7/18

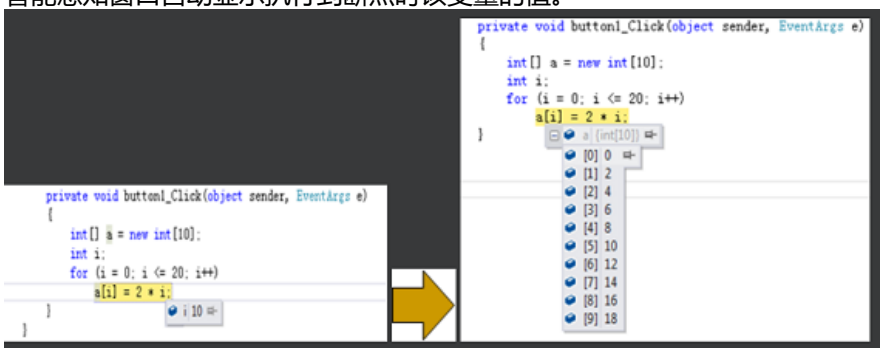
笔记本: 暑期实习
创建时间: 2021/7/18 9:38 **更新时间:** 2021/7/18 10:57
作者: 134exetj717
URL: http://wiki.suncaper.net/pages/viewpage.action?pageId=39855901

- C#错误分类

    1. 语法错误，也称编译错误：工具->选项->文本编辑器/常规->C#可以进行相应错误编辑
    2. 运行错误：如数据一处、数组下表越界，当出现栈溢出时一般就是该类错误，需检查数组的使用
    3. 逻辑错误

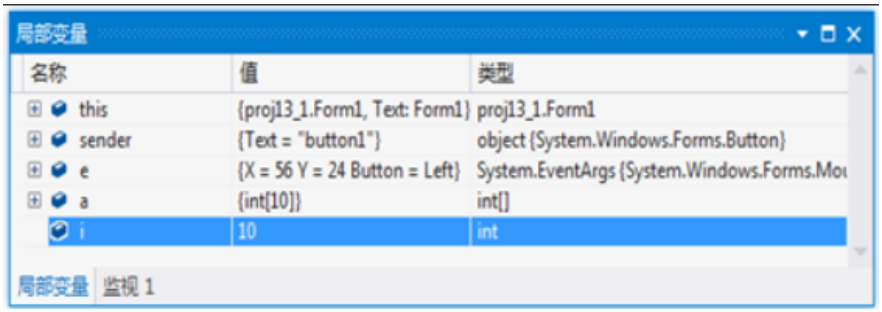
- 程序调试方法
    - 调试工具栏
    - 设置断点，F9；逐行，F11；逐段，F10
    - 调试窗口：

        1. 智能感知窗口： 将鼠标放在希望观察的执行过语句的变量上，调试器会通过智能感知窗口自动显示执行到断点时该变量的值。

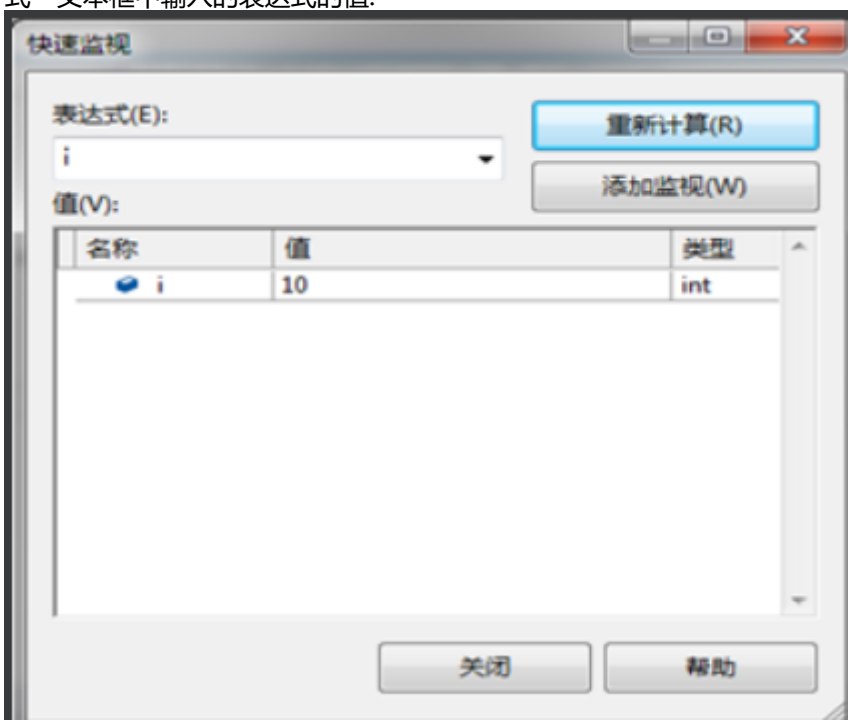


        2. **即时窗口**。此时选择“调试|窗口|即时”命令，出现即时窗口。可以输入“**?变量或表达式**”来显示变量或表达式的值。如下图所示。

        3. **局部变量窗口**。此时选择“调试|窗口|局部变量”命令，出现局部变量窗口，它自动显示当前过程中所有的变量值，如下图所示。



4. **快速监视窗口**。此时在某个对象上或空白处单击鼠标右键，从弹出的快捷菜单中选择“快速监视”命令，出现快速监视窗口，它用于显示用户在“表达式”文本框中输入的表达式的值.



- 异常处理
  - 什么是异常？： 指在程序执行期间发生的错误或意外情况，例如整数除零错误或内存不足警告时，就会产生一个异常。
  - 为什么需要异常处理？：
    - 如果给定异常没有异常处理程序，则程序将停止执行，并显示一条错误信息，因此对程序中的异常处理是非常重要的，一般情况下，在一个比较完整的程序中要尽可能考虑可能出现的各种异常，这样当发生异常时，控制流将立即跳转到关联的异常处理程序（如果存在）。

```
1. try ... catch语句
try
{
    //可能产生异常的程序代码
}
catch (异常类型1 异常类对象1)
{
    //处理异常类型1的异常控制代码
}
//...
catch (异常类型n 异常类对象n)
{
    //处理异常类型n的异常控制代码
}
例如:
using System;
```

```
namespace proj13_2
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 5, y = 0;
            try                                 //try...catch语句
            {
                x = x / y;                //引发除零错误
            }
            catch (Exception err)    //捕捉该错误
            {
                Console.WriteLine("{0}", err.Message);   //显示错误信息
            }
        }
    }
}
```

2. try ... catch ... finally语句
同try ... catch语句相比，try ... catch ... finally语句增加了一个finally块，其作用是不管是否发生异常，即使没有catch块，都将执行finally块中的语句，也就是说，finally块始终会执行，而与是否引发异常或者是否找到与异常类型匹配的catch块无关。其余与try%catch语句相同。
finally块通常用来释放资源，而不用等待由运行库中的垃圾回收器来终结对象。
归纳起来，在try块的代码出现异常后，其处理过程的顺序如下：
* try块在发生异常的地方中断程序的执行。
* 如果有catch块，就检查该块是否与已发生的异常类型匹配。
* 如果有catch块，但它与已发生的异常类型不匹配，就检查是否有其他catch块。
* 如果有catch块与已发生的异常类型匹配，就执行它包含的代码，再执行finally块（如果有）。
* 如果所有catch块都与已发生的异常类型不匹配，就执行finally块（如果有）。
【例3】 创建一个控制台应用程序Proj12-3项目，说明finally块的作用。

```
using System;
namespace proj13_3
{
    class Program
    {
        static void Main(string[] args)
        {
            int s = 10, i; int[] a = new int[5] { 1, 2, 3, 0, 4 };
            try
            {
                for (i = 0; i < a.Length; i++)
                    Console.Write("{0} ", s / a[i]);
                Console.WriteLine();
            }
            catch (Exception err)
            {
                Console.WriteLine("{0}", err.Message);
            }
            finally
            {
                Console.WriteLine("执行finally块");
            }
        }
    }
}
```

3. throw语句
throw语句有两种使用方式：
（1）直接抛出异常；
（2）在出现异常时，通过含有catch块对其进行处理并使用throw语句重新把这个异常抛出并让调用这个方法的程序进行捕捉和处理。throw语句的使用语法格式如下：

```
throw [表达式];
```

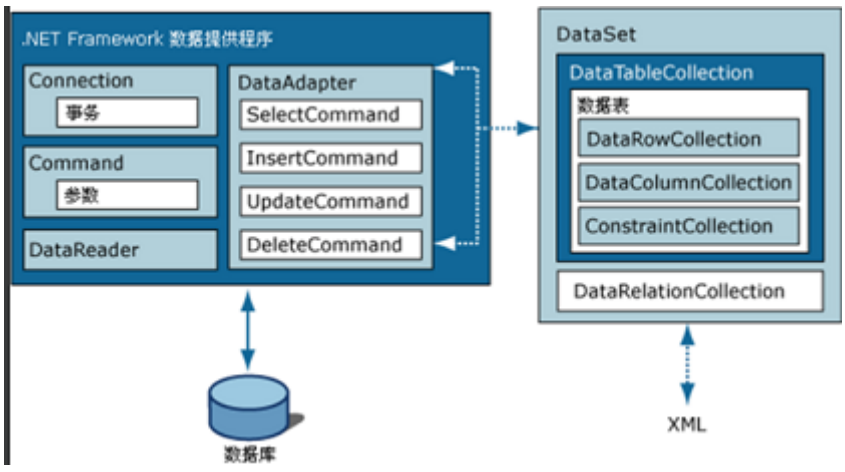
其中“表达式”类型必须是System.Exception或从System.Exception派生的类的类型。

throw语句也可以不带“表达式”，此时只能用在catch块中，在这种情况下，它重新抛出当前正在由catch块处理的异常。

【例4】创建一个控制台应用程序proj13-5项目，说明throw语句的作用。

```
using System;
using System.Collections.Generic;
using System.Text;
namespace proj13_5
{
    class Program
    {
        static void fun()
        {
            int x = 5, y = 0;
            try                          //try...catch语句
            {
                x = x / y;                  //引发除零错误
            }
            catch (Exception err)    //捕捉该错误
            {
                Console.WriteLine("fun:{0}", err.Message);
                throw;                   //重新抛出异常
            }
        }
        static void Main(string[] args)
        {
            try
            {
                fun();
            }
            catch (Exception err)    //捕捉该错误
            { Console.WriteLine("Main:{0}", err.Message); }
        }
    }
}
```

- ADO.NET数据库的访问技术
  - 定义：ADO.NET是在.NET Framework上访问数据库的一组类库，它利用.NET Data Provider（数据提供程序）以进行数据库的连接与访问。通过ADO .NET，数据库程序设计人员能够很轻易地使用各种对象来访问符合自己需求的数据库内容。
  - 体系结构



1. .NET Data Provider

对象名称	功能说明
Connection	提供和数据源的连接功能。
Command	提供运行访问数据库命令，传送数据或修改数据的功能，例如运行SQL命令和存储过程等。
DataAdapter	是DataSet对象和数据源间的桥梁。DataAdapter使用4个Command对象来运行查询、新建、修改、删除的SQL命令，把数据加载到DataSet，或者把DataSet内的数据送回数据源。
DataReader	通过Command对象运行SQL查询命令取得数据流，以便进行高速、只读的数据浏览。

2. **DataSet（数据集）：是ADO .NET离线数据访问模型中的核心对象，主要使用时机是在内存中暂存并处理各种从数据源中所取回的数据。DataSet其实就是一个存放在内存中的数据暂存区，这些数据必须通过DataAdapter对象与数据库进行数据交换。在DataSet内部允许同时存放一个或多个不同的数据表（DataTable）对象。**

- 访问流程

（1）建立Connection对象，创建一个数据库连接。

（2）在建立连接的基础上可以使用Command对象对数据库发送查询、新增、修改和删除等命令。

（3）创建DataAdapter对象，从数据库中取得数据。

（4）创建DataSet对象，将DataAdapter对象填充到DataSet对象（数据集）中。

（5）如果需要，可以重复操作，一个DataSet对象可以容纳多个数据集合。

（6）关闭数据库连接。

（7）在DataSet上进行所需要的操作。数据集的数据要输出到窗体中或者网页上面，需要设定数据显示控件的数据源为数据集。

- ADO.NET的数据访问对象
- OleDbConnection对象：在数据访问中首先必须是建立数据库的物理连接。.NET Data Provider使用OleDbConnection类的对象标识与一个数据库的物理连接。

1.1 OleDbConnection类

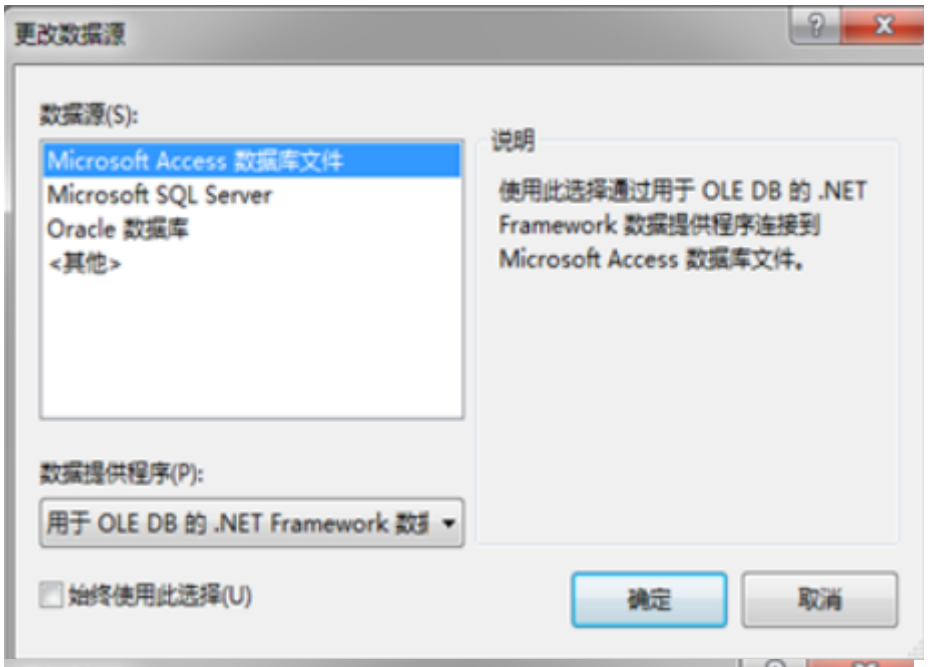
OleDbConnection类的属性	说　　明
ConnectionString	获取或设置用于打开数据库的字符串。
ConnectionTimeout	获取在尝试建立连接时终止尝试并生成错误之前所等待的时间。
Database	获取当前数据库或连接打开后要使用的数据库的名称。
DataSource	获取数据源的服务器名或文件名。
Provider	获取在连接字符串的“Provider=”子句中指定的OLEDB提供程序的名称。
State	获取连接的当前状态。其取值及其说明如表15.7所示。
OleDbConnection类的方法	**说　　明**
Open	使用 ConnectionString 所指定的属性设置打开数据库连接。
Close	关闭与数据库的连接。这是关闭任何打开连接的首选方法。
CreateCommand	创建并返回一个与 OleDbConnection 关联的OleDbCommand 对象。
ChangeDatabase	为打开的OleDbConnection 更改当前数据库。

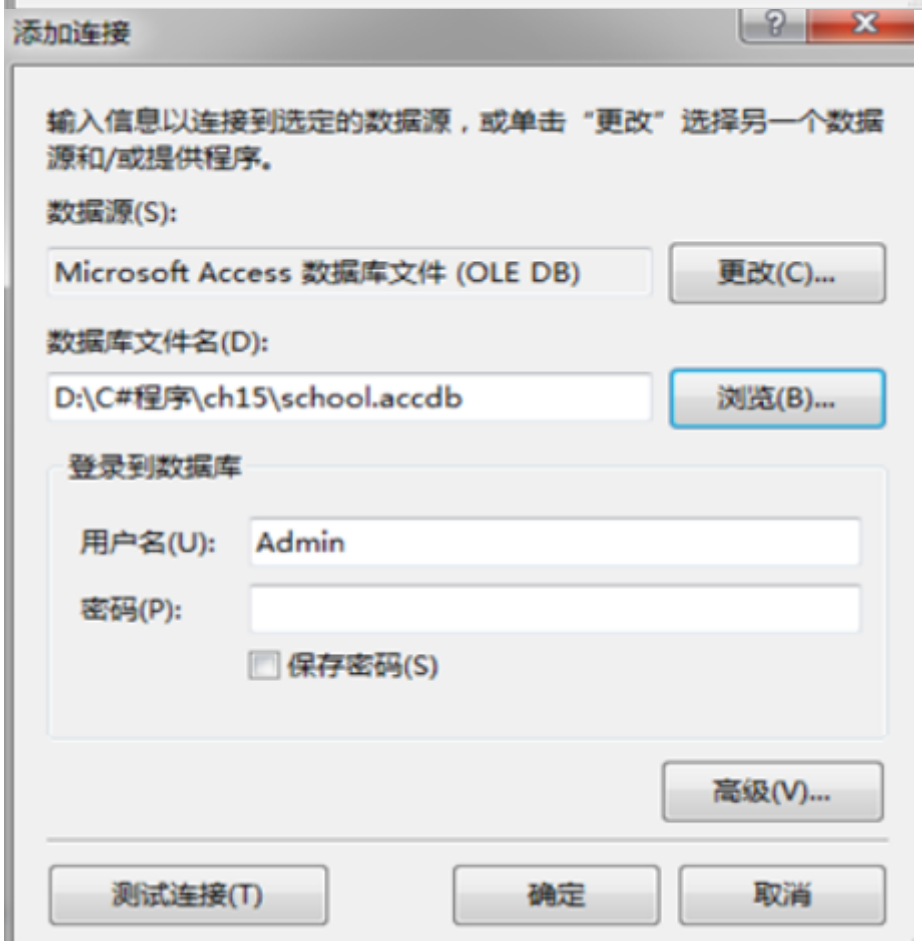
【例】 设计一个窗体，说明直接建立连接字符串的连接过程。

```
Form1，事件过程：
private void button1_Click(object sender, EventArgs e)
{
    string mystr;
    OleDbConnection myconn = new OleDbConnection();
```

```
    mystr = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\C#程序
\ch15\school.accdb";
    myconn.ConnectionString = mystr;
    myconn.Open();
    if (myconn.State == ConnectionState.Open)
        label1.Text = "成功连接到Access数据库";
    else
        label1.Text = "不能连接到Access数据库";
    myconn.Close();
}
```

- 通过属性窗口建立连接： 先要在窗体上放置一个OleDbConnection控件， 在“属性”窗口中单击OleDbConnection控件的ConnectionString属性右侧的 按钮，从弹出的下拉列表中选择“新建连接”选项，打开 “添加连接”对话框，操作如下。





【例】 设计一个窗体，说明通过属性窗口建立连接字符串的连接过程。
Form2：有一个命令按钮button1、一个标签label1和一个OleDbConnection控件oleDbConnection1（已建连接）。
事件过程：

```
private void button1_Click(object sender, EventArgs e)
{
    oleDbConnection1.Open();
    if (oleDbConnection1.State == ConnectionState.Open)
        label1.Text = "成功连接到Access数据库";
    else
        label1.Text = "不能连接到Access数据库";
    oleDbConnection1.Close();
```

```
}
```

- OleDbCommand对象

OleDbCommand类的属性	说　明
CommandText	获取或设置要对数据源执行的 T-SQL 语句或存储过程。
CommandTimeout	获取或设置在终止执行命令的尝试并生成错误之前的等待时间。
CommandType	获取或设置一个值，该值指示如何解释CommandText属性。其取值如表15.10所示。
Connection	数据命令对象所使用的连接对象
Parameters	参数集合（OleDbParameterCollection）
OleDbCommand类的方法	说　明
CreateParameter	创建OleDbParameter对象的新实例。
ExecuteNonQuery	针对Connection 执行SQL语句并返回受影响的行数。
ExecuteReader	将CommandText发送到Connection并生成一个OleDbDataReader。
ExecuteScalar	执行查询，并返回查询所返回的结果集中第一行的第一列。忽略其他列或行。

  - 构造函数

```
OleDbCommand();
OleDbCommand(cmdText) ;    //cmdText参数指定查询的文本
OleDbCommand(cmdText, connection);    //connection参数是一个OleDbConnection，它表示
到Acess数据库的连接

例如：
OleDbConnection myconn = new OleDbConnection();
mystr=@"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\C#程序
\ch15\school.accdb";
myconn.ConnectionString = mystr;
myconn.Open();
OleDbCommand mycmd = new OleDbCommand("SELECT * FROM student",myconn);
```

  - 通过OleDbCommand对象返回单个值

```
例如，如果想获取Student数据库中学生的总人数，则可以使用这个方法执行SQL查询SELECT
Count(*) FROM student。
示例：
string mystr,mysql;
OleDbConnection myconn = new OleDbConnection();
OleDbCommand mycmd = new OleDbCommand();
mystr=@"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\C#程序
\ch15\school.accdb";
myconn.ConnectionString = mystr;
myconn.Open();
mysql = "SELECT AVG(分数) FROM score";
mycmd.CommandText = mysql;
mycmd.Connection = myconn;
textBox1.Text = mycmd.ExecuteScalar().ToString();
myconn.Close();
```

  - 通过OleDbCommand对象执行修改操作

【例】 设计一个窗体，通过OleDbCommand对象将score表中所有分数增5分和减5分。
Form4，设计界面

```
private void Form4_Load(object sender, EventArgs e)
{
    string mystr;
    mystr = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\C#程序
\ch15\school.accdb";
    myconn.ConnectionString = mystr;
    myconn.Open();
}
private void button1_Click(object sender, EventArgs e)
{
    string mysql;
    mysql = "UPDATE score SET 分数=分数+5";
    mycmd.CommandText = mysql;
    mycmd.Connection = myconn;
    mycmd.ExecuteNonQuery();
}
private void button2_Click(object sender, EventArgs e)
{
    string mysql;
    mysql = "UPDATE score SET 分数=分数-5";
    mycmd.CommandText = mysql;
    mycmd.Connection = myconn;
    mycmd.ExecuteNonQuery();
}
```

效果：

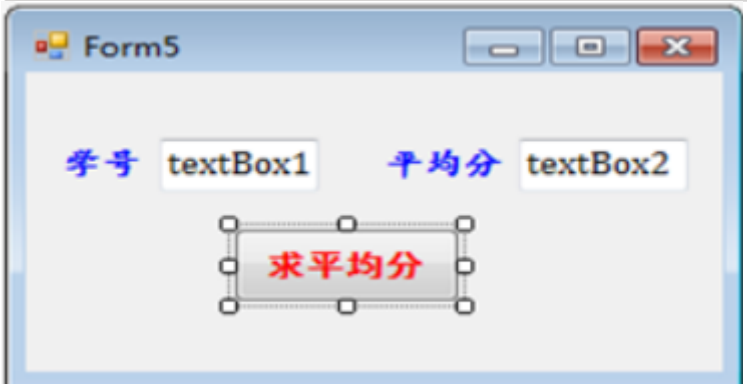
- 在OleDbCommand对象的命令中指定参数

【例】 设计一个窗体，通过OleDbCommand对象求出指定学号学生的平均分。
Form5，设计界面

事件过程：

```
private void button1_Click(object sender, EventArgs e)
{
    string mystr, mysql;
    OleDbConnection myconn = new OleDbConnection();
    OleDbCommand mycmd = new OleDbCommand();
    mystr = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\C#程序
\ch15\school.accdb";
    myconn.ConnectionString = mystr;
    myconn.Open();
    mysql = "SELECT AVG(分数) FROM score WHERE 学号=@no";
    mycmd.CommandText = mysql;
    mycmd.Connection = myconn;
    mycmd.Parameters.Add("@no", OleDbType.VarChar, 5).Value =
textBox1.Text;    //设置参数值
    textBox2.Text = mycmd.ExecuteScalar().ToString();
    myconn.Close();
```

```
}
```



- DataReader对象：当执行返回结果集的命令时，需要一个方法从结果集中提取数据。处理结果集的方法有两个：

    1. 使用DataReader对象（数据阅读器）；
    2. 同时使用DataAdapter（数据适配器），和ADO.NET DataSet。
    3. 使用DataReader对象可以从数据库中得到只读的、只能向前的数据流。使用DataReader对象还可以提高应用程序的性能，减少系统开销，因为同一时间只有一条行记录在内存中

3.1 DataReader类的属性和方法

DataReader类的属性	说　　明
FieldCount	获取当前行中的列数
IsClosed	获取一个布尔值，指出DataReader对象是否关闭
RecordsAffected	获取执行SQL语句时修改的行数
DataReader类的方法	说明
Read	将DataReader对象前进到下一行并读取，返回布尔值指示是否有多行。
Close	关闭DataReader对象。
IsDBNull	返回布尔值，表示列是否包含NULL值。
NextResult	将DataReader对象移到下一个结果集，返回布尔值指示该结果集是否有多行。
GetBoolean	返回指定列的值，类型为布尔值。
GetString	返回指定列的值，类型为字符串。
GetByte	返回指定列的值，类型为字节。
GetInt32	返回指定列的值，类型为整型值。
GetDouble	返回指定列的值，类型为双精度值。
GetDataTime	返回指定列的值，类型为日期时间值。
GetOrdinal	返回指定列的序号或数字位置（首为0）。
GetBoolean	返回指定列的值，类型为对象。

    - DataReader的创建