记录修改

1.

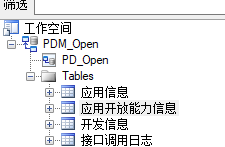


彪哥增加了一个web api的过滤器。



2.

数据库——open

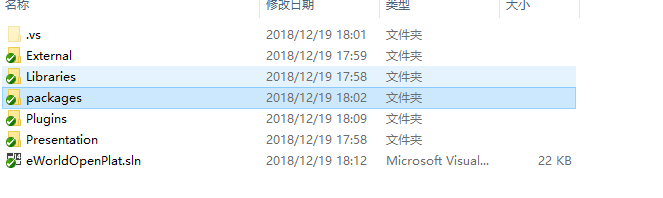




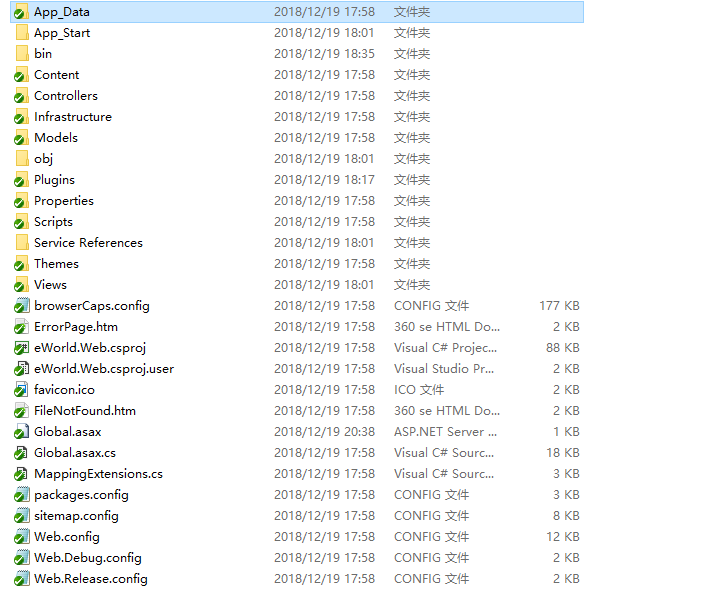
3.

彪哥教导

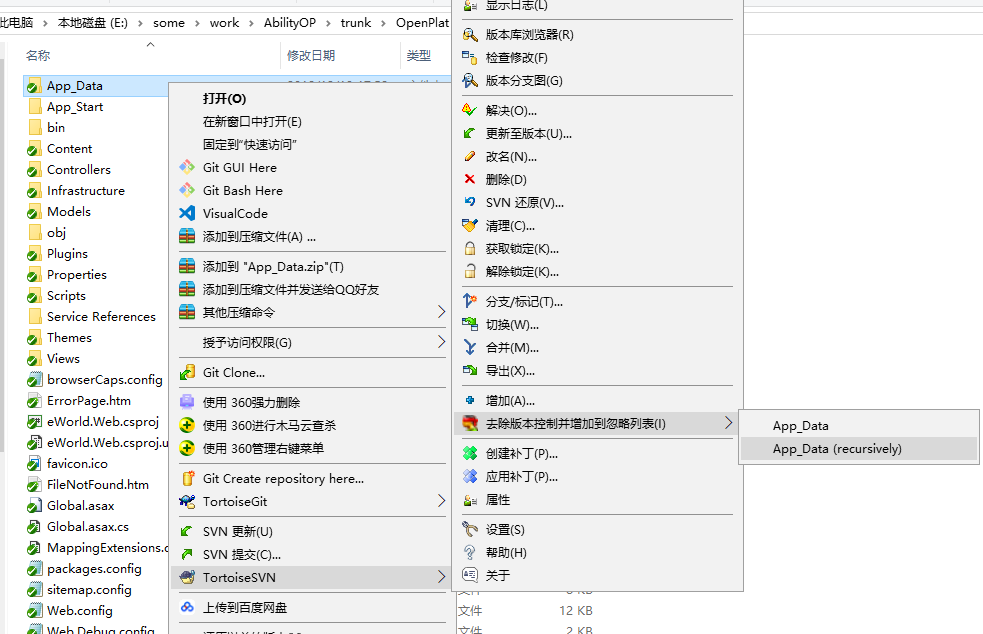
packages文件夹不需要上传



App\_Data，App\_Start，bin，obj文件夹都不要提交的。

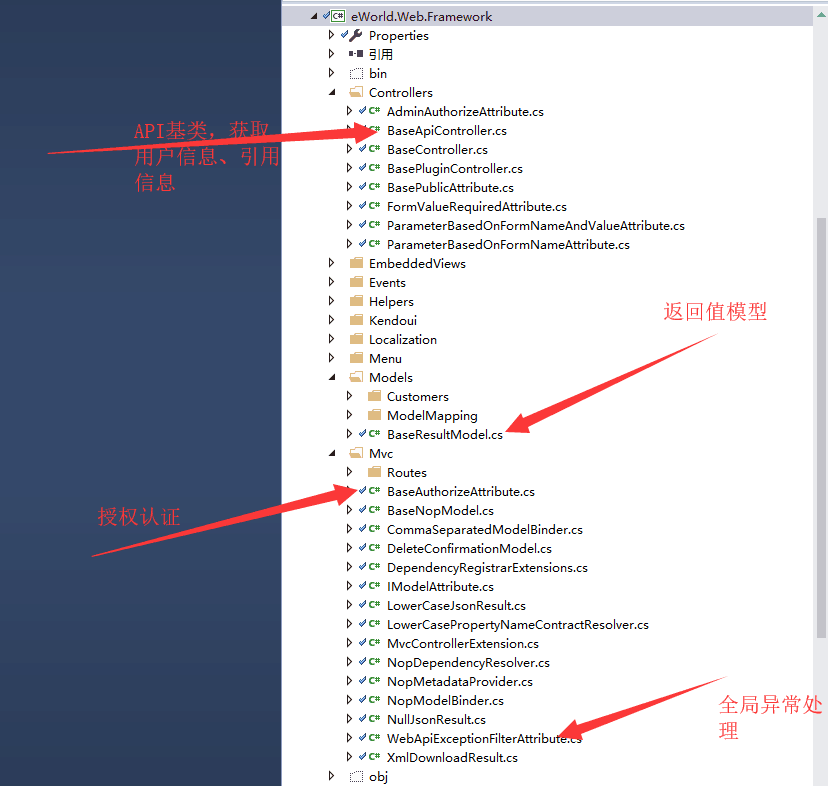


增加到忽略当中。



还有，nuget装包的时候，不要只管自己，要把其他用到的，也都一起都装了。

4.



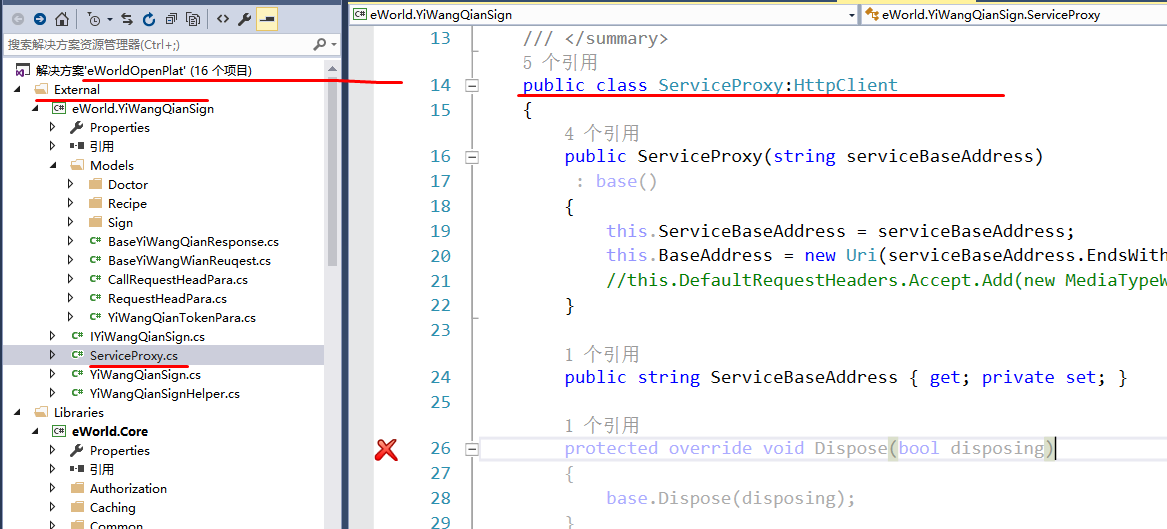


5.

调用wcf的时候，能直接用对方返回的model，就直接用，必须建立model的时候才建类，比方要调用三个接口，然后组装信息，就需要自己建立类了。

6.

调用api，可以用扩展里面的医网签的类。



7.

成长的一步

public static T To<T>(object value)

{

//return (T)Convert.ChangeType(value, typeof(T), CultureInfo.InvariantCulture);

if (value != null) return (T)To(value, typeof(T));

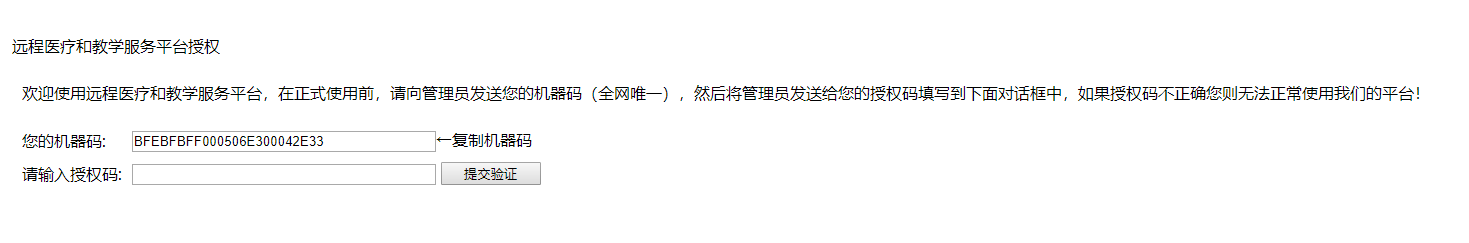
return default(T);

}

上面的方法报错，说是未将引用对象引用到实例。

按照Resharper的提示判断的存在，又返回了default(T)，然后就好了。

感觉很好。



8.

IHFIHIHH777871F477762F44

/Plugin/List

浙江-蜗牛（彪哥） 2018/12/11 14:00:12

这个地址下面是插件管理界面

浙江-蜗牛（彪哥） 2018/12/11 14:00:29

Admin

tomtaw

浙江-蜗牛（彪哥） 2018/12/11 14:00:31

这是账号

浙江-蜗牛（彪哥） 2018/12/11 14:00:55

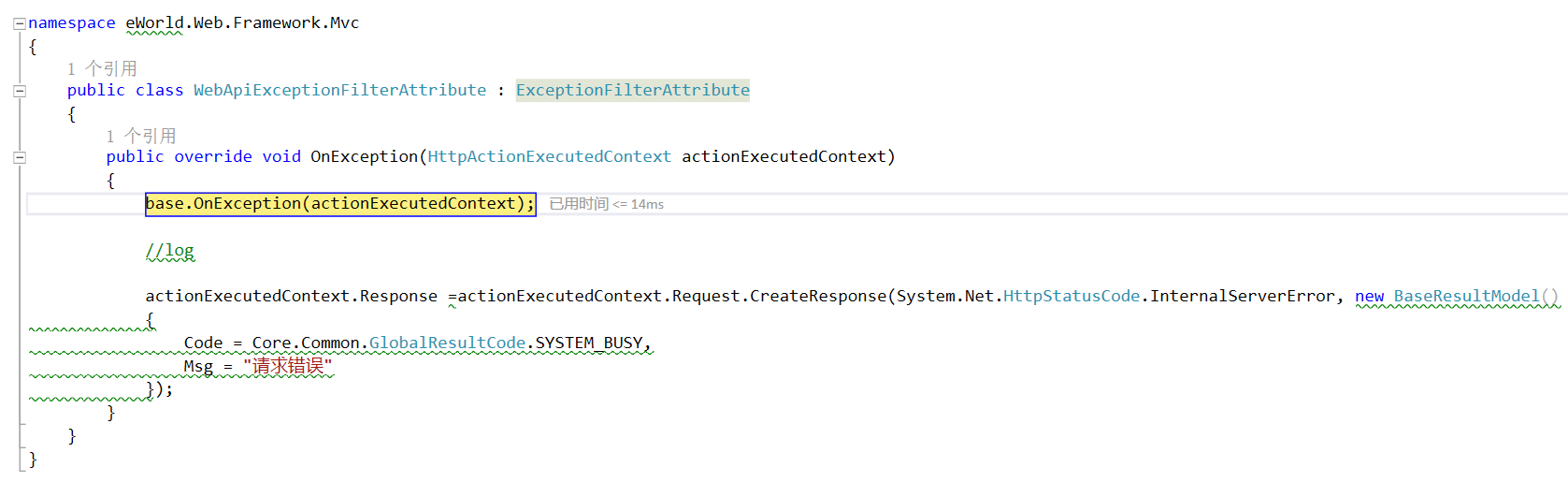
检查下是不是没有安装插件

9.

军哥这里学到一招：

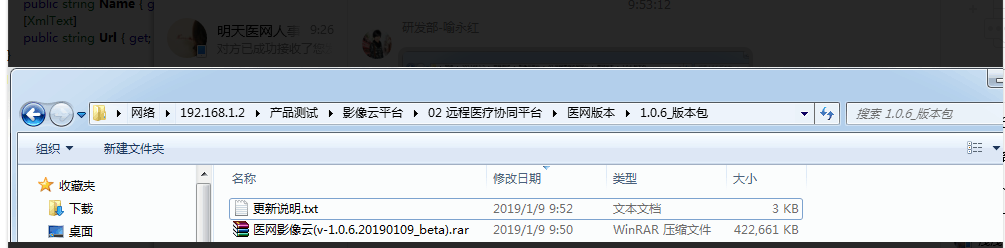
到这里面查看没有获取的异常。

我前面是Try-Catch住查看的。



10.

看到喻大神那里的测试地址：



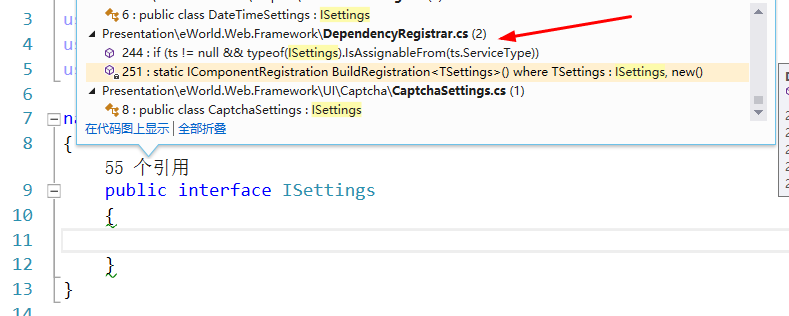
11.

热心的彪哥：

nuget包，公司的推不上，可以用我本机的  C:\Users\ywb\AppData\Local\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttp://192.168.1.91:9995/ C:\Users\ywb\AppData\Local\Temp\U[P7BL{JJDUR31F{Z`1_H4K.gif

12.

配置表的批量注入：



不知道是不是就是这个：

public IEnumerable<IComponentRegistration> RegistrationsFor(

Service service,

Func<Service, IEnumerable<IComponentRegistration>> registrations)

{

var ts = service as TypedService;

if (ts != null && typeof(ISettings).IsAssignableFrom(ts.ServiceType))

{

var buildMethod = BuildMethod.MakeGenericMethod(ts.ServiceType);

yield return (IComponentRegistration)buildMethod.Invoke(null, null);

}

}

static IComponentRegistration BuildRegistration<TSettings>() where TSettings : ISettings, new()

{

return RegistrationBuilder

.ForDelegate((c, p) =>

{

var currentWebSiteId = c.Resolve<IWebSiteContext>().CurrentWebSite.Id;

//uncomment the code below if you want load settings per WebSite only when you have two WebSites installed.

//var currentWebSiteId = c.Resolve<IWebSiteService>().GetAllWebSites().Count > 1

// c.Resolve<IWebSiteContext>().CurrentWebSite.Id : 0;

//although it's better to connect to your database and execute the following SQL:

//DELETE FROM [Setting] WHERE [WebSiteId] > 0

return c.Resolve<ISettingService>().LoadSetting<TSettings>(currentWebSiteId);

})

.InstancePerHttpRequest()

.CreateRegistration();

}

13.

（1）彪哥说了接下来要统一参数，我是用VO，Model去处理。到时候整体定下来再去重构。

（2）现在的接口返回是用object，可以用彪哥的注入方法

protected void Application\_Start()

{

GlobalConfiguration.Configuration.Formatters.Clear();

GlobalConfiguration.Configuration.Formatters.Add(new System.Net.Http.Formatting.JsonMediaTypeFormatter()

{

SerializerSettings = new JsonSerializerSettings()

{

ContractResolver = new LowerCasePropertyNameContractResolver()

}

});

但是如果返回值是ActionResult，就不会执行彪哥这个方法：

而要用这种扩展类静态方法的方式：

namespace eWorld.Web.Framework.Mvc

{

public static class MvcControllerExtension

{

public static LowerCaseJsonResult LowerCaseJson(this Controller controller, object data)

{

return new LowerCaseJsonResult(data);

}

}

}

应用示例:

//获取开放能力字典信息

[HttpGet]

public ActionResult GetAbilityTypeInfo()

{

data.Data = result;

data.Code = GlobalResultCode.SUCCESS;

data.Msg = "获取开放能力列表成功";

return this.LowerCaseJson(data);

}

（3）将Data转变成data，还有以后要做的json压缩算法

namespace eWorld.Web.Framework.Mvc

{

public class LowerCaseJsonResult : JsonResult

{

public LowerCaseJsonResult() { }

public LowerCaseJsonResult(object data)

{

Data = data;

}

private static JsonSerializerSettings jsonSetting = new JsonSerializerSettings

{

ContractResolver = new LowerCasePropertyNameContractResolver()

};

public override void ExecuteResult(ControllerContext context)

{

var response = context.HttpContext.Response;

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

if (ContentEncoding != null)

response.ContentEncoding = ContentEncoding;

var serializedObject = JsonConvert.SerializeObject(Data, jsonSetting);

response.Write(serializedObject);

}

}

}

以后的加密算法就在上面这里了，比方协同，云归档，接口的返回内容可能都一kb了，太大了，就压缩下，可能就变成一半了。再返回。终于也就省了带宽。也就增加平台的吞吐，抗并发能力。

但是问题就是，app的请求，要在请求头里面加上算法的识别，不然无法访问还是上面的，彪哥说。

14.

jobtriggerinfo = Mapper.Map(jobTriggerInfo, jobtriggerinfo);

jobtriggerinfo = Mapper.Map<JobTriggerMst>(jobTriggerInfo);

两者的区别与整理。

上面一个才是当已经查询了数据库中的实体，再从入参映射过去，update的正确写法。

当其中不需要映射的，或是本事实体中就存在的数据，不会被覆盖。

而下面这种写法，纯粹是把A映射成B，而如果再将本来就存在的实体C去等于B，那就造成所有的数据被覆盖了。

就好比有一个共有参数Flag，默认是false的。而实体C中是True。

当用前者时，如果设置了ignore，赋值之后仍旧是True。

而如果是下面这个，当使用映射之后，无论有没有Ignore，反正model B的flag肯定是false，当令C=B，那C就变成了B，Flag成了false。

但不明白，为什么这个时候，数据库update，数据中没有变化，不知道为什么。反正用前者，Map(A,B)这种是没有问题的。

15.

16.