IOC容器:AutoFac

ContainerBuilder builder=new ContainerBuilder();

//把UserBll注册为IUserBll实现类

//builder.RegisterType<UserBll>().As<IUserBll>();

builder.RegisterType<UserBll>().AsImplementedInterfaces();

IContainer container=builder.Build();

//创建IUserBll实现类的对象

IUserBll bll=container.Resolve<IUserBll>();//new UserBll

bll.AddNew(“aa”,”123”);

ContainerBuilder builder=new ContainerBuilder();

Assembly asb=Assembly.Load(”MyBllImpl”);

//builder.RegisterAssemblyTypes(asm).AsImplementedInterfaces();

builder.RegisterAssemblyTypes(asm).AsImplementedInterfaces().PropertiesAutowired().SingleInstance();

IContainer container=builder.Build();

IUserBll bll=container.Resolve<IUserBll>();

PropertiesAutowired 只有被autofac创建出来的对象才会被属性自动注入

AutoFac生命周期

1. PerDependency 每次Resovle都返回一个新的对象 InstancePerDependency()
2. SingleInstance:单利,每次请求返回同一个对象:SignleInstance()
3. PerLifetimeScope:每个生命周期一个对象
4. InstancePerRequest:Asp.Net Mvc专用,每个请求一个对象.InstancePerRequest()

建议autoFac对象最好配置成无状态的(实现类中不要有成员变量),使用单例方式.

MVC中controller和属性自动注入 在global的 application\_start 中写入

//继承自IController的类自动注入,属性也自动注入

var builder = new ContainerBuilder();

builder.RegisterControllers(typeof(MvcApplication).Assembly).PropertiesAutowired().InstancePerRequest();

Assembly asmService = Assembly.Load("TestService");

builder.RegisterAssemblyTypes(asmService).Where(t => !t.IsAbstract).AsImplementedInterfaces();

var container = builder.Build();

DependencyResolver.SetResolver(new AutofacDependencyResolver(container));

没法自动注入是获取service对象: IUserServeice svc=DependecyResolver.

自定义JsonNetResult

public class JsonNetResult:JsonResult

{

public JsonSerializerSettings Settings { get; private set; }

public JsonNetResult()

{

Settings = new JsonSerializerSettings

{

// 忽略循环引用,如果设置为ERROR,则遇到循环引用的时候报错(建议设置为Error,这样更规范)

ReferenceLoopHandling = ReferenceLoopHandling.Ignore,

//日期格式化,默认的格式也不好看

DateFormatString = "yyyy-MM-dd HH:mm:ss",

//Json中属性开头字母小写的驼峰命名

ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver()

};

}

public override void ExecuteResult(ControllerContext context)

{

if (context == null)

throw new ArgumentException("content");

if (this.JsonRequestBehavior == JsonRequestBehavior.DenyGet && string.Equals(context.HttpContext.Request.HttpMethod, "GET",StringComparison.OrdinalIgnoreCase)){

throw new InvalidOperationException("JSON GET is not allowed");

}

HttpResponseBase response = context.HttpContext.Response;

response.ContentType = string.IsNullOrEmpty(this.ContentType) ? "application/json" : this.ContentType;

if (this.ContentEncoding != null)

{

response.ContentEncoding = this.ContentEncoding;

}

if (this.Data == null) { return; }

var scriptSerializer = JsonSerializer.Create(this.Settings);

scriptSerializer.Serialize(response.Output, this.Data);

}

}

利用ActionFilter降低侵入性

public class JsonNetActionFilter : IActionFilter

{

public void OnActionExecuted(ActionExecutedContext filterContext)

{

if(filterContext.Result is JsonResult&&!(filterContext.Result is JsonNetResult))

{

JsonNetResult jsonNetResult = new JsonNetResult();

JsonResult jsonResult = (JsonResult)filterContext.Result;

jsonNetResult.ContentEncoding = jsonResult.ContentEncoding;

jsonNetResult.ContentType = jsonResult.ContentType;

jsonNetResult.Data = jsonResult.Data;

jsonNetResult.JsonRequestBehavior = jsonResult.JsonRequestBehavior;

jsonNetResult.MaxJsonLength = jsonResult.MaxJsonLength;

jsonNetResult.RecursionLimit = jsonResult.RecursionLimit;

filterContext.Result = jsonNetResult;

}

}

public void OnActionExecuting(ActionExecutingContext filterContext)

{

}

}

global 的application\_start 方法中添加

GlobalFilters.Filters.Add(new JsonNetActionFilter());