**控制反转(IoC)容器：Unity，构造函数注入**

[柚子](http://www.ityouzi.com/archives/author/柚子)2015-08-26**3116**阅读

[Unity](http://www.ityouzi.com/archives/tag/Unity)[Ioc](http://www.ityouzi.com/archives/tag/Ioc)[依赖注入](http://www.ityouzi.com/archives/tag/依赖注入)[控制反转](http://www.ityouzi.com/archives/tag/控制反转)[构造函数](http://www.ityouzi.com/archives/tag/构造函数)A+

Unity的注入方式分为3种：构造函数注入、属性注入、方法注入。这里我们主要讲解下构造函数注入。

**基础准备工作**

1、安装Unity，<http://www.ityouzi.com/archives/ioc-unity-install.html>

2、创建几个简单的类。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /// <summary>  /// 班级接口  /// </summary>  public  interface IClass  {      string ClassName { get; set; }        string ShowInfo();  } |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /// <summary>  /// 计科班  /// </summary>  public class CbClass : IClass  {     public string ClassName { get; set; }       public string ShowInfo()     {        return string.Format("计科班：{0}", ClassName);     }  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /// <summary>  /// 电商班  /// </summary>  public class EcClass : IClass  {       public string ClassName { get; set; }         public string ShowInfo()       {           return string.Format("电商班：{0}", ClassName);       }  } |
| 1  2  3  4  5  6  7  8  9  10 | /// <summary>  /// 学生接口  /// </summary>  public interface IStudent  {      string Name { get; set; }        //就读班级      string ShowInfo();  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /// <summary>  /// 学生  /// </summary>  public class QlinStudent : IStudent  {      public string Name { get; set; }      private IClass ToClass { get; set; }        public QlinStudent(IClass \_class)      {          ToClass = \_class;      }        public string ShowInfo()      {          return string.Format("{0}就读班级：{1}", Name, ToClass.ClassName);      }  } |

3、设置配置参数。

Unity是通过配置参数来描述类之间的关系，所以需要设置配置文件。

在项目根目录创建一个Unity.config文件，并粘贴以下内容

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <configuration>    <configSections>      <section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection,Microsoft.Practices.Unity.Configuration"/>    </configSections>      <unity xmlns="<http://schemas.microsoft.com/practices/2010/unity>">        <!--引用命名空间-->      <namespace name="WEB" />      <!--引用程序集-->      <assembly name="WEB.App\_Code" />      <!--容器-->      <container name="FirstClass">        <!--映射关系-->        <register type="IClass"  mapTo="CbClass"></register>        <register type="IClass" name="ec" mapTo="EcClass"></register>        <register type="IStudent"  mapTo="QlinStudent"> </register>      </container>    </unity>      <system.web>      <compilation debug="true" targetFramework="4.0" />    </system.web>    </configuration> |

对于上面的参数配置意思我们这里不详细介绍，具体介绍待后面说明。如果是web也可以直接写在Web.config中，为了方便管理我这里单独成一个文件。

为了方便管理，我们将Unity的相关操作封装成一个助手类。

添加引用：

|  |  |  |
| --- | --- | --- |
| 1  2  3 | | using Microsoft.Practices.Unity;  using System.Configuration;  using Microsoft.Practices.Unity.Configuration; |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | /// <summary>      /// Unity助手类      /// </summary>      public class UnityHelper      {          /// <summary>          /// 容器对象          /// </summary>          private static IUnityContainer  UnityContainer;            /// <summary>          /// 获取容器          /// </summary>          /// <returns></returns>          public static IUnityContainer GetIUnityContainer()          {              if (UnityHelper.UnityContainer != null) return UnityHelper.UnityContainer;                //通过Unity.config文件来设置容器              IUnityContainer container = new UnityContainer();                string configFilePath = HttpContext.Current.Server.MapPath("~/Unity.config");              var fileMap = new ExeConfigurationFileMap { ExeConfigFilename = configFilePath };                Configuration configuration = ConfigurationManager.OpenMappedExeConfiguration(fileMap, ConfigurationUserLevel.None);              UnityConfigurationSection section = (UnityConfigurationSection)configuration.GetSection("unity");                container.LoadConfiguration(section, "FirstClass");              UnityHelper.UnityContainer = container;                return UnityHelper.UnityContainer;          }          } | |

**项目运用**

通过上面的准备工作，我们现在开始代码正式使用。

**1、通过默认的构造函数来实现注入**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /// <summary>          /// 使用默认构造函数来注入          /// </summary>          private void Test()          {              //获得容器对象              IUnityContainer container = UnityHelper.GetIUnityContainer();                //默认注册（无命名）,如果后面还有默认注册会覆盖前面的              container.RegisterType<IClass, CbClass>();              container.RegisterType<IStudent, QlinStudent>();                //解析默认对象              IStudent splitClass = container.Resolve<IStudent>();             string result=  splitClass.ShowInfo();               this.lbl.Text = result;              //返回结果：就读班级：          } |

上面我们不管是在注册还是解析对象的时候都没有传递相关参数，那么解析对象的时候就通过默认的构造函数来完成的。就好像有我们不传递参数来new对象一样。

默认方式跟new一个对象，它会根据你提供的材料，选择一个构造函数，即要有构造器要能访问权限，用Public修饰，构造函数的参数也要提供，即IClass也要能解析，不然就报错了。

如果有显示构造函数就会使用显示的构造函数，没有的话就会使用默认的构造函数（空构造函数）

返回结果：就读班级：。我们没有传递Name和IClass 2个参数，所以结果上没有显示相对应的值。

**2、多构造函数使用"[InjectionConstructor]"特性来显示指定**

如果类具有多个构造函数，那么我们可以通过“InjectionConstructor”特效来显示指定使用那个构造函数。

我们给QlinStudent类添加一个构造函数，现在这个类有2个构造函数，

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public QlinStudent(IClass \_class,string name)  {        ToClass = \_class;        Name = name;  }    [InjectionConstructor]  public QlinStudent(IClass \_class)  {      ToClass = \_class;  } |

运行结果会调用下面的构造函数，因为我们使用了“InjectionConstructor”特效来显示指定了构造函数。

**3、指定参数依赖的注册名称**

构造函数中IClass参数，如果IUnityContainer注册了多个，默认是使用无名称的那个注册，也可以通过Dependency依赖哪个名称来指定哪个来注册，代码，指

定ec名称如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | [InjectionConstructor]  public QlinStudent([Dependency("ec")]IClass \_class)  {      ToClass = \_class;  } |

下面注册一个名称为ec的映射，如果没有名称ec的映射将报错

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public static void Test()  {      IUnityContainer container =  UnityHelper.GetIUnityContainer();      //默认注册（无命名）,如果后面还有默认注册会覆盖前面的      container.RegisterType<IClass, CbClass>();      //命名注册      container.RegisterType<IClass, EcClass>("ec");      container.RegisterType<IStudent, QlinStudent>();      //解析默认对象      IStudent splitClass = container.Resolve<IStudent>();      splitClass.ShowInfo();  } |

配置文件方式，代码不变，配置中添加一个 name属性就行，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <container name="FirstClass">      <!--映射关系-->      <register type="IClass" mapTo="CbClass"></register>          <register type="IClass" name="ec" mapTo="EcClass"></register>          <register type="IStudent" mapTo="QlinStudent">      </register>  </container> |

**4、指定构造函数的参数**

构造器中的参数也可以依赖一个指定的类型值，如下代码依赖于EcClass类型，可以让构造函数中可以传入一个具体的类型，这也是构造函数传参数，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public static void Test()  {      IUnityContainer container =  UnityHelper.GetIUnityContainer();      //默认注册（无命名）,如果后面还有默认注册会覆盖前面的      container.RegisterType<IClass, CbClass>();      //命名注册      container.RegisterType<IClass, EcClass>("ec");      container.RegisterType<IStudent, QlinStudent>(new InjectionConstructor(new CbClass()));      IStudent splitClass = container.Resolve<IStudent>();      splitClass.ShowInfo();  } |

或者注册一个实例对象，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public static void Test()  {      IUnityContainer container =  UnityHelper.GetIUnityContainer();      IClass cbClass = new CbClass { ClassName="计科051班" };      //实例注册命名实例      container.RegisterInstance<IClass>("ec", cbClass);      container.RegisterType<IStudent, QlinStudent>();      IStudent splitClass = container.Resolve<IStudent>();      splitClass.ShowInfo();  } |

配置文件也可以指定类型依赖，如下，指定EcClass：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <register type="IStudent" mapTo="QlinStudent">      <constructor>          <param name="\_class" type="IClass">              <dependency type="EcClass"/>          </param>      </constructor>  </register> |

上面已经介绍了传参数，是用InjectionConstructor类型，现在构造函数，多一个参数，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [InjectionConstructor]  public QlinStudent([Dependency("ec")]IClass \_class, string name)  {      ToClass = \_class;      Name = name;  } |

多了一个name参数，那必须为容器IUnityContainer提供这个参数，没有这个原材料，它无法构造，就会报错，如下代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public static void Test()  {      IUnityContainer container =  UnityHelper.GetIUnityContainer();      container.RegisterType<IStudent, QlinStudent>(new InjectionConstructor(new CbClass() { ClassName = "计科051" }, "Qlin"      IStudent splitClass = container.Resolve<IStudent>();      splitClass.ShowInfo();  } |

注入参数后，也可以下次解析的时候，通过ParameterOverrides类来覆盖原来的参数，改变参数值，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public static void Test()  {      IUnityContainer container = UnityHelper.GetIUnityContainer();      container.RegisterType<IStudent, QlinStudent>(new InjectionConstructor(new CbClass() { ClassName = "计科051" }, "Qlin"      IStudent student = container.Resolve<IStudent>();      student.ShowInfo();        //覆盖参数解析      IStudent student1 = container.Resolve<IStudent>(new ParameterOverrides(){          {"\_class",new EcClass(){ ClassName="电商051"}},          {"name","linq"}      });      student1.ShowInfo();  } |