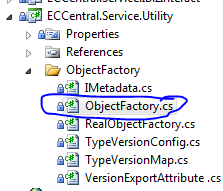
       在我们的Service端，做了一个ObjectFactory<T>工具，命名空间为ECCentral.Service.Utility，用来实现类型注入（暂时还只支持无参构造的类型，如果后期有需要再改造支持有参数的构造），最大的特点是能够节省大量的配置。

       传统的方式，我们要使用一个类型实例，都是用new关键字来调用某个具体类型的构造函数。现在我们不会直接使用具体的实现类型，而是都用ObjectFactory<T>来创建一个接口或基类的具体实现（之前提过的我们现在写的类的public都用virtual方法，都可以视为以后扩展的基类，都通过ObjectFactory<T>来创建）



使用方式如下：

1. **在类上面使用：**

在类型上定义一个VersionExport的attribute：

[VersionExport(typeof(OrderProcessBase))]

public class OrderProcessBase

{

   ......

}

在需要使用OrderProcessBase的地方：

OrderProcessBase obj = ObjectFactory<OrderProcessBase>.Instance;

就可以直接获得OrderProcessBase的对象实例

1. **在接口上使用**

对于接口也是一样方式：

public interface ICaculate

{

      ......

}

[VersionExport(typeof(ICaculate))]

public class MyCaculator : ICaculate

{

}

在需要使用ICaculate的地方：

ICaculate obj = ObjectFactory<ICaculate>.Instance; // 会得到MyCaculator的对象实例

1. **对于一个接口多个实现，或者一个基类多个子类的时候：**

[VersionExport(typeof(ICaculate) , "1.0.0.0")]

public class MyCaculator : ICaculate

[VersionExport(typeof(ICaculate) , "2.0.0.0")]

public class YourCaculator : ICaculate

在需要使用ICaculate的地方：

ICaculate obj = ObjectFactory<ICaculate>.GetInstance("1.0.0.0"); // 会得到MyCaculator的对象实例

ICaculate obj = ObjectFactory<ICaculate>.GetInstance("2.0.0.0"); // 会得到YourCaculator的对象实例

如果此时用ObjectFactory<ICaculate>.Instance 得到的对象实例则是，系统默认设置的一个版本号的对象实例

VersionExport(typeof(ICaculate))] 如前面所示，这种标签并没有明确指定该Export的类型的版本号，则会使用系统默认设置的一个版本号

所以所有VersionExport的类型都是有版本号的，没有人为明确指定则使用系统默认版本号。

ObjectFactory<ICaculate>.Instance 内部的逻辑是会先根据ICaculate类型去检查是否有在配置文件里配置了对应类型的版本号，如果有，则使用配置文件的版本号来获取对应的对象实例

如果配置文件里没有对应类型配置，则看配置文件是否设定了全局默认版本号**【1】**，如果有设定则使用，如果还是没有设定，则使用系统默认的一个版本号（会和VersionExport(typeof(ICaculate))]所使用的默认版本号是一致的）

所以默认在没有任何配置的情况下，VersionExport(typeof(ICaculate))] 和 ObjectFactory<ICaculate>.Instance 是会使用相同的版本号，会自动匹配上，这样可以减少大家敲击键盘的次数

1. **对于一个实现类，需要导出2个或以上的接口或基类的时候：**

增加ExportAttribute即可，为每个需要导出的接口或基类添加一个Export标签，但他们都只能使用同一个版本号，即定义在VersionExport里的版本号。

[Export(typeof(IXXXService))]

[Export(typeof(OrderBase))]

[VersionExport(typeof(ICaculate))]

public class CaculatorAppService : ICaculate, IXXXService, OrderBase

这里将导出的2个接口+1个基类，都是使用的默认版本号

[Export(typeof(IXXXService))]

[Export(typeof(OrderBase))]

[VersionExport(typeof(ICaculate) , "3.0.0.0")]

public class CaculatorAppService : ICaculate, IXXXService, OrderBase

{

。。。。

}

ICaculate obj = ObjectFactory<ICaculate>.GetInstance("3.0.0.0");

IXXXService obj1 = ObjectFactory< IXXXService >.GetInstance("3.0.0.0");

OrderBase obj2 = ObjectFactory<OrderBase >.GetInstance("3.0.0.0");

得到的三个实例对象，但其具体类型都是CaculatorAppService

1. **如何升级**

也正是通过第三点这个版本特性，我们后期可以很方便的为某个类进行升级，而不需要动到以前的代码。

因为在代码中，我们默认使用的ObjectFactory<ICaculate>.Instance方式来获取类型的实例，那么我们只要为接口类型ICaculate配置新的对应版本号，就可以得到一个新的实现类型的对象实例

所以我们会有个配置文件，结构如下：

<?xml version="1.0"?>

<typeVersionMaps globalDefaultVersion="1.0.0.0" assemblyFolder="Bin; Bin\Inventory">

  <map type="TestDemo.ICaculate, TestDemo" version="3.0.0.0" />

 …..

</typeVersionMaps>

在里面为ICaculate定义了使用标记了3.0.0.0的VersionExport标签的类型。我们只要把这个新开发的3.0.0.0的类型的dll放到assemblyFolder所指定的目录里即可（assemblyFolder可指定多个目录，用;或,隔开）。

另外我们看到还有个globalDefaultVersion 的属性，它是用来设定ObjectFactory<>.Instance在无法找到类型对应配置时，所使用的默认版本号，上面**【1】**处所提到的

从这里也可以看出，仅仅在需要升级类型的实现的时候，才需要使用配置，第一次完成系统的时候，应该是一行配置都不需要。

另外ObjectFactory<T>还有NewInstance的方法，和Instance属性/GetInstance方法的区别在于，Instance/GetInstance()获取的是Singlton的对象实例，而NewInstance()每一次调用获取到的都是一个新的对象实例

如果没有特别的需求，建议使用GetInstance

1. **如果程序里需要多态的情况（不是版本的多态，而根据业务设计出的多态）**

举个列子，比如针对OrderType为A的订单的审核，会有一个对应的SO\_A\_Auditor类型，对于OrderType为B的订单的审核SO\_B\_Auditor类型，对于OrderType为C的订单的审核，会有一个对应的SO\_C\_Auditor类型；

而SO\_A\_Auditor、SO\_B\_Auditor、SO\_C\_Auditor都是从IAuditor接口派生出来的，我们会有个AuditorFactory，会根据OrderType来创建具体的某一种Auditor，可能方法如下：

public IAuditor CreateAuditor(OrderType orderType)

{

switch(orderType)

{

case “A”:

return SO\_A\_Auditor的对象实例

case “B”:

return SO\_B\_Auditor的对象实例

default:

return SO\_C\_Auditor的对象实例

}

}

那么此种情况下，我们如何用ObjectFactory来创建对应的对象实例呢？

做法如下：

(1). 在SO\_A\_Auditor、SO\_B\_Auditor、SO\_C\_Auditor的类的定义上，使用VersionExport的一个重载方法：

public VersionExportAttribute(Type type, string version, string[] filter)

关键在于第三个参数上，可以指定固定字符串的匹配条件，例如可以如下使用：

[VersionExport(typeof(IAuditor) , "3.0.0.0", new string[] { "A" } )]

public class SO\_A\_Auditor : IAuditor { … }

[VersionExport(typeof(IAuditor) , "3.0.0.0", new string[] { "B" } )]

public class SO\_B\_Auditor : IAuditor { … }

[VersionExport(typeof(IAuditor) , "3.0.0.0", new string[] { "C" } )]

public class SO\_C\_Auditor : IAuditor { … }

(2). 在创建对象时，使用ObjectFactory的一个重载方法：

public static T GetInstance(string version, string[] filters)

或者

public static T GetInstance(string[] filters)

通过filter参数的匹配，来获取对应的对象实例；

所以我们前面的CreateAuditor工厂方法可以修改为如下：

public IAuditor CreateAuditor(OrderType orderType)

{

switch(orderType)

{

case “A”:

return ObjectFactory< IAuditor >.GetInstance("3.0.0.0", new string[] { "A" });

case “B”:

return ObjectFactory< IAuditor >.GetInstance("3.0.0.0", new string[] { "B" });

default:

return ObjectFactory< IAuditor >.GetInstance("3.0.0.0", new string[] { "C" });

}

}

或者直接修改为：

public IAuditor CreateAuditor(OrderType orderType)

{

return ObjectFactory< IAuditor >.GetInstance("3.0.0.0", new string[] { orderType });

}