

除了构造函数注入(ServicesCollection支持，后两个不支持)，还有属性注入，方法注入

//调用

IZhaoxiContainer zhaoxiContainer = new ZhaoxiContainer();

zhaoxiContainer.Register<ITDAL, TDAL>();

zhaoxiContainer.Register<ITeestDAL, TeestDAL>();

zhaoxiContainer.Register<IUserDAL, UserDAL>();

zhaoxiContainer.Register<IUserBLL, UserBLL>();

IUserDAL iuserDAl = zhaoxiContainer.Resolve<IUserDAL>();

IUserBLL userBLL = zhaoxiContainer.Resolve<IUserBLL>();

//构造函数注入 bll或者dal构造函数是多个参数的，就是用下面的递归方法，递归注入

public class UserBLL : IUserBLL

{

private IUserDAL \_userDAL;

private ITeestDAL \_testDAL;

private ITDAL \_ tdal;

[ZhaoxiConstructorAttribute]

public UserBLL(IUserDAL userDAL, ITeestDAL testDAL)

{

this.\_userDAL = userDAL;

this.\_testDAL = testDAL;

}

public UserBLL(IUserDAL userDAL, ITeestDAL testDAL,ITDAL tdal)

{

this.\_userDAL = userDAL;

this.\_testDAL = testDAL;

this.\_tdal = tdal;

}

public void LastLogin(UserModel user)

{

user.LoginTime = DateTime.Now;

this.\_userDAL.Update(user);

}

public UserModel Login(string account)

{

return this.\_userDAL.Find(u => u.Account.Equals(account));

}

}

/// <summary>

/// 只能在构造函数上使用，标记优先是用的构造函数

/// </summary>

[AttributeUsage(AttributeTargets.Constructor)]

public class ZhaoxiConstructorAttribute : Attribute

{

}

public interface IZhaoxiContainer

{

/// <summary>

/// 约定TTo必须是TForm的子类，防止乱写

/// </summary>

/// <typeparam name="TFrom"></typeparam>

/// <typeparam name="TTo"></typeparam>

void Register<TFrom, TTo>() where TTo : TFrom;

TFrom Resolve<TFrom>();

}

public class ZhaoxiContainer : IZhaoxiContainer

{

private Dictionary<string, Type> dic = new Dictionary<string, Type>();

public void Register<TFrom, TTo>() where TTo : TFrom

{

this.dic.Add(typeof(TFrom).FullName, typeof(TTo));

}

public TFrom Resolve<TFrom>()

{

return (TFrom)this.ResolveObject(typeof(TFrom));

}

/// <summary>

/// 递归调用-- A调用了B，B调用了C,不知道会调用多少个，所以用递归

/// </summary>

/// <param name="abstractType"></param>

/// <returns></returns>

public object ResolveObject(Type abstractType)

{

var key = abstractType.FullName;

Type type = this.dic[key];

//准备构造函数的参数（暂时只获取第一个构造函数）

var ctor = type.GetConstructors()[0];

List<object> list = new List<object>();

foreach (var para in ctor.GetParameters())

{

Type paratype = para.ParameterType;//获取参数的类型 IUserDAL

object parainstance = this.ResolveObject(paratype);

}

Object obj = Activator.CreateInstance(type, list.ToArray());

return obj;

}

}