记录修改

1.

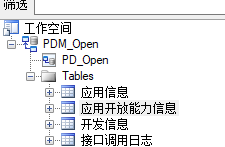


彪哥增加了一个web api的过滤器。



2.

数据库——open

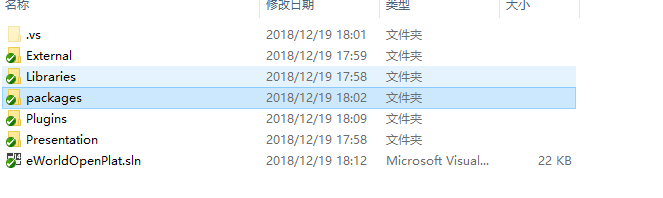




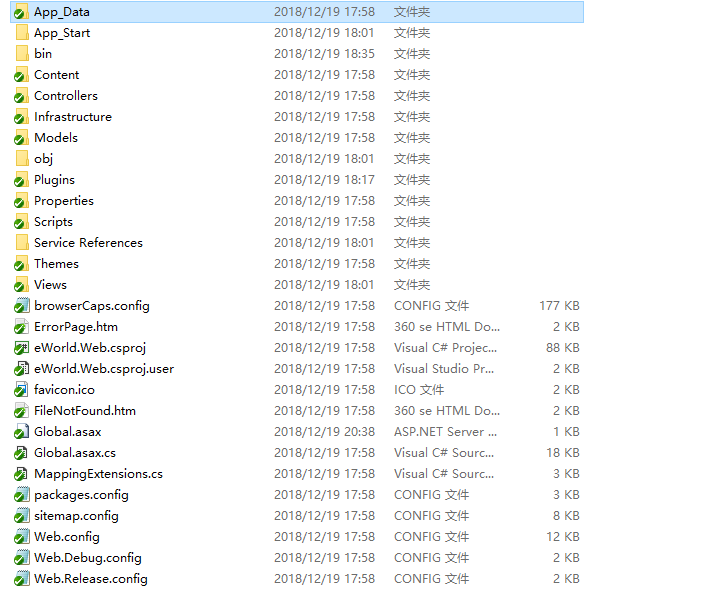
3.

彪哥教导

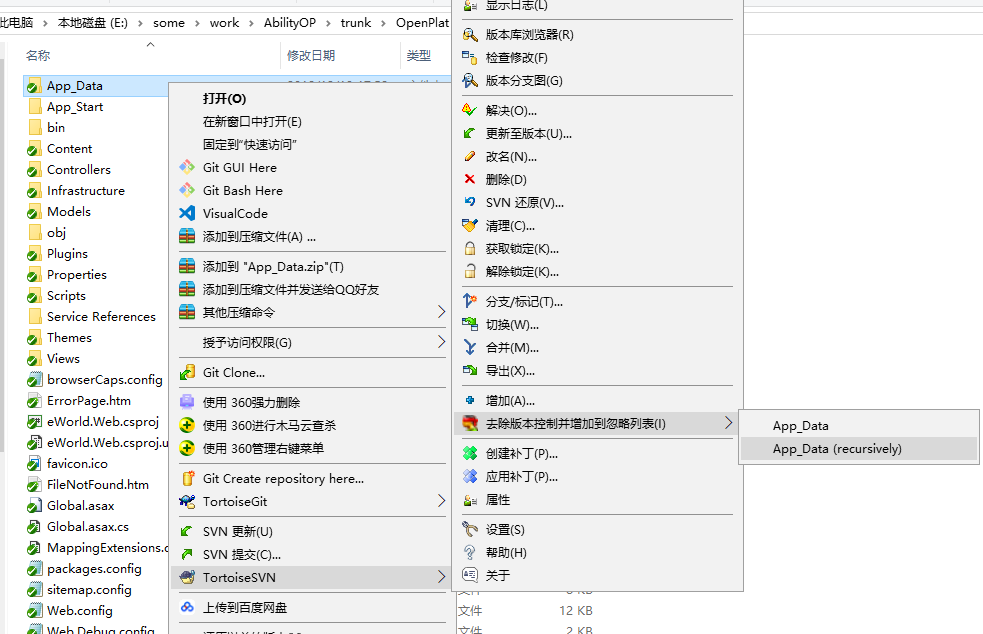
packages文件夹不需要上传



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

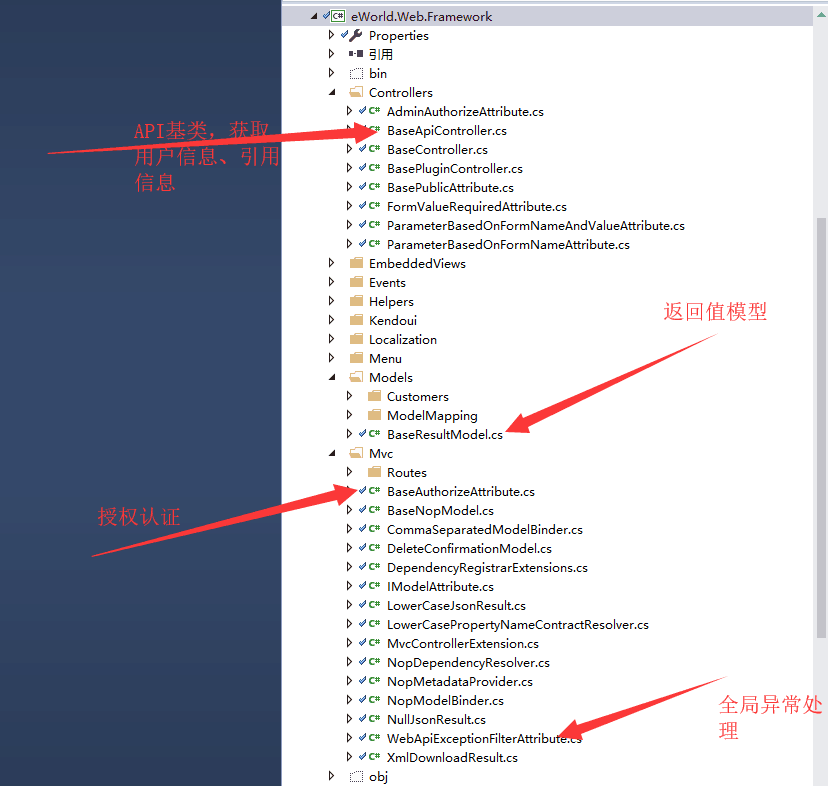


增加到忽略当中。



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.



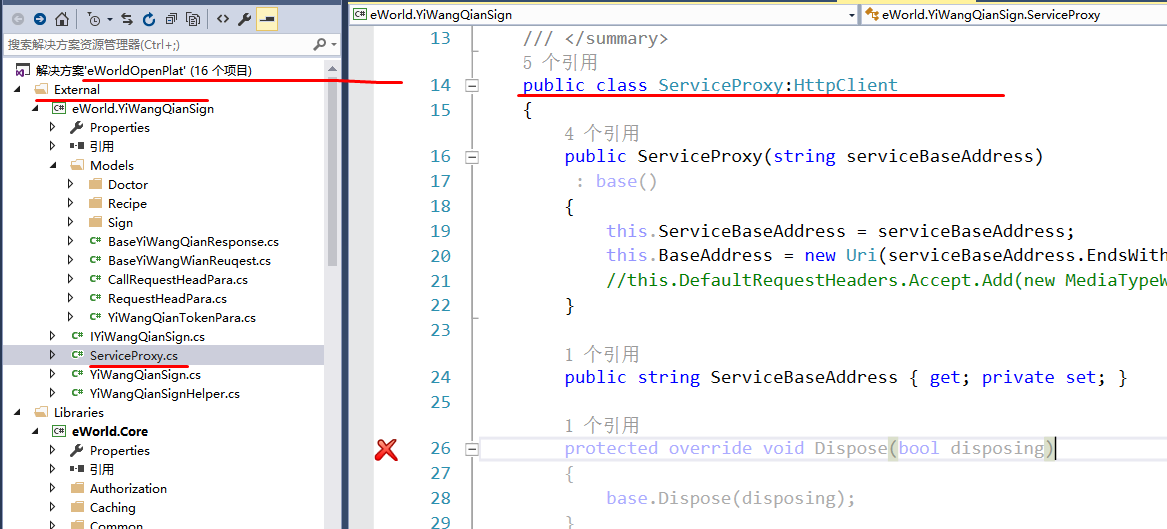


5.

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));

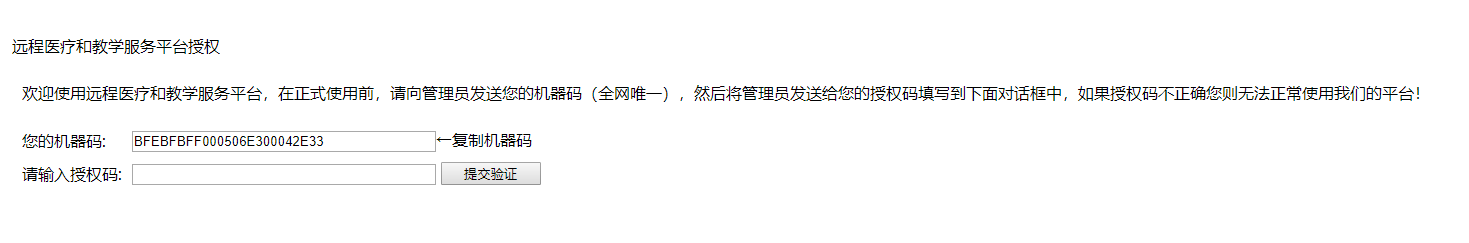
return default(T);

}

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

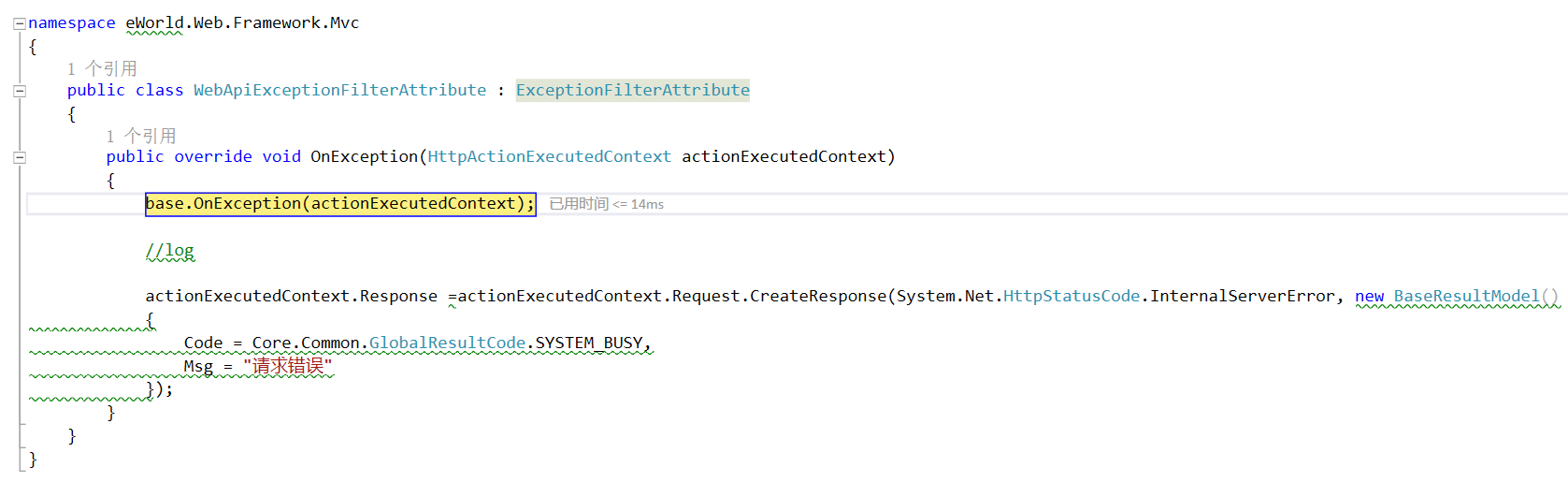
检查下是不是没有安装插件

9.

军哥这里学到一招：

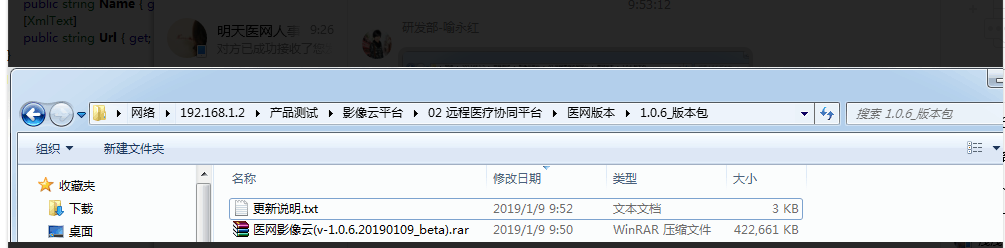
到这里面查看没有获取的异常。

我前面是Try-Catch住查看的。



10.

看到喻大神那里的测试地址：



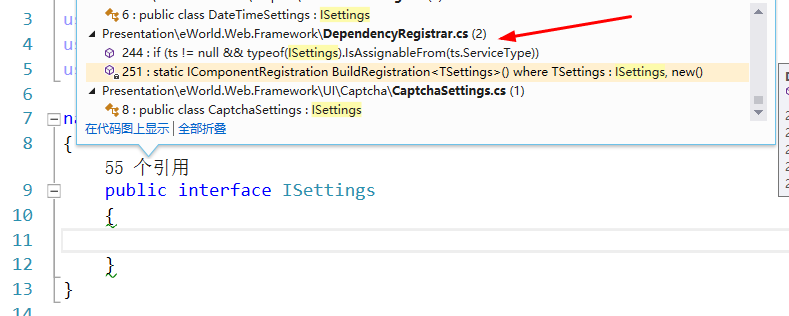
11.

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，或是本身实体中就存在的数据，不会被覆盖。

但是在SaveJobServer方法中，当组织ID，

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.

映射忽略问题，原来是查询出来结果，再根据结果中是否有值去忽略映射，

ObservationRequest obs = null;

if (examWithFileInfo.ObservationRequest.ObservationUID != null)

{

obs = \_repository.GetTable<ObservationRequest>().FirstOrDefault(t => t.ObservationUID == examWithFileInfo.ObservationRequest.ObservationUID);

}

if (obs.FilmCount.HasValue)

{

mapExpObs = mapExpObs.ForMember(t => t.FilmCount, dto => dto.Ignore());

}

if (!string.IsNullOrWhiteSpace(obs.PatientMasterID))

{

mapExpObs = mapExpObs.ForMember(t => t.PatientMasterID, dto => dto.Ignore());

}

if (!string.IsNullOrWhiteSpace(obs.PatientID))

{

mapExpObs = mapExpObs.ForMember(t => t.PatientID, dto => dto.Ignore());

我自己是感觉应该用入参，因为此时是update，已经查询出来了结果，当入参为null，则忽略映射，当入参不为null，则覆盖到当前查询出来的实体中的具体字段上面。

云归档这种映射+update的方法，就要求把所有没有忽略的入参都给写全了，才不出错。

还是喜欢协同的更新方式，先查询出该guid的具体实体，然后根据入参去赋值，最后update：

try

{

var itemModel = \_detectionService.GetModel(model.ServiceId);

if (itemModel == null)

{

root.code = ResultStatusCode.DATA\_DONOT\_EXIST;

root.msg = "查询检测记录失败，检测记录不存在";

return root;

}

//赋值更新

if (model.AddDetectionTimesFlag.HasValue && model.AddDetectionTimesFlag.Value)//检测次数是否可加1

{

itemModel.DetectionTimes = itemModel.DetectionTimes + 1;//检测次数

}

if (!string.IsNullOrWhiteSpace(model.DetectionResult))

{

itemModel.DetectionResult = model.DetectionResult;//检测结果

}

if (model.State.HasValue)

{

itemModel.State = model.State;//检测状态

}

itemModel.DetectionDate = itemModel.DetectionDate;//检测时间

\_detectionService.Update(itemModel);

}

协同的，也是同一个接口，有关更新和插入，就是插入用map映射，更新采用逐个替换：

Patient patient = \_patientService.Get(model.IdNumber);

if (patient == null)

{

ckResult = ckResult.Check(s => string.IsNullOrWhiteSpace(s.IdNumber), "患者身份证号码不能为空")

.Check(s => !s.PaperKind.HasValue, "证件类型不能为空")

.Check(s => string.IsNullOrEmpty(s.PatientSex), "患者性别不能为空")

.Check(s => string.IsNullOrWhiteSpace(s.PhoneCode), "联系电话不能为空")

.Check(s => string.IsNullOrEmpty(s.PhoneCode), "患者姓名不能为空")

.Check(s => string.IsNullOrEmpty(s.PatientName), "患者姓名不能为空");

patient = model.MapTo<Patient>();

patient.GPatientId = Guid.NewGuid();

\_patientService.Insert(patient);

}

else

{

if (!string.IsNullOrWhiteSpace(model.OtherPaperNumber))

patient.OtherPaperNumber = model.OtherPaperNumber;

if (!string.IsNullOrWhiteSpace(model.PatientName))

patient.PatientName = model.PatientName;

if (!string.IsNullOrWhiteSpace(model.PatientSex))

patient.PatientSex = model.PatientSex;

if (!string.IsNullOrWhiteSpace(model.Address))

patient.Address = model.Address;

if (!string.IsNullOrWhiteSpace(model.Company))

patient.Company = model.Company;

if (!string.IsNullOrWhiteSpace(model.Education))

patient.Education = model.Education;

if (!string.IsNullOrWhiteSpace(model.HealthCardNo))

patient.HealthCardNo = model.HealthCardNo;

if (!string.IsNullOrWhiteSpace(model.Height))

patient.Height = model.Height;

if (!string.IsNullOrWhiteSpace(model.InsuranceType))

patient.InsuranceType = model.InsuranceType;

if (!string.IsNullOrWhiteSpace(model.MedNo))

patient.MedNo = model.MedNo;

if (model.PaperKind.HasValue)

patient.PaperKind = model.PaperKind;

if (!string.IsNullOrWhiteSpace(model.PatientBirthday))

patient.PatientBirthday = model.PatientBirthday;

if (!string.IsNullOrWhiteSpace(model.PhoneCode))

patient.PhoneCode = model.PhoneCode;

if (!string.IsNullOrWhiteSpace(model.Weight))

patient.Weight = model.Weight;

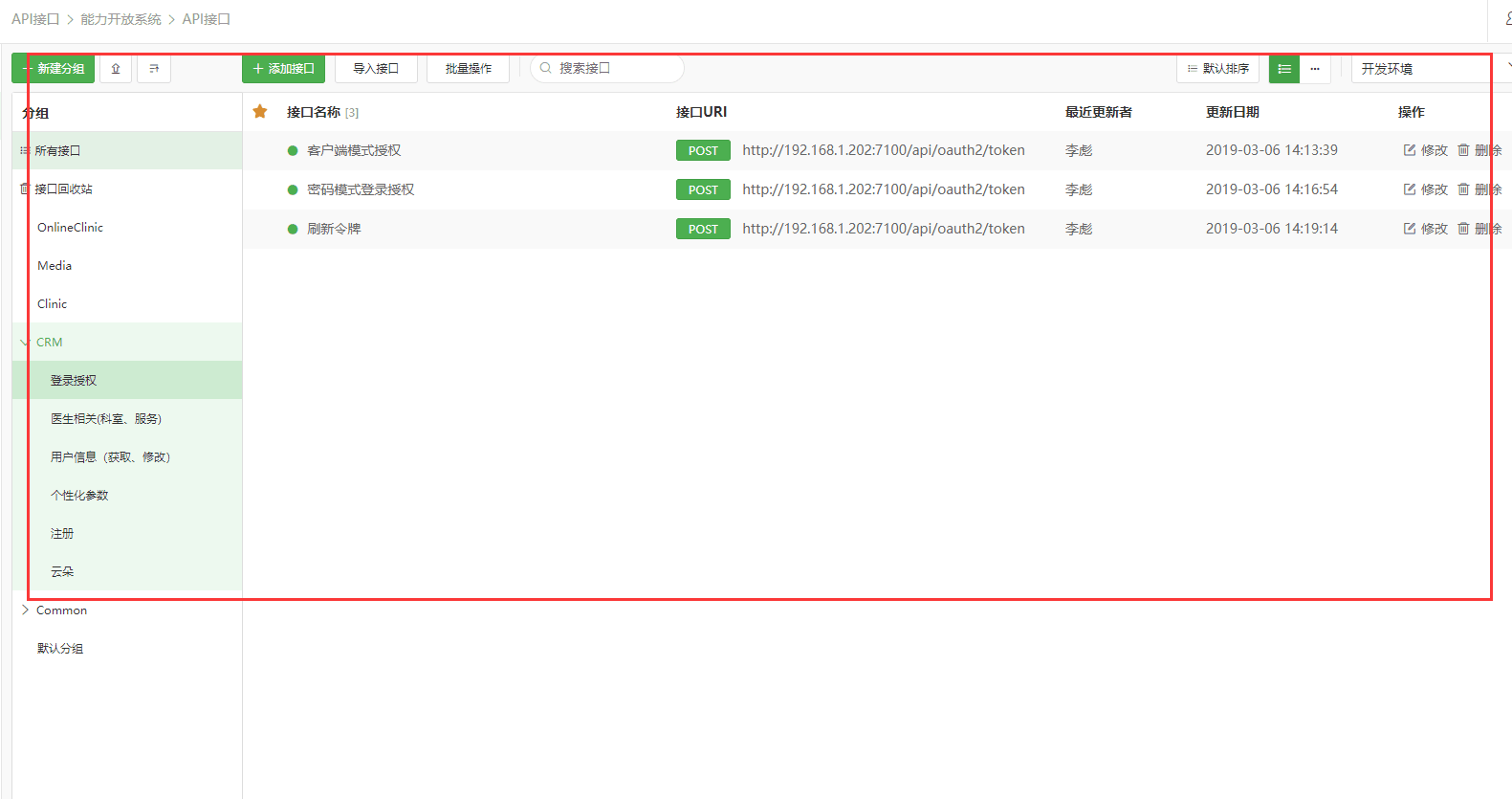
\_patientService.Update(patient);

}

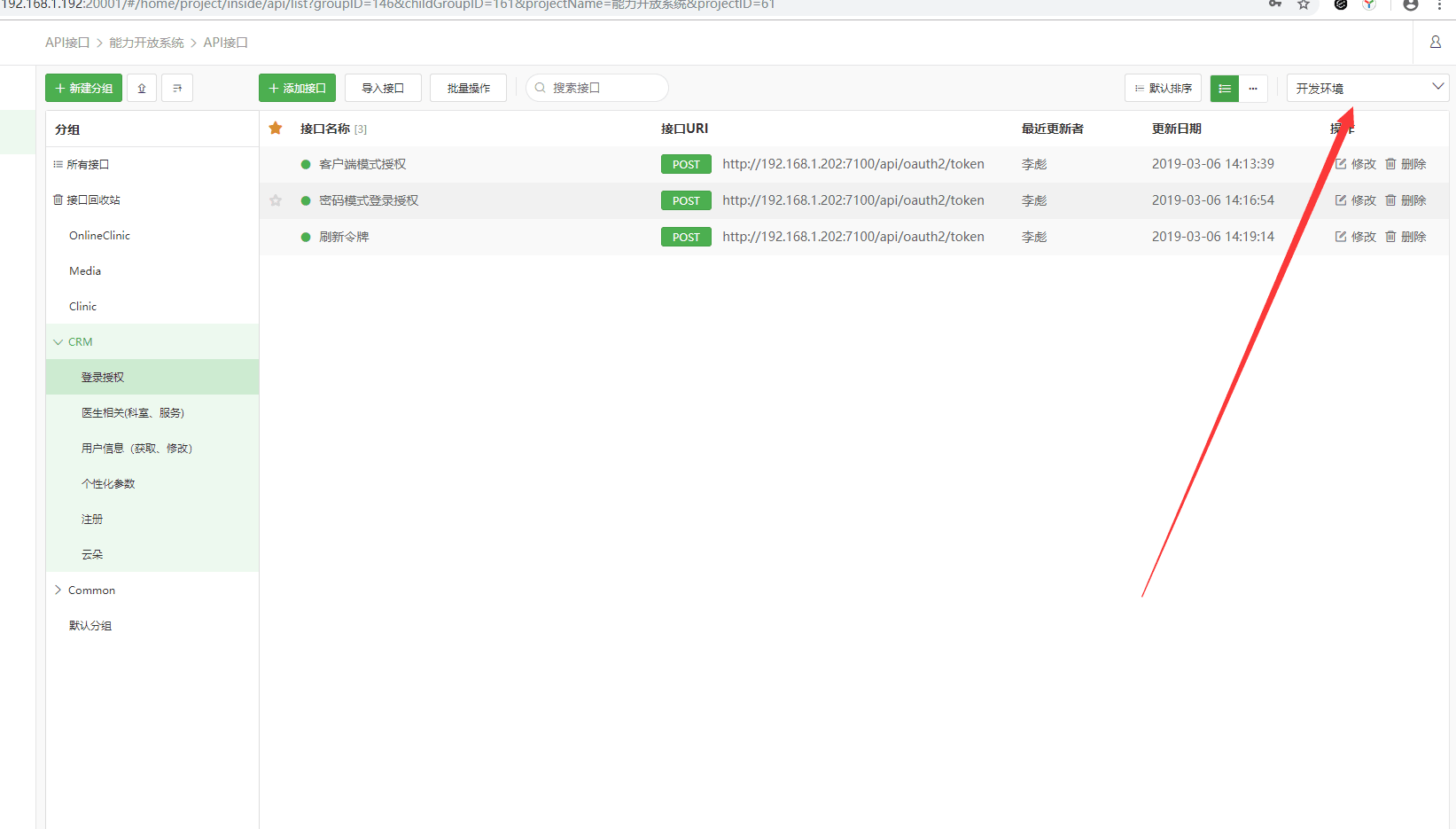
result.data = patient.GPatientId;

16.彪哥发布的授权

登录授权的接口文档加好了：



如果在网站上测试选开发环境



浙江-蜗牛（彪哥） 2019/3/6 14:22:20

我待会把内往外的环境加上去

浙江-蜗牛（彪哥） 2019/3/6 14:23:19

目前不涉及CRM相关的接口可以用客户端授权模式

浙江-蜗牛（彪哥） 2019/3/6 14:23:38

涉及到CRM的需要使用密码模式

17.

使用这个可以走进授权:



进不了一大步，抄了协同的授权过滤器，可以实现，也可以在重写方法中加上自己的判断。

摘录下协同的：

public class MvcAuthorizeAttribute : AuthorizeAttribute

{

/// <summary>

/// 授权过滤器

/// </summary>

/// <param name="httpContext"></param>

/// <returns></returns>

protected override bool AuthorizeCore(HttpContextBase httpContext)

{

CustomerPrincipal customerPrincipal = httpContext.User as CustomerPrincipal;

//这里要求当前登陆用户是存在的，并且授权验证是通过了的，而且单点登陆的是否登陆为登陆状态，就返回false。我可以参照。

if (customerPrincipal == null || !httpContext.User.Identity.IsAuthenticated || !EngineContext.Current.Resolve<ISsoProvider>().IsLogin)

{

httpContext.Response.StatusCode = 401;

return false;

}

上面的验证是通过的，但是不在可以访问的角色中

if (!customerPrincipal.IsInRole())

{

httpContext.Response.StatusCode = 403;

return false;

}

return true;

}

/// <summary>

/// 重写HandleUnauthorizedRequest身份验证

/// </summary>

/// <param name="filterContext"></param>

protected override void HandleUnauthorizedRequest(AuthorizationContext filterContext)

{

base.HandleUnauthorizedRequest(filterContext);

//这里处理错误的请求，不同的返回结果处理方式不同

但是授权，单点，不存在用户这些原因，转跳登陆页面，这里还存在是普通请求还是ajax请求，ajax证明是前端发来的，则给前端返回code msg这种标准化的结果。

if (filterContext.HttpContext.Response.StatusCode == 401)

{

string url = $"{HttpContext.Current.Request.Url.GetLeftPart(UriPartial.Authority).TrimEnd('/')}/Login";

if (!filterContext.HttpContext.Request.IsAjaxRequest())

{

filterContext.Result = new RedirectResult($"{url}?returnUrl={Uri.EscapeDataString(HttpContext.Current.Request.Url.PathAndQuery)}");

}

else

{

filterContext.Result = new JsonResult

{

Data = new { code = ResultStatusCode.VALID\_TOKEN\_FAIL, msg = "登录失效", url },

ContentType = null,

ContentEncoding = null,

JsonRequestBehavior = JsonRequestBehavior.AllowGet

};

}

}

当没有权限，则转跳没有权限页面

else if (filterContext.HttpContext.Response.StatusCode == 403)

{

if (!filterContext.HttpContext.Request.IsAjaxRequest())

filterContext.Result = new RedirectResult("/Common/RedirectPage?msg=您无权访问该网站'</script>");

else

filterContext.Result = new JsonResult

{

Data =

new

{

code = ResultStatusCode.UNAUTHORIZED,

msg = "非法访问",

url = "/Common/RedirectPage?msg=您无权访问该网站"

},

ContentType = null,

ContentEncoding = null,

JsonRequestBehavior = JsonRequestBehavior.AllowGet

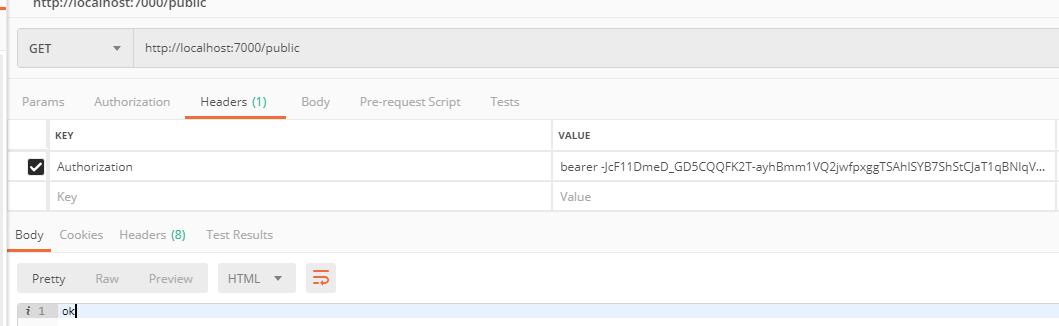
};

}

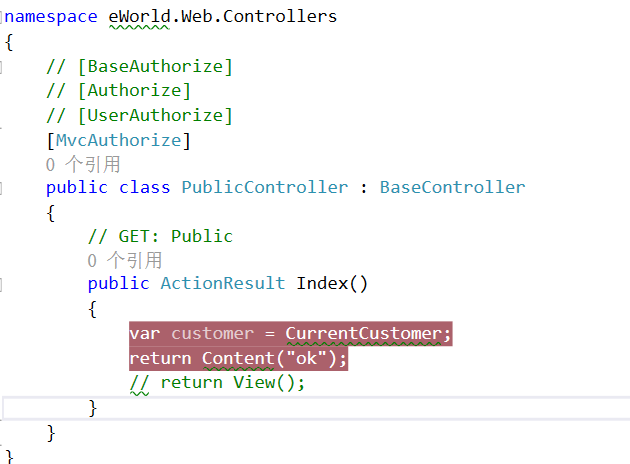
}

}

验证了，只有是可以用的。协同之前，应该是把Authorization放进cookie中的，然后解密出来。



直接集成军哥的BaseController是拿得到结果的，但如果是继承BaseAdmin控制器，会走不进这个方法。显示连接不到这个接口。



新建了一个TestBaseController，去集成BaseController，复制了彪哥BaseApi里面的代码，发现拿到了guid，userId，还有clientId的值。而BaseController里面的customer是拿不到的，从token里面。因为现在的授权，能拿到的只有这三项?

[MvcAuthorize]

public class PublicController : TestBaseController

{

// GET: Public

public ActionResult Index()

{

var customer = CurrentCustomer;

var guid = CurrentCustomerGuid;

var userId = CurrentUserId;

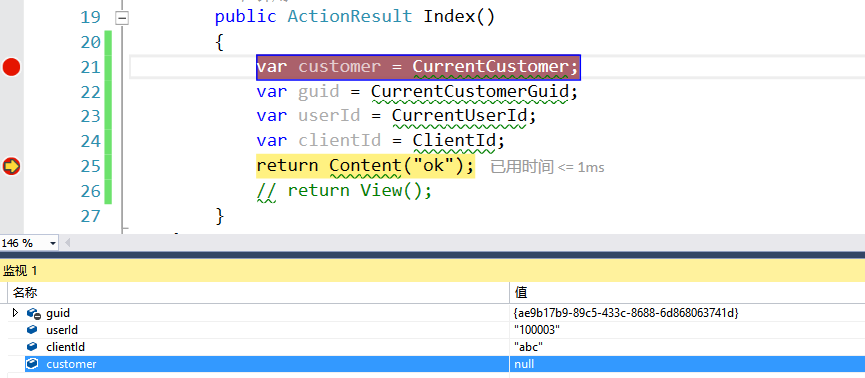
var clientId = ClientId;

return Content("ok");

// return View();

}

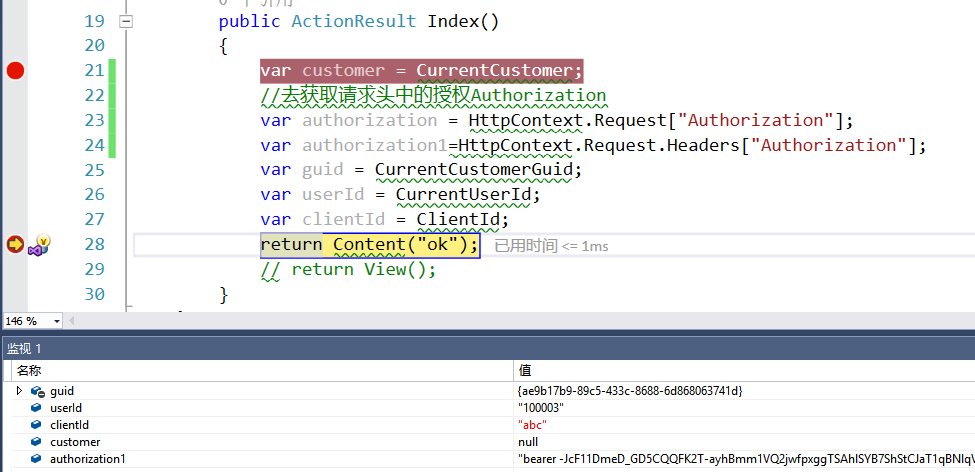
}



如果走我这边的逻辑，我是重写后判断授权是否通过，如果通过了，且获取了clientId的值，就放行？或者这里只要授权通过了，就放行?

如果不通过，就跳到无权限页面。

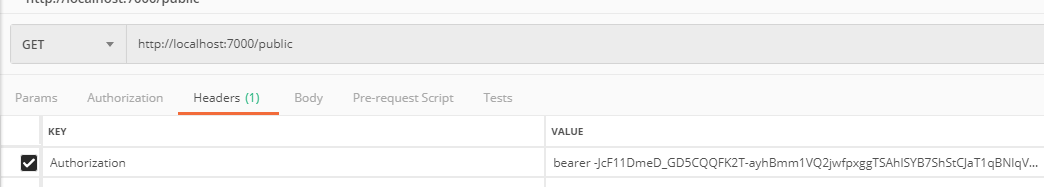
验证了可以从头里面获取到这个token：



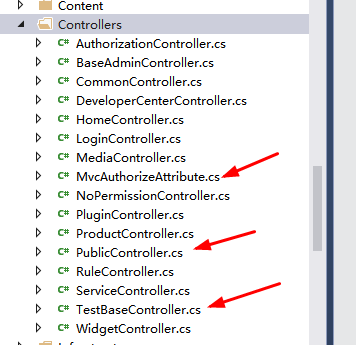
从头里面拿到授权码了：

"bearer -JcF11DmeD\_GD5CQQFK2T-ayhBmm1VQ2jwfpxggTSAhlSYB7ShStCJaT1qBNlqV4vYzZBLEP\_CHaOliaNxADOUaxkilG691P9Odgax--6FJ1nbtV6jnk5QAwgbtakP4-X6ms3ua03h2K8-MtLPpKmQzxNbcifhyWc\_dNc\_QYbwwzi4Yie5Ufm3sXPsaNU6uIt2-tE9dWTlcu6-I96\_4IEIXMaff-adL50R0cHpagy2RBnlXuynHfQ2hqZ-3WW0ElCdTTB0KUO9cnq0VFEbC9dFd2XPJJIdV3h-yeyvEMlSS5wE9i3vFD97LJEAh7CboDvScQvnGeGmk0WEYGudBAXjxnbNA"

postman的请求：



这三个最后删除



18.

杨杨那边获取的时间多了给T字，因为json的缘故，wcf过来是正常的。

查了后，在彪哥的Global里面加了格式。

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

{

SerializerSettings = new JsonSerializerSettings()

{

// 新增时间返回格式

DateFormatString = "yyyy-MM-dd HH:mm:ss",（这里是我加的）

ContractResolver = new LowerCasePropertyNameContractResolver()

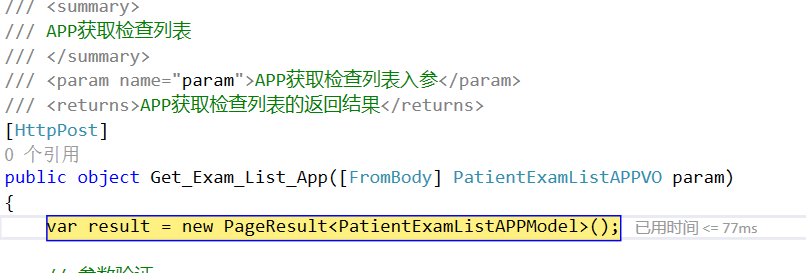
}

});

调试，果然，进入控制器后，在验证授权之后，走到了这个最中结果返回的json处理中：



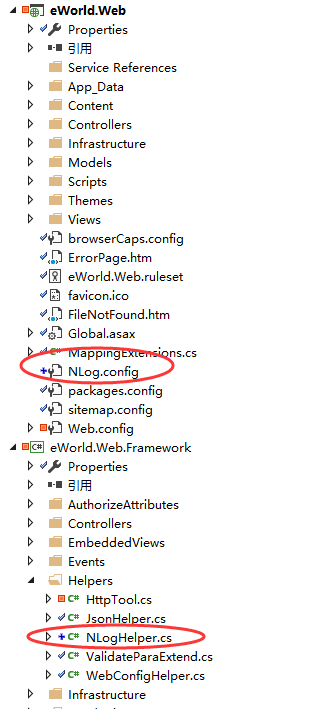
然后再进入到方法中：



19.

浙江-蜗牛（彪哥） 2019/3/13 14:25:19

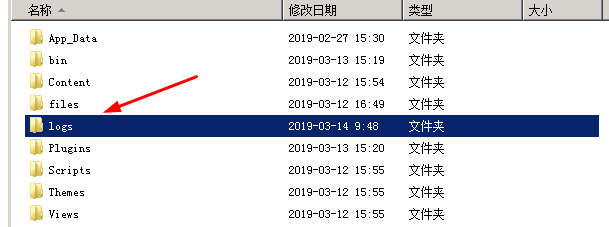
@张军 @研发部-黄铿 能力开放增加了日志输出



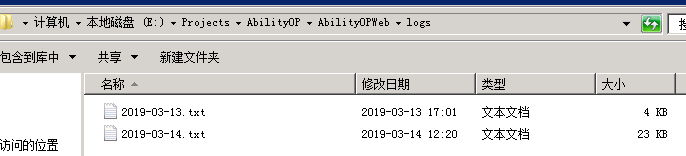
svn记录：

增加Nlog日志帮助类以及配置文件

看了下202，有了。



已经有内容了：



20.

分页结果，我是code，msg，page，再data比方用automap的list去映射。

军哥是用了静态方法：

//获取云朵服务器信息

[HttpGet]

public object get\_cloud\_pagelist([FromUri]PageParameter page)

{

var clouds = \_crmClient.Invoke(s => s.GetPageListCloud(page.page\_index, page.page\_size));

return clouds.MappingPageResult<CloudDto, CloudModel>();

}

这个分页映射的静态方法，其实内部和我写的代码是一样的，也是code，msg，page分别相等，然后再用automap去映射list的data结果集。

public static class DataStructureConvert

{

/// <summary>

/// pageindex转换,泛型不能自己转换，故用静态方法转换

/// </summary>

public static PageResult<NewObj> MappingPageResult<T, NewObj>(this PageResult<T> source, bool isBegin1 = false)

{

if (source == null)

return null;

var ret = new PageResult<NewObj>();

ret.code = source.code;

ret.msg = source.msg;

ret.page = source.page;

if (ret.page != null && isBegin1 == false)

ret.page.page\_index += 1;

if (source.data != null)

ret.data = Mapper.Map<List<T>, List<NewObj>>(source.data);

return ret;

}

这样处理的逻辑就是，直接把契约的类复制过来，一模一样，打上json的标签，然后初始化里面直接建立map，这样，函数体内部直接映射就可以了，因为参数类型和名字都是一致的。只是进来和出去的json参数限制了下。完美啊，简直。

Mapper.CreateMap<CustomConfigParameterModel, CustomConfigParameter>();

21.

彪哥提到未来的能力开放平台的趋势，所有的api接口都往这里走，到时候前后端分离，web端直接调能力开放平台的接口，这样前端玩自己的去就可以了。就不走目前紧耦合的mvc模式了。

另外，接口如果要升级，又要兼容之前的，那直接新建一个云归档的插件，开发该接口，然后，路由规则的路径增加一个v2就可以了。调的就是新的插件了，但老的接口也存在着。这样能力开发平台发布出去，就兼容所有的新旧各个版本，而只需要一个发布版本了。

接口会自己根据路径走到对应的插件中。

22.

异常的处理，不需要try-catch，有问题，直接throw，上抛，统一处理。

彪哥自己调http接口，加了try-catch。

23.

24.

25.