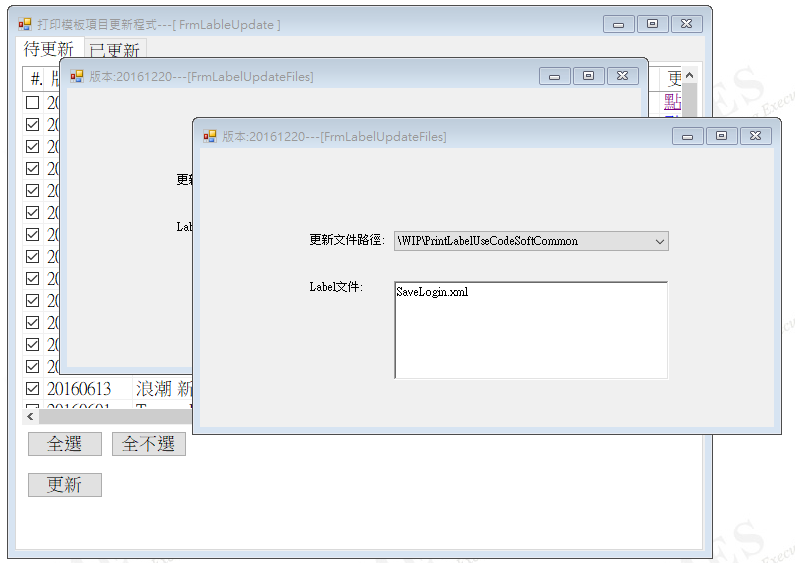
今天在工作过程中遇到了一个问题：

我对一个控件的事件添加一个响应方法。



因为我的datagridview中的数据cell是动态生成的，所以如果在最开始的designer.cs文件中声明会发生报错（即最开始是不存在cell的所以并不能添加cellclick事件）。因此只能在datagridview的数据填充完毕之后再给事件添加方法。

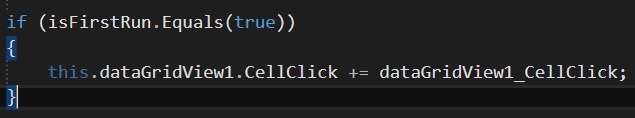
但是因为这个方法（BindLabelUpdate()）会多次执行,所以就会针对这个事件进行多次重复的添加，即最后触发时也会多次执行。如下图所示：



正常情况下，我点击链接按钮只会弹出一个新的窗口，但是这里重复弹出了，即该响应方法执行了两次。

所以应该在添加该事件的响应方法之前添加一个判断，只有当该事件没有添加这个方法时才会去添加。

但是应该怎么实现呢？

如果使用简单的方法的话只需要添加一个状态变量isFirstRun即可.只有在第一次执行的时候才会完成方法的添加。

下面来说另一个方法，说这个方法还要先了解一个概念：反射。

什么是反射：

两个现实中的例子：

B超：大家体检的时候B超可以透过肚皮探测到你内脏的生理情况。B超就是B型超声波，它可以透过肚皮通过向你体内发射B型超声波，当超声波遇到内脏壁的时候就会产生一定的回音反射，然后把回音进行处理就可以得到内脏的情况了。

上面这个例子的特点就是从一个对象的外部去了解对象内部的构造，而且都是利用了波的反射功能。在.NET中的反射也可以实现从对象的外部来了解对象（或程序集）内部结构的功能。哪怕你不知道这个对象是个什么东西，另外.NET中的反射还可以动态创建出对象并执行它其中的方法。

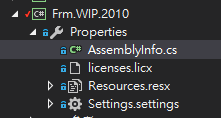
反射是.NET的重要机制，通过反射，可以在运行时获得程序或程序集中每一个类型（包括类、结构、委托、接口和枚举等）的成员和成员信息。有了反射，即可对每个类型了如指掌。另外我还可以直接创建对象，即使这个对象的类型在编译时还不知道。

反射的用途：

1. 使用Assembly定义和加载程序集（专案即dll），加载在程序集清单中列出模块，以及从此程序集中查找类型并创建该类型的实例。

System.Reflection.Assembly类

Assembly类可以获取程序集的信息，也可以动态的加载程序集，以及在程序集中查找类型信息，并创建该类型的实例。使用Assembly类可以降低程序集之间的耦合，有利于软件结构的合理化。



如图中每个专案里的AssemblyInfo文件就是实现这个功能的。

通过程序集名称返回Assembly对象

Assembly ass = Assembly.Load(“ClassLibrary831”);

通过DLL文件名称返回assembly对象

Assembly ass = Assembly.LoadFrom(“ClassLibrary831.dll”);

通过Assembly获取程序集种类

Type t = ass.GetType(“ClassLibrary831.NewClass”); // 参数必须是类的全名

通过Assembly获取程序集中所有的类

Type[] t = ass.GetTypes;

通过程序集的名称反射

Assembly ass = Assembly.Load(“ClassLibrary831”);

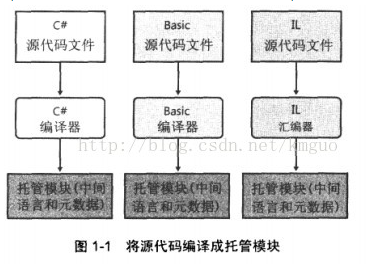
Type t = ass.GetType(“ClassLibrary831.NewClass”);

1. 使用Module了解包含模块的程序集以及模块中的类等，还可以获取在模块上定义的所有全局方法或其他特定的非全局方法。

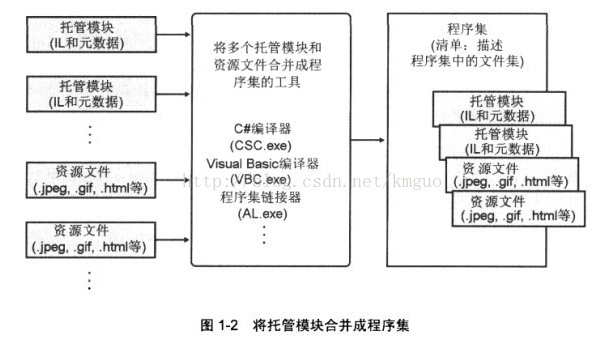
**.NET中托管模块和程序集的关系：**

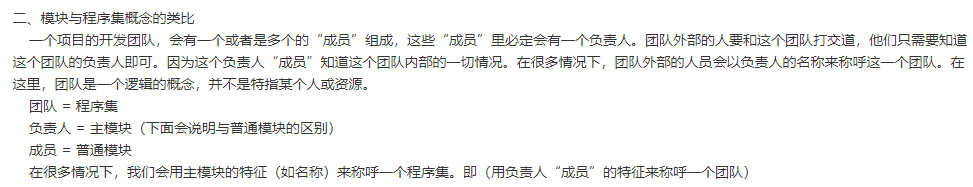
托管模块：

一个标准的32的可移植执行体（PE32）文件或64位（PE32+）文件。由用支持CLR的任何一种语言创建的源代码文件，再经过相应的编译器检查语法和分析源代码，编译生成。

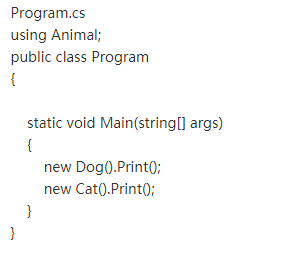


程序集：由一个或多个模块/资源文件的逻辑性组合。程序集是重用、安全性以及版本控制的最小单元。可以将.exe和.dll文件认为是一个程序集；其中exe是一个自己执行的程序集，dll将被其他程序集加载后运行。











另外还有命名空间和程序集的关系：



1. 使用ConstructorInfo了解构造函数的名称，参数，访问修饰符（如public或private）和实现详细信息（如abstract或virtual）
2. 使用MethodInfo了解方法的名称、返回类型、参数、访问修饰符和实现详细信息。
3. 使用FieldInfo了解字段的名称、访问修饰符和实现详细信息如（static）等
4. 使用EventInfo了解事件的名称、事件处理程序数据类型、自定义属性、声明类型和反射类型等，添加或移除事件处理程序。
5. 使用propertyInfo了解属性的名称、数据类型、声明类型、反射类型和只读或可写状态等，获取或设置属性值。
6. 使用ParameterInfo了解参数的名称、数据类型是输入参数还是输出参数，以及参数在方法签名中的位置等。

反射用到的命名空间：System.Reflection

反射用到的主要类：

System.Type类：

System.Type类对于反射起着核心的作用，但它是一个抽象的基类，Type有与每种数据类型对应的派生类，我们使用这个派生类的对象的方法、字段、属性来查找有关该类型的所有信息。

获取给定类型的Type引用有3中常用方式：

* 使用C# typeof运算符。

Type t = typeof(string);

* 使用对象GetType() 方法

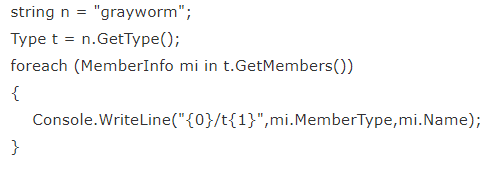
String s = “123”

Type t = s.GetType();

* 还可以调用Type类的静态方法GetType()

Type t = Type.GetType(“System.String”);

上面三类代码都是获取string类型的Type，在取出string类型的Type引用t后，我们就可以通过t来探测string类型的结构了。



获取string类型的成员信息。

Type类的属性：

Name 数据类型名

FullName 数据类型的完全限定名（包括命名空间名）

IsAbstract 指示该类型是否是抽象类型

IsArray 指示该类型是否是数组

IsClass

IsEnum

isInterface

Type类的方法：

GetConstructor(), GetConstructors():返回ConstructorInfo类型，用于取得该类的构造函数的信息

GetEvent(), GetEvents():返回EventInfo类型，用于取得该类的时间的信息

GetField(), GetFields(): 返回FieldInfo类型，用于取得该类的字段（成员变量）的信息

GetInterface(), GetInterfaces():返回InterfaceInfo类型，用于取得该类实现的接口的信息

GetMember(), GetMembers()

GetMethod(), GetMethods()

GetProperty(), GetProperties()

可以调用这些成员，其方式是调用Type的InvokeMember()方法，或者调用MethodInfo，PropertyInfo和其他类的Invoke()方法。

查看类中的构造方法：

New Classw nc = new NewClassw();

Type t = nc.GetType();

ConstructorInfo[] ci = t.GetConstructors(); // 获取类的所有构造函数

foreach(ConstructorInfo c in ci)

{

ParameterInfo[] ps = c.GetParameters(); // 去获取每个构造函数的参数

foreach(ParameterInfo pi in ps)

{

Console.Write(pi.ParameterType.ToString() + pi.Name);

}

}

用构造函数动态生成对象：

Type t = typeof(NewClassw);

Type[] pt = new Type[2];

Pt[0] = typeof(string);

Pt[1] = typeof(string);

//根据参数类型获取构造函数

ConstructorInfo ci = t.GetConstructor(pt);

//构造Object数组，作为构造函数的输入参数

Object[] obj = new object[2]{“grayworm”, “hi.baidu.com/grayworm”};

// 调用构造函数 生成对象

**Object o = ci.invoke(obj);**

// 调用生成的对象的方法测试是否对象生成成功

//((NewClassw)o).show();

用Activator生成对象

Type t = typeof(NewClassw);

// 构造函数的参数

Object[] obj = new object[2] {“123”, “456”};

// 用Activator的CreateInnstance静态方法，生成新对象

Object o = Activator.CreateInstance(t, obj[0], obj[1]);

// ((NewClassw)o),show();

查看类中的属性：

NewClassw nc = new NewClassw();

Type t = nc.GetType();

PropertyInfo[] pis = t.GetProperties();

foreach(PropertyInfo pi in pis)

{

Console.WriteLine(pi.Name);

}

查看类中的public方法

NewClassw nc = new NewClassw();

Type t = nc.GetType();

MethodInfo[] mis = t.GetMethods();

Foreach(MethodInfo mi in mis)

{

Console.WriteLine(mi.ReturnType + mi.Name);

}

**查看类中的public字段**        NewClassw nc = new NewClassw();  
        Type t = nc.GetType();  
        FieldInfo[] fis = t.GetFields();  
        foreach (FieldInfo fi in fis)  
        {  
            Console.WriteLine(fi.Name);  
        }

**用反射生成对象，并调用属性、方法和字段进行操作**        NewClassw nc = new NewClassw();  
        Type t = nc.GetType();  
        object obj = Activator.CreateInstance(t);  
        //取得ID字段   
        FieldInfo fi = t.GetField("ID");  
        //给ID字段赋值   
        fi.SetValue(obj, "k001");  
        //取得MyName属性   
        PropertyInfo pi1 = t.GetProperty("MyName");  
        //给MyName属性赋值   
        pi1.SetValue(obj, "grayworm", null);  
        PropertyInfo pi2 = t.GetProperty("MyInfo");  
        pi2.SetValue(obj, "hi.baidu.com/grayworm", null);  
        //取得show方法   
        MethodInfo mi = t.GetMethod("show");  
        //调用show方法   
        mi.Invoke(obj, null);