用户操作

[留言] [发消息] [加为好友]

订阅我的博客

0 位读者 POWERED BY FEEDSKY

订阅 订阅到鲜果 订阅到Google 订阅到抓虾

ghfsusan的公告

文章分类

asp.net技术交流

Enterprase library

随意小笔

asp.net网络文档集合

http://hi.baidu.com/honfei

UML概述(1)

观察者模式（Observer Pattern）

存档

2009年01月(4)

2008年12月(1)

2008年11月(5)

2008年10月(2)

原 .net反射简介 - 概述 收藏

.net反射简介 - 概述

反射就是动态发现类型信息的能力。它帮助程序设计人员在程序运行时利用一些信息去动态地使用类型，这些信息在设计时是未知的，这种能力类型于后期绑定。反射还支持的更高级的行为，能在运行时动态创建新类型，并且对这些新类型的操作进行调用。

编辑本段 回目录

.net反射简介 - 一些在反射中经常使用的类

Assembly类

Assembly类是可重用、无版本冲突并且可自我描述的公共语言运行库应用程序构造块。可以使用Assembly.Load和Assembly.LoadFrom方法动态地加载程序集。

Type类

反射的中心是System.Type类。System.Type类是一个抽象类，代表公用类型系统中的一种类型。这个类使您能够查询类型名、类型中包含的模块和名称空间、以及该类型是一个数值类型还是一个引用类型。
System.Type类使您能够查询几乎所有与类型相关的属性，包括类型访问限定符、类型是否、类型的COM属性等等。

Activator类

Activator类支持动态创建.NET程序集和COM对象。可以通过CreateComInstanceFrom、CreateInstance、CreateInstanceFrom、GetObject四个静态方法加载COM对象或者程序集，并能创建指定类型的实例。

Binder类

Binder类是一个用于执行类型转换的绑定器，Type对象的InvokeMember方法接受Binder对象，这个对象描述了如何将传递给InvokeMember的参数转换成方法实际需要的类型。

Binder类是一个抽象类，要创建绑定器，需要重写方法BindToMethod、BindToField、SelectMehtod、SelectProperty和ChangeType。

DefaultMemberAttribute类

DefaultMemberAttribute类用于类型并带有一个指明默认成员名称的字符串参数。能够通过InvokeMember调用默认成员，而不需要传递调用成员的名称。当需要绑定器但不需

要特别的绑定行为时就可以使用它。

编辑本段 回目录

.net反射简介 - 其它

还有一些对元素类型信息描述的类，ConstrutorInfo（构造函数）、MethodInfo（方法）、FieldInfo（字段）、PropertyInfo（属性）、EventInfo（事件）、MemberInfo（成员）、ParameterInfo（参数）。如果查询得到了具有任何类型信息的实例，就可以获得该类型中任意元素的类型信息，当然出于安全原因，不保证会得到程序集中的任何信息。

编辑本段 回目录

.net反射简介 - 示例

类定义：

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ReflectionSample
{
    /**//**//**////
    /// 说明：一个简单的类
    /// 作者：文野
    /// 联系：stwyhm.cnblog.com
    ///
    public class ClassSample
    {
        // 默认构造
        public ClassSample()
        {
            this.name = "您调用了默认构造创建了一个类实例。";
        }

        // 带参构造
        public ClassSample(string name)
        {
            this.name = name;
        }

        // 字段
```

```
28.         public string name;
29.
30.         public string Field;
31.
32.         // 属性
33.         private string property;
34.         public string Property
35.         {
36.             set { this.property = value; }
37.             get { return property; }
38.         }
39.
40.         // public方法
41.         public string PublicClassMethod()
42.         {
43.             return string.Format("您反射了一个Public方法");
44.         }
45.
46.         // private方法
47.         private string PrivateClassMethod()
48.         {
49.             return string.Format("您反射了一个Private方法");
50.         }
51.
52.         // static方法
53.         public static string StaticMethod()
54.         {
55.             return "您反射了一个Static方法";
56.         }
57.
58.         // 带参方法
59.         public string ParameterMethod(string para)
60.         {
61.             return para;
62.         }
63.
64.         public event EventHandler eventHandler;
65.
66.         public void DoEvent()
```

```
67.         {
68.             eventHandler(null,EventArgs.Empty);
69.         }
70.     }
71. }
72. 反射示例
73. 
74. using System;
75. using System.Data;
76. using System.Configuration;
77. using System.Web;
78. using System.Web.Security;
79. using System.Web.UI;
80. using System.Web.UI.WebControls;
81. using System.Web.UI.WebControls.WebParts;
82. using System.Web.UI.HtmlControls;
83. 
84. using System.Reflection;
85. using ReflectionSample;
86. 
87. /**//**//**////
88. /// 说明：一个简单的使用反射示例
89. /// 作者：文野
90. /// 联系：stwyhm.cnblog.com
91. ///
92. public partial class _Default : System.Web.UI.Page
93. {
94.     protected void Page_Load(object sender, EventArgs e)
95.     {
96.         string path = Server.MapPath(Request.Path);
97.         string filePath = path.Substring(0, path.LastIndexOf(''\\'')) + @"\bi
    n\ReflectionSample.dll";
98.         // 获取程序集
99.         Assembly classSampleAssembly = Assembly.LoadFrom(filePath);
100. 
101.         // 从程序集中获取指定对象类型
102.         Type classSampleType = classSampleAssembly.GetType("ReflectionS
    ample.ClassSample");
103. 
```

```
104.        使用Activator创建一个实例使用Activator创建一个实例#region 使用Activat
            or创建一个实例
105.
106.        // 通过对象类型创建对象实例
107.        ClassSample s1 = Activator.CreateInstance(classSampleType) as Cla
            ssSample;
108.
109.        Response.Write(s1.name + "（使用Activator创建一个实例）");
110.
111.        #endregion
112.
113.        动态调用构造函数动态调用构造函数#region 动态调用构造函数
114.
115.        // 调用无参构造
116.        ConstructorInfo studentConstructor1 = classSampleType.GetConstru
            ctor(new Type[] { });
117.        ClassSample s2 = studentConstructor1.Invoke(new object[] { }) as
            ClassSample;
118.        Response.Write(s2.name + "");
119.
120.        // 调用有参构造
121.        ConstructorInfo studentConstructor2 = classSampleType.GetConstru
            ctor(new Type[] { typeof(string) });
122.        ClassSample s3 = studentConstructor2.Invoke(new object[] { "您调
            用了有参构造创建了一个类实例。" }) as ClassSample;
123.        Response.Write(s3.name + "");
124.
125.        #endregion
126.
127.        反射方法反射方法#region 反射方法
128.
129.        // 调用非静态方法
130.        string returnValue1 = classSampleType.InvokeMember("PublicClass
            Method", BindingFlags.InvokeMethod | BindingFlags.Public | BindingFlags.
            Instance, null, s1, new object[] { }) as string;
131.        Response.Write(returnValue1 + "");
132.
133.        // 调用静态方法
134.        string returnValue2 = classSampleType.InvokeMember("StaticMetho
            d", BindingFlags.InvokeMethod | BindingFlags.Public | BindingFlags.Static,
```

```
null, s1, new object[] { }) as string;
        Response.Write(returnValue2 + "");

        // 调用私有方法
        string returnValue3 = classSampleType.InvokeMember("PrivateClassMethod", BindingFlags.InvokeMethod | BindingFlags.NonPublic | BindingFlags.Instance, null, s1, new object[] { }) as string;
        Response.Write(returnValue3 + "");

        #endregion

        反射参数反射参数#region 反射参数

        MethodInfo parameterMethod = classSampleType.GetMethod("ParameterMethod");
        ParameterInfo[] paras = parameterMethod.GetParameters();
        for (int i = 0; i ", new object[] { paras[i].Name, paras[i].ParameterType.ToString(), paras[i].IsOptional.ToString(), paras[i].Position.ToString(), paras[i].DefaultValue.ToString() }));

        #endregion

        反射属性反射属性#region 反射属性

        classSampleType.InvokeMember("Property", BindingFlags.SetProperty | BindingFlags.Public | BindingFlags.Instance, null, s1, new object[] { "您反射了一个属性" });
        string returnValue4 = classSampleType.InvokeMember("Property", BindingFlags.GetProperty | BindingFlags.Public | BindingFlags.Instance, null, s1, new object[] { }) as string;
        Response.Write(returnValue4 + "");

        #endregion

        反射字段反射字段#region 反射字段

        classSampleType.InvokeMember("Field", BindingFlags.SetField | BindingFlags.Public | BindingFlags.Instance, null, s1, new object[] { "您反射了一个字段" });
        string returnValue5 = classSampleType.InvokeMember("Field", BindingFlags.GetField | BindingFlags.Public | BindingFlags.Instance, null, s1, n
```

```
      ew object[] { }) as string;
163.        Response.Write(returnValue5 + "");
164.
165.        #endregion
166.    }
167. }
168.
169.
```


旧一篇:1 反射技术与设计模式 | 新一篇:利用.net反射动态调用指定程序集的中的方法