不了增删改查等操作。尽管Connection对象已经为我们连接好了外部数据源，但它并不提供对外部数据源的任何操作。这些就需要通过Command对象来实现。

它封装了所有对外部数据源的操作（包括增、删、改、查等SQL语句与存储过程），并在执行完成后返回合适的结果。与Connection对象一样，对于不同的数据源，ADO.NET提供了不同的Command对象。



不管是哪种Command对象，它都继承与DBCommand类。与DBConnection类一样，DBCommand类也是抽象基类，不能被实例化，并期待派生类来实现方法。DBCommand类定义了完善的健全的数据库操作的基本方法和基本属性。

它的结构如下：

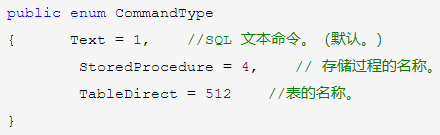
public abstract class DbCommand: Component, IDbCommand, IDisposable

从上面我们可以知道，它继承了Component类以及IDBCommand接口和IDisposable接口。

1. 必须掌握的几个属性

CommandText：获取或设置对数据源执行的文本命令。默认为空字符串。

CommandType：命令类型，指示或指定如何解释CommandText属性。CommandType属性的值是一个枚举类型，定义结构如下：



需要注意的是，将CommandType设置为StoredProcedure时，应将CommandText属性设置为存储过程的名称。当调用Execute方法之一时，该命令将执行此存储过程。

Connection：设置或获取与数据源的连接

Parameters:绑定SQL语句或存储过程的参数。参数化查询中不可或缺的对象

Tranction：获取或设置在其中执行.NET Framework数据提供程序的Command对象的事务。

1. 必须掌握的几个方法

ExecuteNonQuery：执行不返回数据行的操作，并返回一个int类型的数据。

注意：对于UPDATE、INSERT和DELETE语句，返回值为该命令所影响的行数。对于其他所有类型的语句，返回值为-1.

ExecuteReader:执行查询,并返回一个DataReader对象。

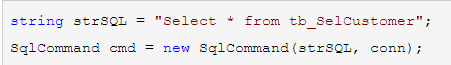
ExecuteScalar：执行查询，并返回查询结果集中第一行的第一列（object类型）。如果找不到结果集中的第一行第一列，则返回null引用。

1. 如何创建Command对象？

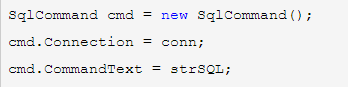
在创建Command对象之前，你需要明确两件事情：（1）你要执行什么样的操作？（2）你要对哪个数据源进行操作？

我们可以用string字符串来构造一条SQL语句，也可以通过Connection对象指定连接的数据源。那么我们如何将这些信息交给Command对象呢？

1. 通过构造函数。



1. 通过Command对象的属性。



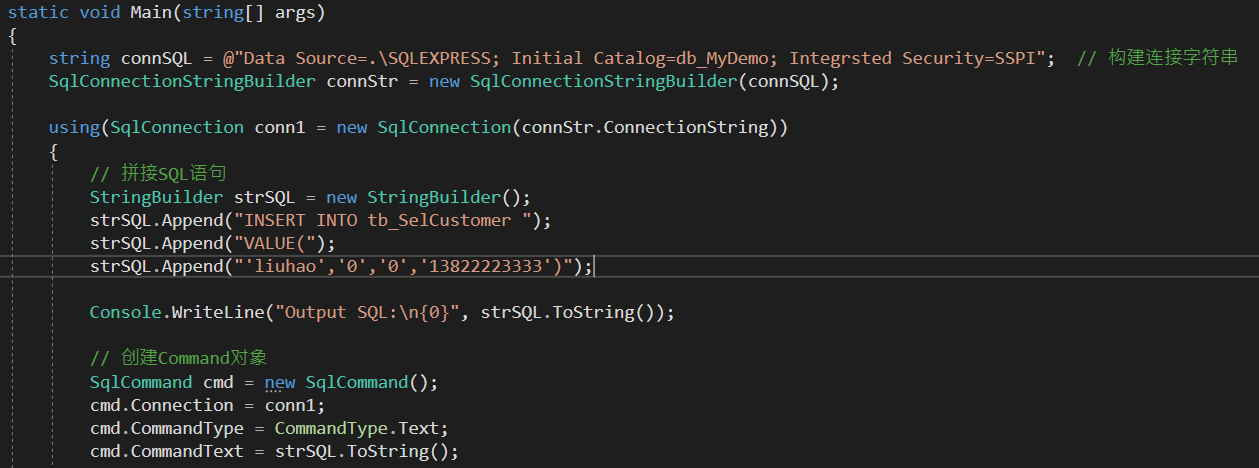
1. 选择合适的执行命令

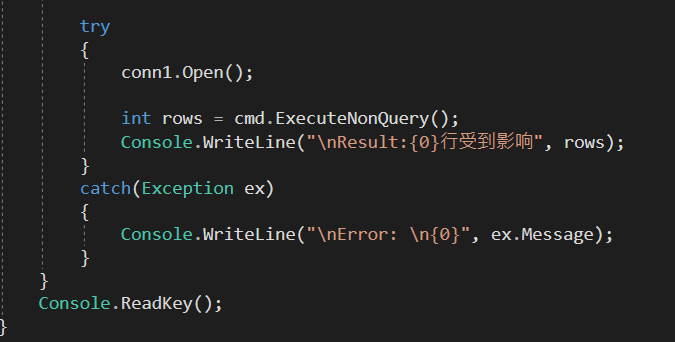
Command对象提供了丰富的执行命令操作。根据不同的场景选择合适的执行命令。

1. 场景一：执行CRUD操作，不返回数据行，返回影响的行数

当我们对数据表的行进行增加，删除，更新操作或者处理数据定义语句（比如create table来创建表结构）时，实际上数据库是不返回数据行的，仅仅返回一个影响行数信息的整数。

我们在tb\_SelCustomer表中插入一行记录。



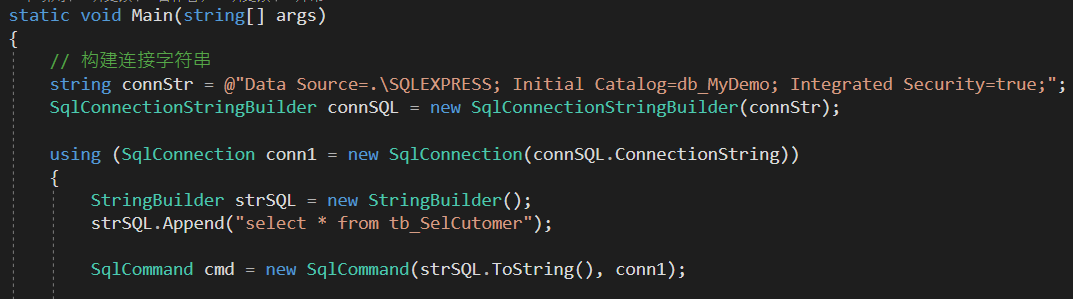


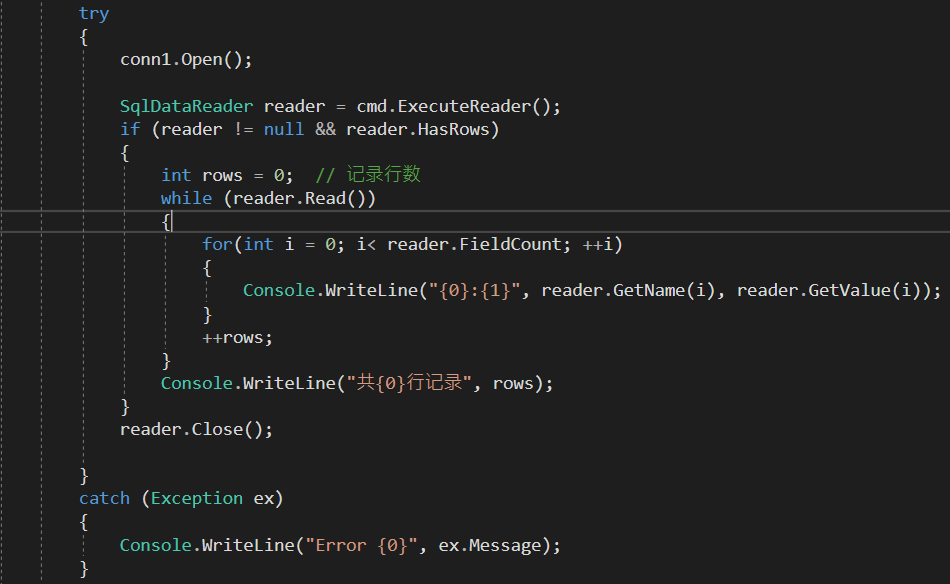
1. 场景二：执行Select操作，返回多个数据

当我们通过执行Select操作返回一行或多行数据时，这时候ExecuteNonQuery就需要休息了。处理这种事情就需要请ExecuteReader方法上场了。ExcuteReader返回一个DataReader对象。DataReader是一个快速的、轻量级、只读的遍历访问每一行数据的数据流。使用DataReader时，需要注意一下几点：

* DataReader一次遍历一行数据，并返回一个包含列名称集合。
* 第一次调用Read()方法获取第一行数据，并将游标指向下一行数据。当再次调用该方法的时候，将读取下一行数据。
* 当检测到不再有数据行时，Read()方法将返回false
* 通过HasRows属性，我们知道查询结果中是否有数据行。
* 当我们使用完DataReader时，一定要注意关闭。SQL Server默认只允许打开一个DataReader。

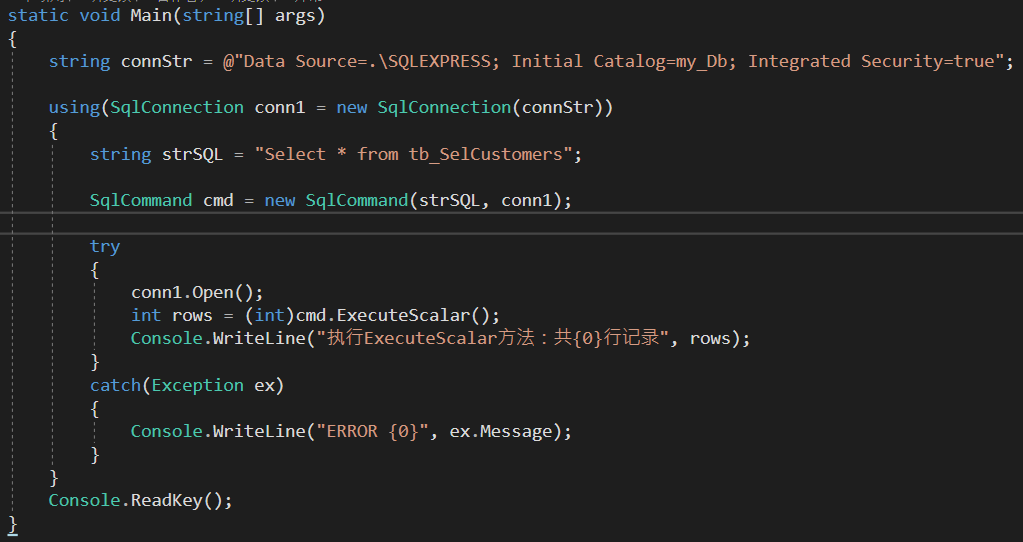
查询出tb\_SelCustomer表中所有的数据。代码如下：





1. 场景三：执行select操作，返回单个值

ExecuteScalar方法就是处理单个数据最优秀的工具。ExecuteScalar返回一个System.object类型的数据，因此我们在获取数据时需要进行强制类型转换。



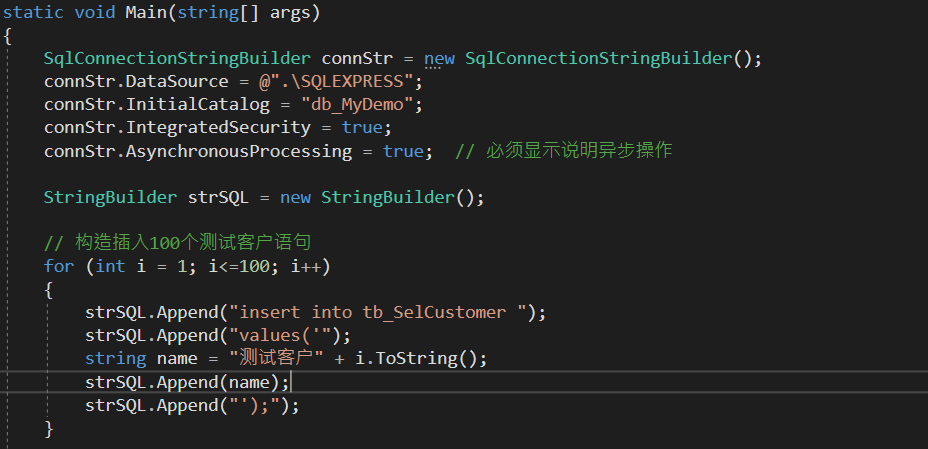
Command对象高级应用

1. 异步执行命令

在ADO.NET2.0版本之前，执行Command对象命令时，需要等待命令完成才能执行其他操作。比如，执行ExecuteNonQuery（）方法，应用程序将会保持阻塞，直到数据操作成功完成或者异常终止以及连接超时。在ADO.NET2.0版本引入异步执行特性，显然，ADO.NET更稳健了，更完美了。

异步执行的根本思想是，在执行命令操作时，无需等待命令操作完成，可以并发的处理其他操作。ADO.NET提供了丰富的方法来处理异步操作，BeginExecuteNonQuery和EndExecuteNonQuery就是一对经典的为异步操作服务的方法。BeginExecuteNonQuery方法返回System.IAsyncResult接口对象。我们可以根据IAsyncResult的IsCompleted属性来轮询命令是否执行完成。来看一个简单的实例。

我们将在tb\_SelCustomer中插入100行数据，并计算执行时间。





1. 使用参数化查询

在ADO.NET中，查询语句是以字符串的形式传递给外部数据库服务器的。这些字符串不仅包含了基本命令关键字，操作符，还包含了限制查询的数值。与其他编程语言不同，.NET 是基于强类型来管理查询字符串数据的。通过提供类型检查和验证，命令对象可使用参数来将值传递给SQL语句或存储过程。与命令文本不同，参数输入被视为文本值，而不是可执行代码。这样可帮助抵御SQL注入攻击。参数化命令还可以提高查询执行性能，因为它们可帮助数据库服务器将传入命令与适当的缓存查询计划进行准确匹配。

对不同的数据源来说，Parameter对象不同，但都派生自Db