C# 利用Autofac批量接口注入依赖的问题小结

作者：有故事的人 来源：范文网 www.wtabcd.cn/fanwen/

本文原地址：https://www.wtabcd.cn/fanwen/zuowen/e5a5c43339c838acb574136cf015d880.html

背景：

本人在一位大佬的colder框架中看到了这个接口注入，然后呢就想学习一下ioc思想与di设计模式。此写法给我的感觉就是

非常的 优雅
，优雅永不过时。关于接口注入具体是什么可以最后推荐的地址。话不多说，开撸。

安装：

打开nuget管理工具，将我下面标红色的包都进行安装（注：千万别安装错了，按照名字不差的安装）

使用：

我们新建一个di的文件夹，在文件夹中增加一个接口：idependency.cs

namespace coldairarrow{ /// <summary> /// 注入标记 /// </summar
y> public interface idependency { }}

先不要问问什么后面会解释。

后面：其实。。就是这个依赖注入的一个原理吧。根据这个接口去找依赖的实现。

推荐去这个地址看一下：https://www.jb51.net/article/206284.htm

好了，继续分别新建student.cs，studentrepository.cs，istudentrepository.cs三个类。studentrepository.cs里面的具体业务根据需要自行修改，这里是为了测试使用。

using system;using system.collections.generic;using system.linq;using system.web;namespace byzk
api{ public class student { public int id { get; set; }

```
public string name { get; set; }  public string graduati
on { get; set; }  public string school { get; set; }  pu
blic string major { get; set; }  }}

using coldairarrow;using system;using system.collections.generic;using system.linq;using system.w
eb;namespace byzkapi{  public class studentrepository : istudentrepository,idependency
  {  public student add(student item)  {
   throw new notimplementedexception();  }  public
 bool delete(int id)  {  throw new notimplemente
dexception();  陈楚生 何洁  }  public student get(int id)
   {  return new student() { name = "张三" };
   }  public ienumerable<student> getall()  {
    throw new notimplementedexception();  }
 public bool update(student item)  {  throw new
 notimplementedexception();  }  }}

using system;using system.collections.generic;using system.linq;using system.text;using system.thre
ading.tasks;namespace byzkapi{  public interface istudentrepository  {
  ienu武汉职业技术学校merable<student> getall();  student get(int id);
   student add(student item);  bool update(student item);
  bool delete(int id);  }}

注意：这里好好看一下studentrepository 是实现了两个接口分别是
istudentrepositoryidependency（注入标记）

最关键的地方来了，我们打开项目启动的函数application_start，然后注入一下接口依赖。

using autofac;using autofac.extras.dynamicproxy;using autofac.integration.mvc;using autofac.integra
tion.webapi;using byzkapi.controllers;using byzkapi.interface;using coldairarrow;using microsoft.ext
ensions.dependencyinjection;using system.linq;using system.reflection;using system.web.compilation;
using system.web.http;using system.web.mvc;using system.web.optimization;using system.web.routi
ng;namespace byzkapi{  public class webapiapplication : system.web.httpapplication
 {  protected void application_start()  {
  //初始化autofac  initautofac(globalconfiguration.configuration);
   arearegistration.registerallareas();  globalconfigura
tion.configure(webapiconfig.register);  filterconfig.registerglobalfilters(globalfil
ters.filters);  routeconfig.registerroutes(routetable.routes);
  bundleconfig.registerbundles(bundletable.bundles);  }  priv
ate void initautofac(httpconfiguration config)  {  var
 builder = new contai小学生特长nerbuilder();  var basetype = t
ypeof(idependency);  //可以进行筛选如： where(x => x.fullname.con
tains("coldairarrow"))  var assemblys = buildmanager.getreferencedasse
mblies().cast<assembly>()  .tolist();  /
/自动注入idependency接口,支持aop,生命周期为instanceperdependency  b
uilder.registerassemblytypes(assemblys.toarray())  .where(x =>
basetype.isassignablefrom(x) && x != basetype)  .asimpleme
```

```
ntedinterfaces()            .propertiesautowired()
   .instanceperdependency()            .enableinterfaceinterceptors
()            .interceptedby(typeof(interceptor));
 //注册controller          builder.registercontrollers(assemblys.toarray())
          .propertiesautowired();          builder.registerapicont
rollers(assemblys.toarray()).propertiesautowired();                //aop
         builder.registertype<interceptor>();          builder.regist
erwebapifilterprovider(config);          var container = builder.build();
        dependencyresolver.setresolver(new autofacdependencyresolver(container));
         var resolver = new autofacwebapidependencyresolver(container);
        globalconfiguration.configuration.dependencyresolver = resolver;
     autofachelper.container = container;       }   }}
```

到此为止就已经注入完成了~

使用：

这个接口注入好了之后可以在普通的controller使用，也可以在apicontroller进行使用，分别看一下效果，结束。

controller：

```
using system;using system.collections.generic;using system.linq;using system.web;using system.we
b.mvc;namespace byzkapi.controllers{   public class homecontroller : controller
{       readonly istudentrepository repository;       //构造器注入
     public homecontroller(istudentrepository repository)       {
      this.repository = repository;       }       public actionr
esult index()       {          var a = repository.get(1);
        viewbag.title = a.name;          return view();
   }   }}
```

apicontroller：（如果想要多个接口只需要实现接口的时候进行继承idependency就可）

```
 using byzkapi.interface;using coldairarrow.web;using system;using system.collections.generic;using
 syste学习证书m.data;using system.linq;using system.web;using system.web.http;namespace byzk
  api.controllers{   public class testapicontroller : apicontroller   {
 readonly istudentrepository repository;       public itest _test { get; }
     //构造器注入       public testapicontroller(istudentrepository repository,
itest test)       {          this.repository = repository;
       _test = test;       }       [httpget]
 public datatable test(string sql)       {          repository.get(1);
          var data = _test.gettest("sql 语句");          //r
      epository.gettest(sql);          return data;       }   }}
```

到此这篇关于c# 利用autofac批量接口注入依赖的文章就介绍到这了,更多相关c# autofac批量注入内容请搜索www.887551.com以前的文章或继续浏览下面唐雎不辱使命翻译的相关文章希望大家以

后多多支持www.887551.com！