**[详解C#中的反射](file:///\\\\192.168.10.50\\public\\share\\Unity3D%201604\\2016-5-03\\详解C" \l "中的反射 - 记事本 - 博客频道 - CSDN.NET_files/详解C#中的反射 - 记事本 - 博客频道 - CSDN.NET.html)**

|  |
| --- |
| **两个现实中的例子：** 1、B超：大家体检的时候大概都做过B超吧，B超可以透过肚皮探测到你内脏的生理情况。这是如何做到的呢？B超是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”进行处理就可以显示出内脏的情况了（我不是医生也不是声学专家，不知说得是否准确^\_^）。 2、地球内部结构：地球的内部结构大体可以分为三层：地壳、地幔和地核。地壳是固体，地核是液体，地幔则是半液半固的结构（中学地理的内容，大家还记得吧？）。如何在地球表面不用深入地球内部就知道其内部的构造呢？对，向地球发射“地震波”，“地震波”分两种一种是“横波”，另一种是“纵波”。“横波”只能穿透固体，而“纵波”既可穿透固体又可以穿透液体。通过在地面对纵波和横波的反回情况，我们就可以大体断定地球内部的构造了。  大家注意到这两个例子的共同特点，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（或程序集）是个什么东西，另外.NET中的反射还可以运态创建出对象并执行它其中的方法。  反射是.NET中的重要机制，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）的成员和成员的信息。有了反射，即可对每一个类型了如指掌。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，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。      （2）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。      （3）使用ConstructorInfo了解构造函数的名称、参数、访问修饰符（如pulic 或private）和实现详细信息（如abstract或virtual）等。      （4）使用MethodInfo了解方法的名称、返回类型、参数、访问修饰符（如pulic 或private）和实现详细信息（如abstract或virtual）等。     （5）使用FiedInfo了解字段的名称、访问修饰符（如public或private）和实现详细信息（如static）等，并获取或设置字段值。     （6）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。      （7）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。      （8）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     System.Type     System.Reflection.Assembly      **反射用到的主要类：**     System.Type 类－－通过这个类可以访问任何给定数据类型的信息。     System.Reflection.Assembly类－－它可以用于访问给定程序集的信息，或者把这个程序集加载到程序中。      **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种常用方式：     **●使用 C# typeof 运算符。         Type t = typeof(string);     ●使用对象GetType()方法。         string s = "grayworm";         Type t = s.GetType();      ●还可以调用Type类的静态方法GetType()。         Type t = Type.GetType("System.String");**             上面这三类代码都是获取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。              string n = "grayworm";             Type t = n.GetType();             foreach (MemberInfo mi in t.GetMembers())             {                 Console.WriteLine("{0}/t{1}",mi.MemberType,mi.Name);             }          **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)         Namespace 定义数据类型的命名空间名         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   指示该类型是否是类         IsEnum   指示该类型是否是枚举         IsInterface    指示该类型是否是接口         IsPublic 指示该类型是否是公有的         IsSealed 指示该类型是否是密封类         IsValueType 指示该类型是否是值类型     **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，用于取得该类的事件的信息         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()：返回MemberInfo类型，用于取得该类的所有成员的信息         GetMethod(), GetMethods()：返回MethodInfo类型，用于取得该类的方法的信息         GetProperty(), GetProperties()：返回PropertyInfo类型，用于取得该类的属性的信息     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：**         NewClassw 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) //遍历每一个构造函数         {             **ParameterInfo[] ps = c.GetParameters();**    //取出每个构造函数的所有参数             foreach (**ParameterInfo pi in ps**)   //遍历并打印所该构造函数的所有参数             {                 Console.Write(pi.ParameterType.ToString()+" "+pi.Name+",");             }             Console.WriteLine();         }          **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] { "grayworm", "hi.baidu.com/grayworm" };            //用Activator的CreateInstance静态方法，生成新对象          **object o = Activator.CreateInstance(t,"grayworm","hi.baidu.com/grayworm");**         //((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);         } (<http://hi.baidu.com/grayworm>)             **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);          **System.Reflection.Assembly类**       Assembly类可以获得程序集的信息，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。          **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获取程序集中类          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");**     object o = Activator.CreateInstance(t, "grayworm", "<http://hi.baidu.com/grayworm>");     MethodInfo mi = t.GetMethod("show");     mi.Invoke(o, null);  **//通过DLL文件全名反射其中的所有类型**     **Assembly assembly = Assembly.LoadFrom("xxx.dll的路径");     Type[] aa = a.GetTypes();**      foreach(Type t in aa)     {         if(t.FullName == "a.b.c")         {             object o = Activator.CreateInstance(t);         }     } |