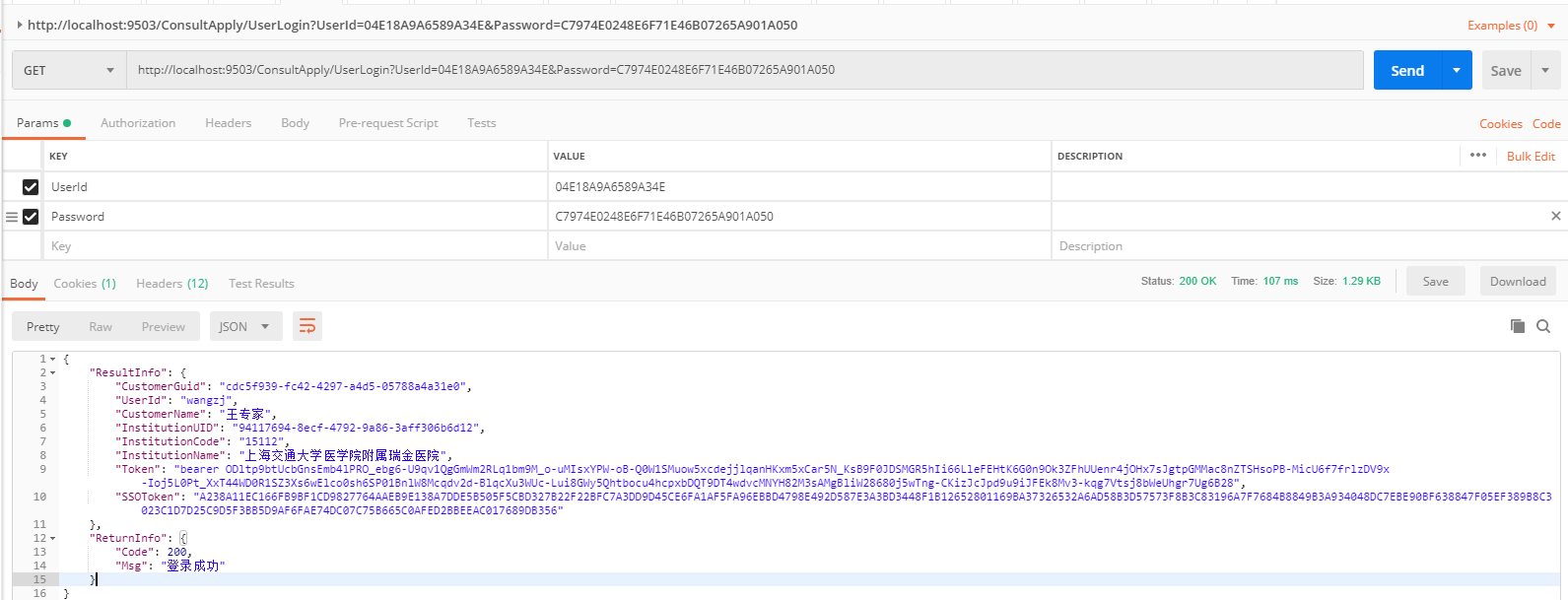
接口的一次整理

1.

查看这个获取Token的接口

<http://localhost:9503/ConsultApply/UserLogin?UserId=04E18A9A6589A34E&Password=C7974E0248E6F71E46B07265A901A050>



（1）get请求，将影像云平台的用户名和密码以DES加密的方式传给接口。

接口接受，开始将参数进行处理。

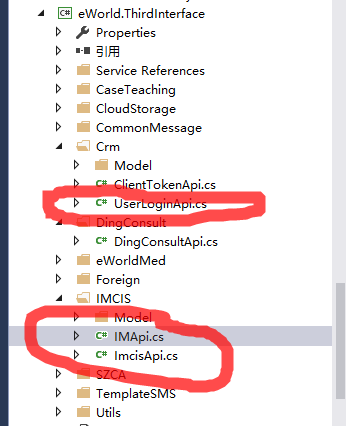
（2）BLL层，逻辑处理

a.

（2.1）

接口内部，在BLL层调用CRM的接口，和我后面写的调用IM的接口在同一个第三方接口的位置。

其中内部都是调用了WebHelper的不同方法。



（2.2）

在bll层访问配置文件中的地址，直接在BLL层加一个这个配置类。

ApiUrl = CRMSettings.WebApiIntranetUrl

//远程医疗协同设置

private static readonly CRMSettings CRMSettings = EngineContext.Current.ContainerManager.Resolve<CRMSettings>();

（2.3）

这里针对传来的DES加密后的密码先解密，然后再加上统一的签名Key，重新进行DES加密，然后将DES加密后的账号，密码，封装成报文体，然后发送给固定的，CRM的api，获取结果。

var encryptPassword = RSAEncrypt.Encrypt("<RSAKeyValue><Modulus>yzD2IAHWgOX5LAtEFI/5XAYGfHV1kK5dYHwFcVCBIZlzyIDBni1BN/FAi+bsXpK2bF4B7hV2HTcGMx/p54mFxTds3aCT23fmtOBg3U8pv7HT7BCYOA0otFeZBBZ9Rt6/tev6W96lSYt/0x5+AN2Zz4r+ql40NRAwTGVCSIWHntM=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>", DESEncrypt.Decrypt(param.Password));

param.Password = HttpUtility.UrlEncode(encryptPassword);

var bodyData = $"grant\_type=password&username={param.UserName}&password={param.Password}";

var result = WebHelper.LoginRequestUtile($"{param.ApiUrl.TrimEnd('/')}/api/login", WebRequestMethods.Http.Post, bodyData, "",

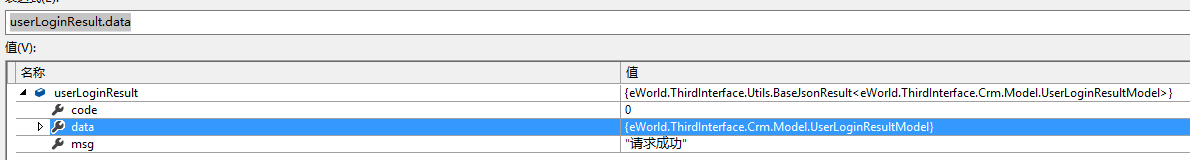
"text/plain");

return result;

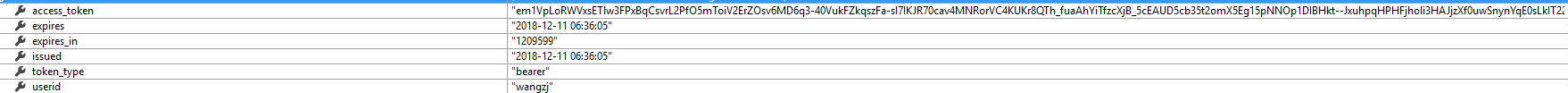
（2.4）

我在这里设置断点，获取到了接口的请求信息。

i：请求成功



II：data中的内容：



这就获取到access\_token了：

"em1VpLoRWVxsETlw3FPxBqCsvrL2PfO5mToiV2ErZOsv6MD6q3-40VukFZkqszFa-sI7IKJR70cav4MNRorVC4KUKr8QTh\_fuaAhYiTfzcXjB\_5cEAUD5cb35t2omX5Eg15pNNOp1DIBHkt--JxuhpqHPHFjholi3HAJjzXf0uwSnynYqE0sLkIT22q3OKZoJKsbJz5oUtCIqECkuTaLp65mYOxdzgvVdS6sKGgv\_fJijb2bjNBhF2xvtxX-MSWCt2AJo5Fo5yEik-6xhpa6xySyOAlKH8edSdnXJ9\_qOSXslqXcrhBB1il8Xl81dbddHKJFE9xkqUGJfWF4fXzmT0P-JcY"

其中token的生成：

Token = userLoginResult.data.token\_type + " " + userLoginResult.data.access\_token,

b. 生成SSOToken

将用户名，密码，还有用户的Guid，再次生成SSOToken

#region 生成SSOToken

//**TODO:SSOToken值是一个json（UserName，PassWord，CustomerGuid）加密后结果**

//**TODO:SSOToken加密的步骤1：先DES加密JSON；2.加密后取结果的前5位 3.移到末尾**

//**TODO:例如：1.加密前{"UserName":"BD26D4CAA62465F3","PassWord":"E52E542969005783DEE08CB531892FD2B98959E328BBCDAC","CustomerGuid":"5ea38e04-026f-4109-b26f-f01490ee1019"} 2.加密后：DB356A238A11EC1698FB71B5450A 3.取结果前5位移到末尾：A238A11EC1698FB71B5450ADB356**

SSOTokenModel ssoTokenModel = new SSOTokenModel()

{

CustomerGuid = doctorMember.CustomerGuid,

PassWord = param.Password,

UserName = param.UserId

};

string ssoTokenDES = DESEncrypt.Encrypt(JsonConvert.SerializeObject(ssoTokenModel));

if (ssoTokenDES.Length > 5)

{

ssoTokenDES = ssoTokenDES.Remove(0, 5) + ssoTokenDES.Substring(0, 5); //移除前5位字符然后把前五位拼接到末尾

}

所以接口的最终返回结果是：

{

"ResultInfo": {

"CustomerGuid": "cdc5f939-fc42-4297-a4d5-05788a4a31e0",

"UserId": "wangzj",

"CustomerName": "王专家",

"InstitutionUID": "94117694-8ecf-4792-9a86-3aff306b6d12",

"InstitutionCode": "15112",

"InstitutionName": "上海交通大学医学院附属瑞金医院",

"Token": "bearer em1VpLoRWVxsETlw3FPxBqCsvrL2PfO5mToiV2ErZOsv6MD6q3-40VukFZkqszFa-sI7IKJR70cav4MNRorVC4KUKr8QTh\_fuaAhYiTfzcXjB\_5cEAUD5cb35t2omX5Eg15pNNOp1DIBHkt--JxuhpqHPHFjholi3HAJjzXf0uwSnynYqE0sLkIT22q3OKZoJKsbJz5oUtCIqECkuTaLp65mYOxdzgvVdS6sKGgv\_fJijb2bjNBhF2xvtxX-MSWCt2AJo5Fo5yEik-6xhpa6xySyOAlKH8edSdnXJ9\_qOSXslqXcrhBB1il8Xl81dbddHKJFE9xkqUGJfWF4fXzmT0P-JcY",

"SSOToken": "A238A11EC166FB9BF1CD9827764AAEB9E138A7DDE5B505F5CBD327B22F22BFC7A3DD9D45CE6FA1AF5FA96EBBD4798E492D587E3A3BD3448F1B12652801169BA37326532A6AD58B3D57573F8B3C83196A7F7684B8849B3A934048DC7EBE90BF638847F05EF389B8C3023C1D7D25C9D5F3BB5D9AF6FAE74DC07C75B665C0AFED2BBEEAC017689DB356"

},

"ReturnInfo": {

"Code": 200,

"Msg": "登录成功"

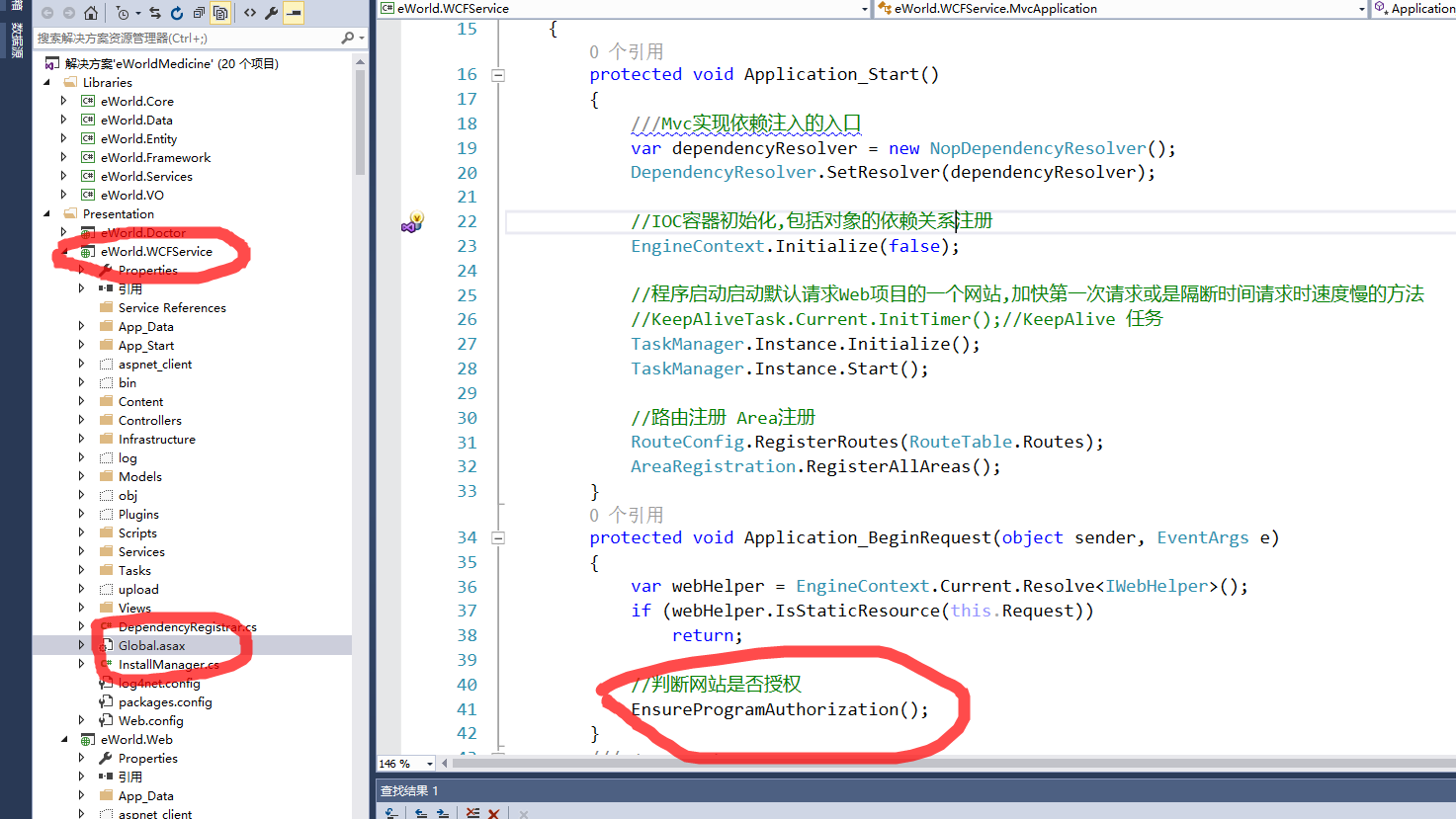
}

}

总结就是，将DES加密后的账号密码，用签名key重新DES加密后，发送给CRM，请求张军的接口，然后获取返回的结果集，再根据用户名+密码+CustomerGuid加密成SSOToken，再返回该全部结果集。

2.

上面获得的Token授权，我才发现，原来是给WCF用的。



3.

不对，Web Api中还是有的。

是通过在token中获取CurrentCustomerGuid来判断是否已经登陆的。否则就不给继续访问的权限。类似Web中的，在控制器上打上了必须验证登陆信息的标签，没有登陆的话，是不给访问的。

if (CurrentCustomerGuid == null)

{

result.code = ResultStatusCode.VALID\_TOKEN\_FAIL;

result.msg = "获取登录信息失败";

return result;

}

4.

Web api和Web中，获取用户信息，都用到了这个——Iidentity（这个应该跟权限验证，或者说授权Authorization有些关系）

/// <summary>

/// 用户信息

/// </summary>

public class CustomerIdentity : IIdentity

5.

先到这里了，只知道web api接口在这里可以获取登陆信息，内部实现还不知道。

if (CurrentCustomerGuid == null)

{

result.code = ResultStatusCode.VALID\_TOKEN\_FAIL;

result.msg = "获取登录信息失败";

return result;

}

6.

7.

8.

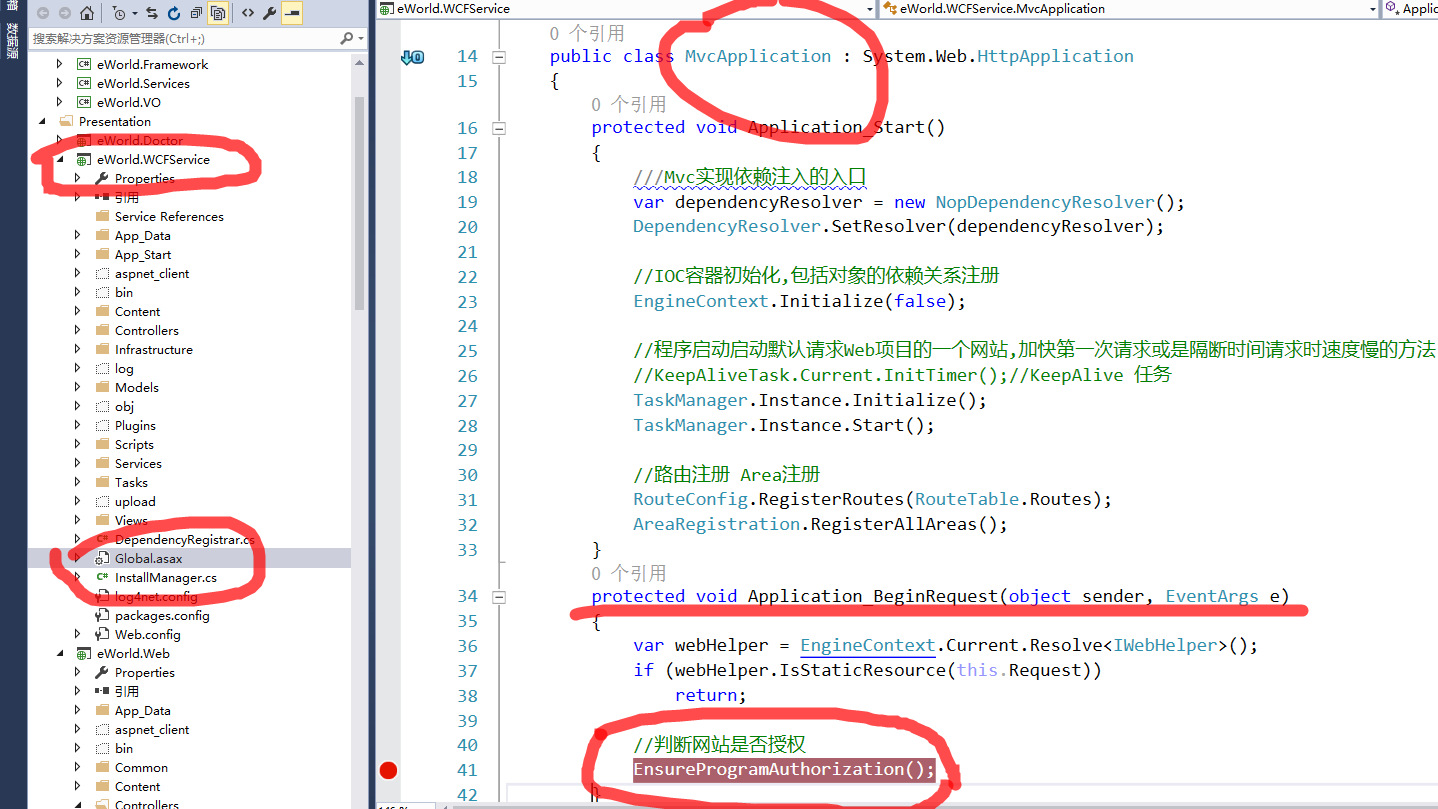
9.

调试Web接口的时候，恰好走到了流程：

请求过来，报文头里面的Cookie：

errormsg=; \_\_RequestVerificationToken=jBCMDKcAFsmIAr45c3b\_0KCqjAJSoW1J7dRyiFyxRicYV9Eo59Of-yDLXbOBmkOdtCbUQiAGkuJycRYYma5TojvPBIK8PiQgl8KMirQmop41; crm\_token=985A11C9469F72F07D16788AEAC323595179941B0DB8A8BC50C60E14C1E47E7BCAB1BEC3D146C3FE; eWorldToken=8F729D6F63E2A696A4E2FF62B97B42C1DDDBBF3746C1B80775B949596594D86D8B6985C07AEEC4AFB15C2C2A01CB86279C14A8555CA729E3072631F5CDB99C6C8C886471DB5E59962FB0FFCD97D36DF725306A6749660922FA125687DBE50E0BC73DE7BAF14CF024C1A6620FFB9D996024B695364B6371B4; Nop.customer=a123d84b-d27b-44ca-affa-667a99d56eba

1.



（1）请求控制器的方法，必然走到了每次请求都要经历的公开事件：

整体处在MvcApplication中。

public class MvcApplication : System.Web.HttpApplication

跳到了是否授权：

protected void Application\_BeginRequest(object sender, EventArgs e)

{

var webHelper = EngineContext.Current.Resolve<IWebHelper>();

if (webHelper.IsStaticResource(this.Request))

return;

//判断网站是否授权

EnsureProgramAuthorization();

}

（2）授权函数的方法体：

/// <summary>

/// 是否授权 如果未授权则跳到授权页面

/// </summary>

protected void EnsureProgramAuthorization()

{

var webHelper = EngineContext.Current.Resolve<IWebHelper>();

string authorazationUrl = string.Format("{0}authorize", webHelper.GetWebSiteLocation());



authorazationUrl = GetLocation(authorazationUrl);



if (!webHelper.IsStaticResource(this.Request) &&

!AuthorizationSettingsHelper.ProgramAuthorzation() &&

!webHelper.GetThisPageUrl(false).StartsWith(authorazationUrl, StringComparison.InvariantCultureIgnoreCase))

this.Response.Redirect(authorazationUrl);

if (AuthorizationSettingsHelper.ProgramAuthorzation())

EnsureDatabaseIsInstalled();

是否配置了数据库

}

是否配置数据库：

/// <summary>

/// 是否配置数据库

/// </summary>

protected void EnsureDatabaseIsInstalled()

{

var webHelper = EngineContext.Current.Resolve<IWebHelper>();

string installUrl = string.Format("{0}install", webHelper.GetWebSiteLocation());



installUrl = GetLocation(installUrl);

if (!webHelper.IsStaticResource(this.Request) &&

!DatabaseConnectionManager.DatabaseIsInstalled() &&

!webHelper.GetThisPageUrl(false).StartsWith(installUrl, StringComparison.InvariantCultureIgnoreCase))

{

this.Response.Redirect(installUrl);

}

}

（3）授权验证之后跳到了自定义的防火墙控制器的重写方法BeginExecute中，从字面意思就是控制器开始之前去执行：

public class FirewallMvcController : BaseController

{

/// <summary>

///

/// </summary>

/// <param name="requestContext"></param>

/// <param name="callback"></param>

/// <param name="state"></param>

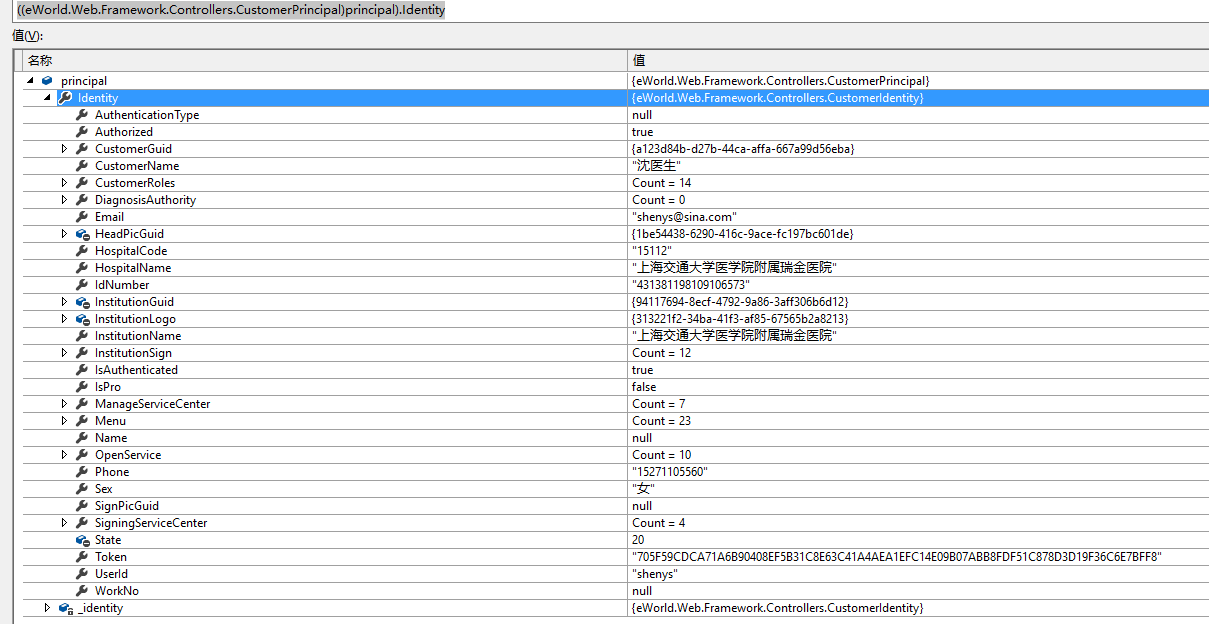
/// <returns></returns>

protected override IAsyncResult BeginExecute(RequestContext requestContext, AsyncCallback callback, object state)

{

var principal = GetCustomerPrincipal(requestContext);

执行了后面的（4）方法后，已经获取了值。



if (principal != null)

{

requestContext.HttpContext.User = principal;

User：包含当前 HTTP 请求的安全信息的对象。

将Principal的值放到上下文的User当中。后面的CurrentCustomer信息就可以从里面直接拿了

ViewData["CustomerIdentity"] = principal.Identity;

principal.Identity:

/// 与当前用户关联的 <see cref="T:System.Security.Principal.IIdentity"/> 对象。

赋值给了ViewData["CustomerIdentity"]，理论上前端就可以直接拿了

}

return base.BeginExecute(requestContext, callback, state);

}

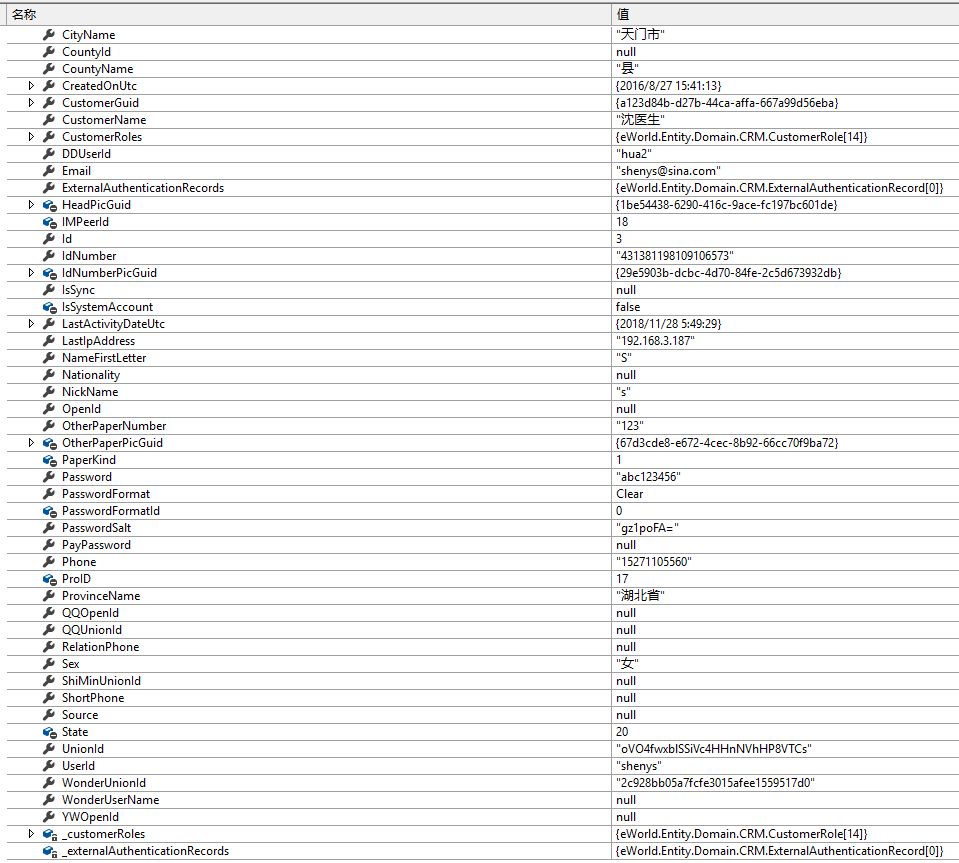
（4）执行上面打了红的方法：

private IPrincipal GetCustomerPrincipal(RequestContext requestContext)

{

var customer = EngineContext.Current.Resolve<IWorkContext>().CurrentCustomer;

这里就拿到了用户信息：



var customerCookie = requestContext.HttpContext.Request.Cookies["eWorldToken"];

获取了eWorldToken的Cookie。报文头里面是有的。

if (customerCookie?.Value != null)

{

if (customer == null)

{

return null;

}

string customerToken = DESEncrypt.Decrypt(customerCookie.Value);

解码前：8F729D6F63E2A696A4E2FF62B97B42C1DDDBBF3746C1B80775B949596594D86D8B6985C07AEEC4AFB15C2C2A01CB86279C14A8555CA729E3072631F5CDB99C6C8C886471DB5E59962FB0FFCD97D36DF725306A6749660922FA125687DBE50E0BC73DE7BAF14CF024C1A6620FFB9D996024B695364B6371B4

解码后："a123d84b-d27b-44ca-affa-667a99d56eba^D5F835516E57CBAEC962DE59352D0E591298116954929AACC6D6E8299677CC1FFC928FBD5689A92D"

if (!customerToken.Contains("^"))

{

return null;

}

var key = "medicine\_" + customerToken.Split(new[] { "^" }, StringSplitOptions.RemoveEmptyEntries)[0];

获取key："medicine\_a123d84b-d27b-44ca-affa-667a99d56eba"

if (!StackExchangeRedisHelper.Exists(key))

判断在缓存中是否存在该key的缓存数据（这里应该是redis中查看是否存在这个缓存，有的话就放行，没有的话就返回null）

{

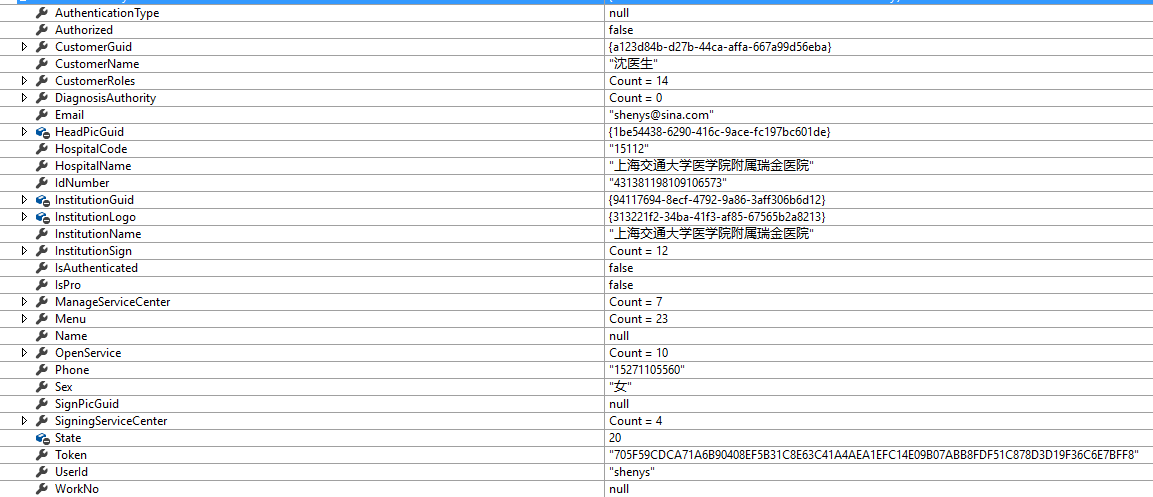
return null;

}

var customerIdentity = JsonConvert.DeserializeObject<CustomerIdentity>(StackExchangeRedisHelper.Get(key).ToString());

根据这个key从Redis中获取value，然后反序列化成CustomerIdentity类型。(感觉可以直接获取，然后判断是否为null，这样只从redis中获取一次)

获取的customerIdentity值：



customerIdentity.Authorized = true;

是否授权置为true，已经授权。

return new CustomerPrincipal(customerIdentity);

子类赋值给父类：public class CustomerPrincipal : IPrincipal

}

return null;

}

2.

（1）第二次测试，发现在调试开始的时候，是判断连接这些

/// <summary>

///　获取一个值，表示当前连接是否安全

/// </summary>

/// <returns>true - secured, false - not secured</returns>

public virtual bool IsCurrentConnectionSecured()

{

bool useSsl = false;//默认是不安全

if (\_httpContext != null && \_httpContext.Request != null)

{

//通过上下文的请求获取该连接是否使用安全套协议

useSsl = \_httpContext.Request.IsSecureConnection;

//当你的主机使用了负载均衡的服务器上，那么Request.IsSecureConnection永远不会得到设置为true　需要取消下面这句注释

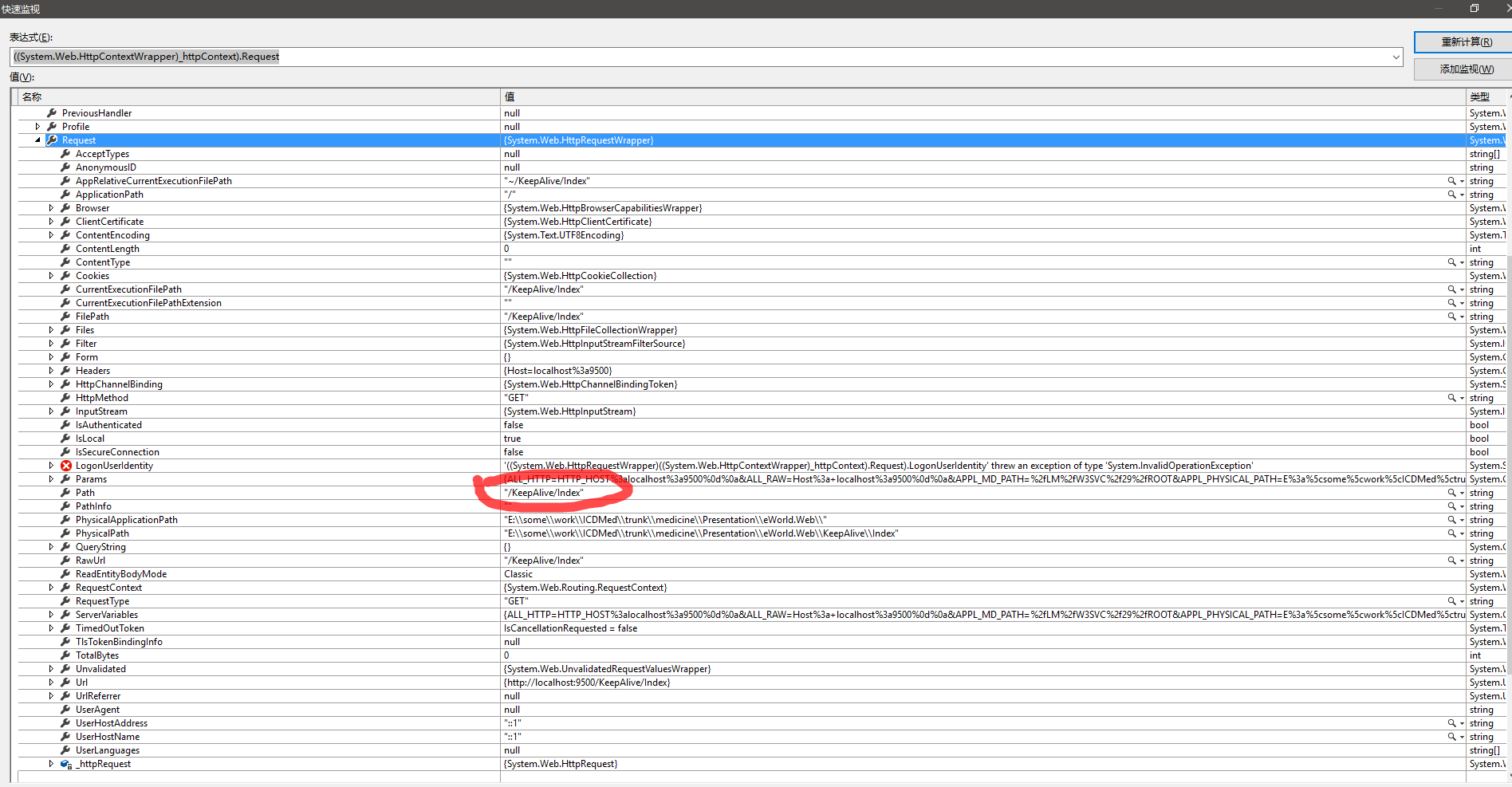
//useSSL = \_httpContext.Request.ServerVariables["HTTP\_CLUSTER\_HTTPS"] == "on" ? true : false;

}

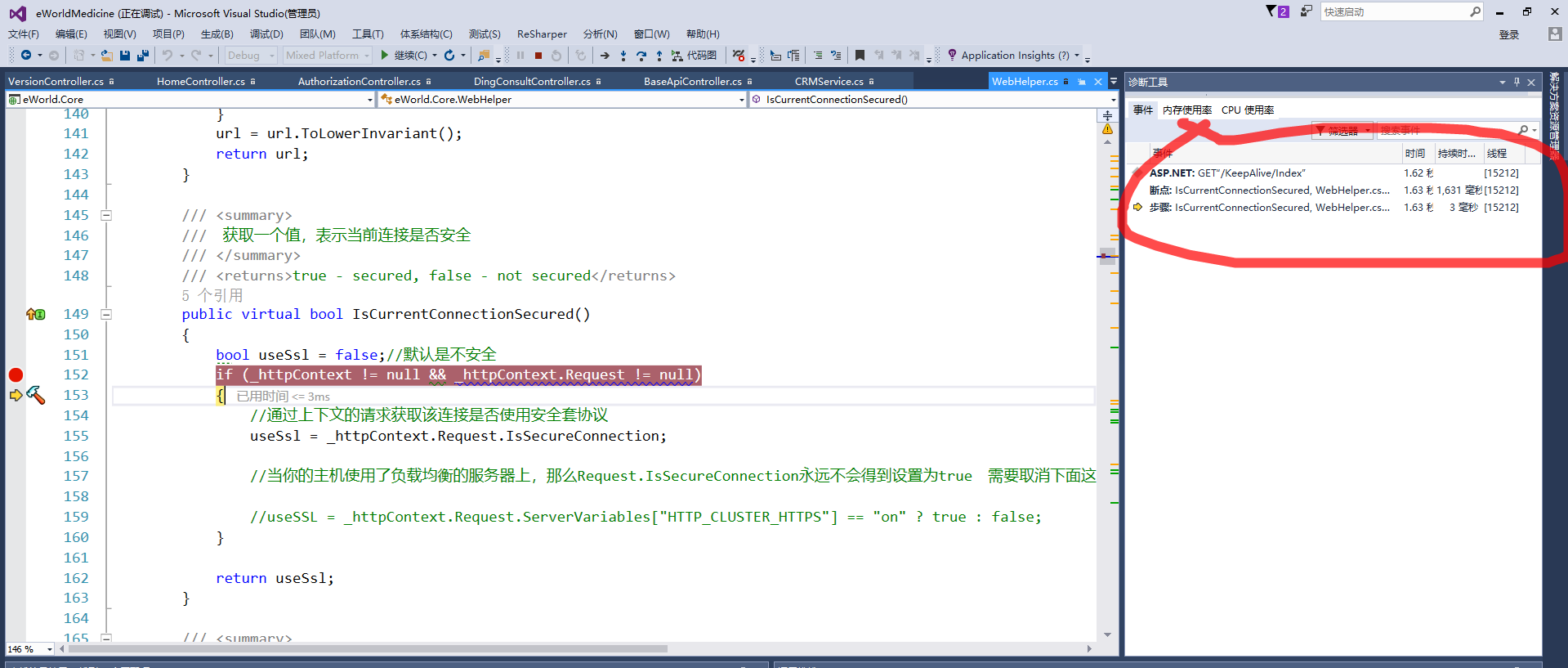
return useSsl;

}

这里查看\_httpContext里面的内容，发现\_httpContext.Request有以下，包含keepalive的东西。



此时右边的KeepAlive还没有在轮询。



（2）走到了获取商店地址。

/// <summary>

///　获取商店地址

/// </summary>

/// <returns>商店地址</returns>

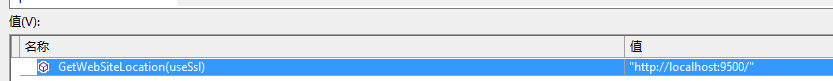
public virtual string GetWebSiteLocation()

{

bool useSsl = IsCurrentConnectionSecured();

return GetWebSiteLocation(useSsl);

获取的地址，就是Web的地址



}

（3）接着进入到了Web的公开事件中：

namespace eWorld.Web

{

// Note: 指令使用IIS6中或IIS7经典模式,

// visit http://go.microsoft.com/?LinkId=9394801

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_BeginRequest(object sender, EventArgs e)

{

//忽略静态资源

var webHelper = EngineContext.Current.Resolve<IWebHelper>();

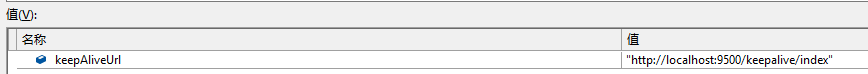
if (webHelper.IsStaticResource(this.Request))

return;

//维持页面请求(我们忽略它,以防止创建一个客户客户记录)

string keepAliveUrl = $"{webHelper.GetWebSiteLocation()}keepalive/index";

在这里启动了keepalive



if (webHelper.GetThisPageUrl(false).StartsWith(keepAliveUrl, StringComparison.InvariantCultureIgnoreCase))

return;

//EnsureProgramAuthorization();

if (DataSettingsHelper.DatabaseIsInstalled())

{

//防止伪造

var securitySetting = FrameworkHelper.GetSecuritySettings();

if (securitySetting.IsEnableCSRF)

{

string localAddr = Request.ServerVariables.Get("Local\_Addr");

string urlReferrer = (this.Request.UrlReferrer == null ? null : this.Request.UrlReferrer.Authority);

string[] csrfUrls = new string[] { };

if (!string.IsNullOrWhiteSpace(securitySetting.CSRFUrl))

{

csrfUrls = securitySetting.CSRFUrl.Split(new[] { '|' }, StringSplitOptions.RemoveEmptyEntries);

}

string host = Request.Headers["host"];

if (string.IsNullOrWhiteSpace(host))

{

host = "";

}

if (!string.IsNullOrWhiteSpace(urlReferrer))

{

urlReferrer = urlReferrer.ToLower();

if (localAddr != "::1")//如果不是本地调试localhost的话走判断,是的话默认调试放行

{

if (!csrfUrls.Any(s => urlReferrer.StartsWith(s.ToLower())))

{

if (securitySetting.IsLogCSRF)

{

var logger = EngineContext.Current.Resolve<ILogger>();

string message = "检测到CSRF攻击! UrlReferrer=" + urlReferrer + " 拒绝时间：" + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

logger.InsertLog(LogLevel.Warning, "检测到CSRF攻击", message, null);

}

//this.Response.Redirect("/ErrorPage.htm");

Response.StatusCode = 403;

return;

}

}

}

if (localAddr != "::1")

{

host = host.ToLower();

if (!csrfUrls.Any(s => host.StartsWith(s.ToLower())))

{

if (securitySetting.IsLogCSRF)

{

var logger = EngineContext.Current.Resolve<ILogger>();

string message = "检测到Host头攻击! Host=" + host + " 拒绝时间：" + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

logger.InsertLog(LogLevel.Warning, "检测到Host头攻击", message, null);

}

Response.StatusCode = 403;

//this.Response.Redirect("/ErrorPage.htm");

return;

}

}

}

}

if (CanPerformProfilingAction())

{

MiniProfiler.Start();

}

}

（4）获取页面名称，也即是keepalive的地址：

/// <summary>

/// 获取页面名称

/// </summary>

/// <param name="includeQueryString">表示是否包含查询字符串</param>

/// <returns>页面名称</returns>

public virtual string GetThisPageUrl(bool includeQueryString)

{

bool useSsl = IsCurrentConnectionSecured();//获取是否使用安全套协议

return GetThisPageUrl(includeQueryString, useSsl);

}

但一直死循环，走不出来了，整个流程。

2.1：撤销这些断点。

（1）

第二次测试，明明是访问web的控制器，但默认的，都是跳出来wcf的加载。



（2）

wcf之后，就是授权的认证。

protected void Application\_BeginRequest(object sender, EventArgs e)

{

var webHelper = EngineContext.Current.Resolve<IWebHelper>();

if (webHelper.IsStaticResource(this.Request))

return;

//判断网站是否授权

EnsureProgramAuthorization();

}

3.

4.

接口这里：

1.

原来接口这边一点校验，或者过滤器都没有的。基本是裸奔。

我把获取当前登陆用户的CurrentCustomerGuid注销掉，自己发一个王专家的guid进去，照样走通的，所以是校验这里，完全靠的CurrentCustomerGuid这一个！



返回结果让就是正常的：

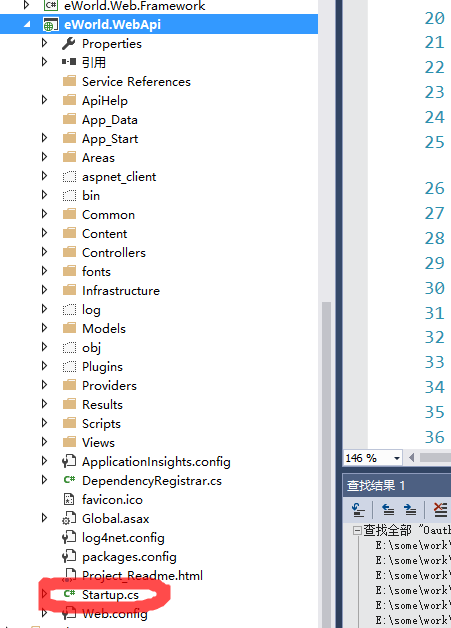


2.

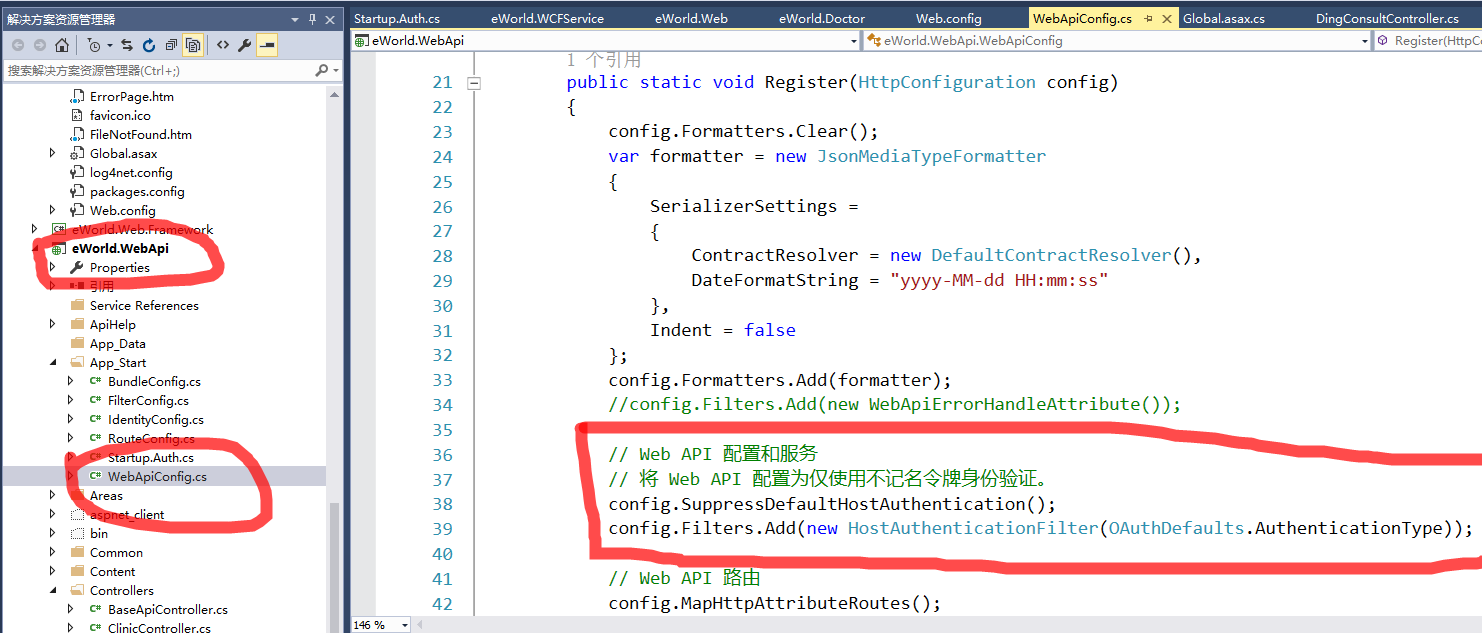
在api中继续找寻授权的踪影：

网上查找：使用ASP.NET Identity 实现WebAPI接口的Oauth身份验证（在收藏夹中）

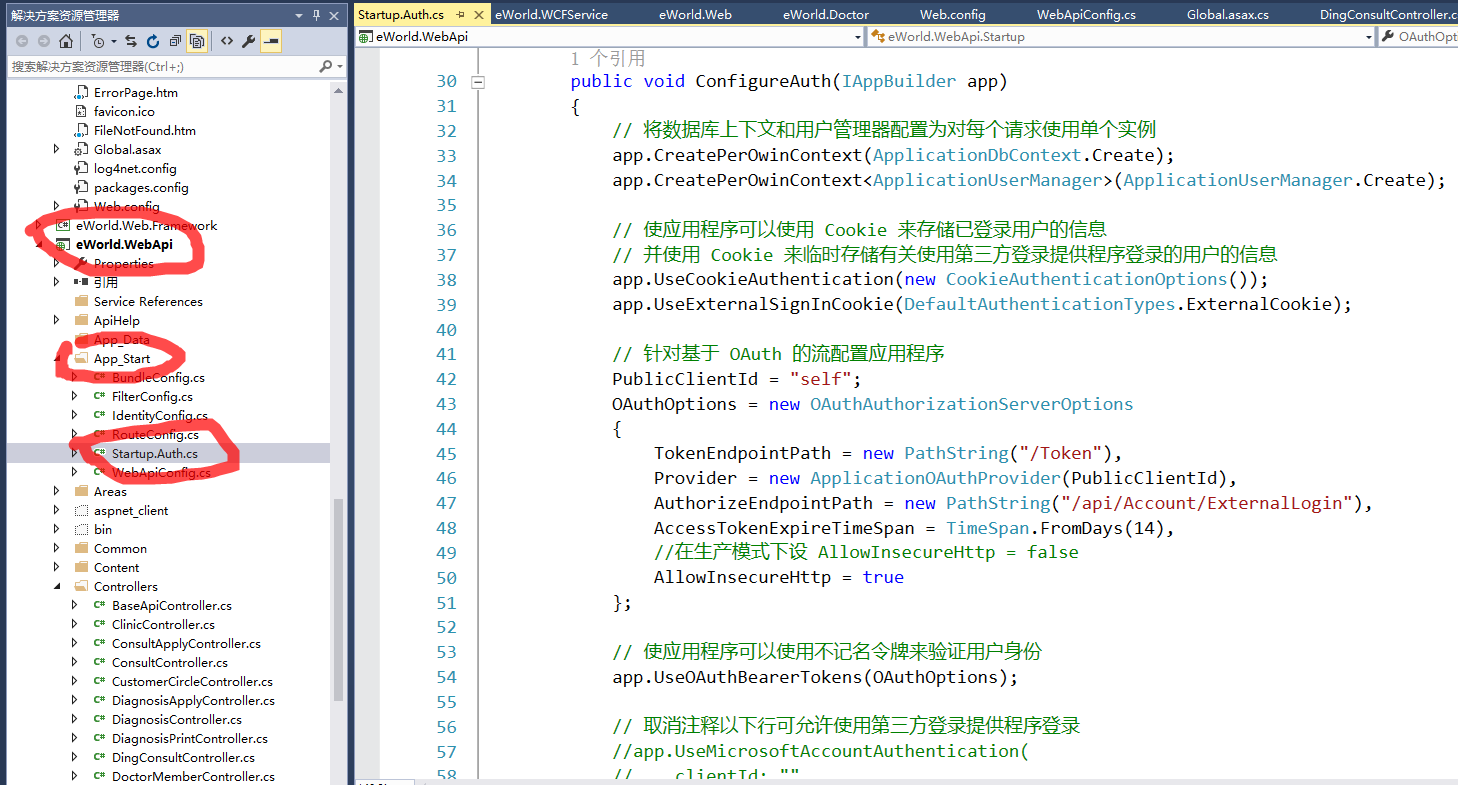
（1）



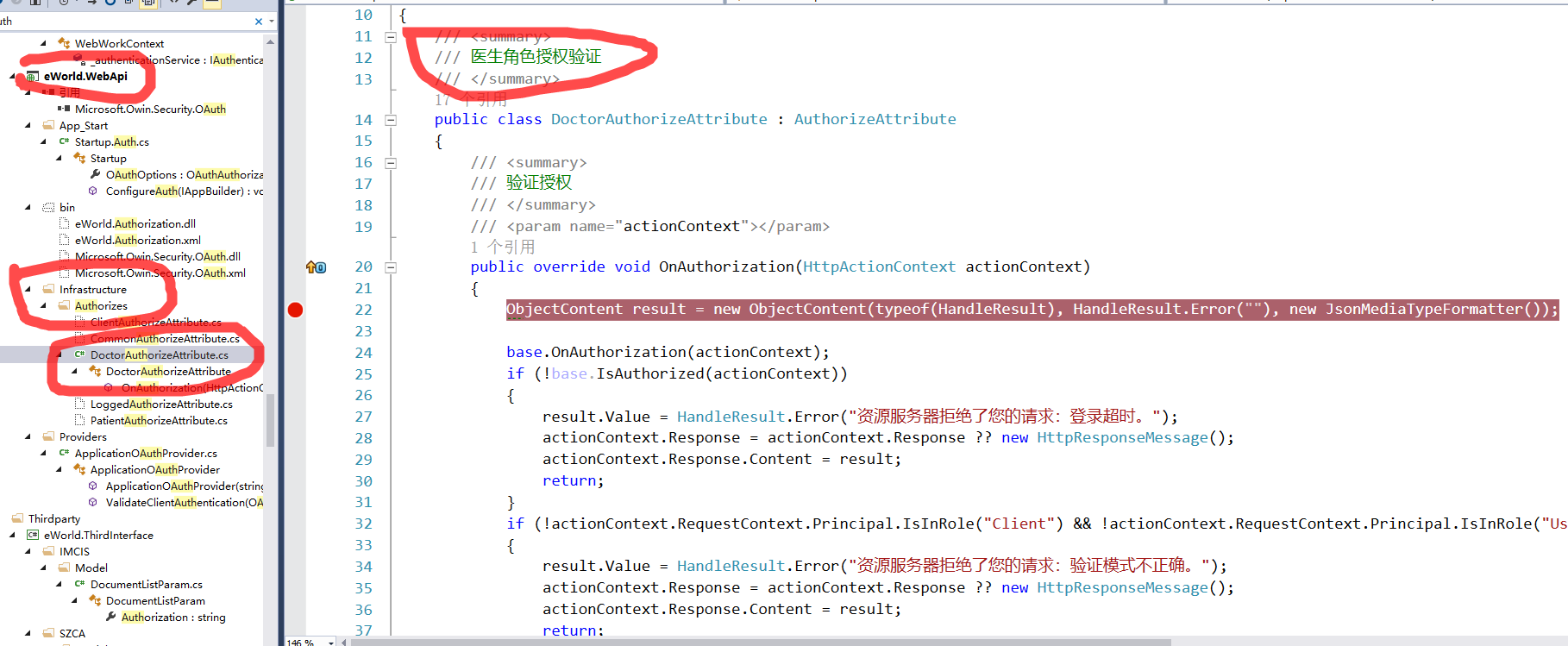
（2）App\_Start的WebApiConfig中。



（3）App\_Start的Startup.Auth中：



（4）有一个医生角色授权验证的标签，我自己的接口没有去用过。



3.

4.

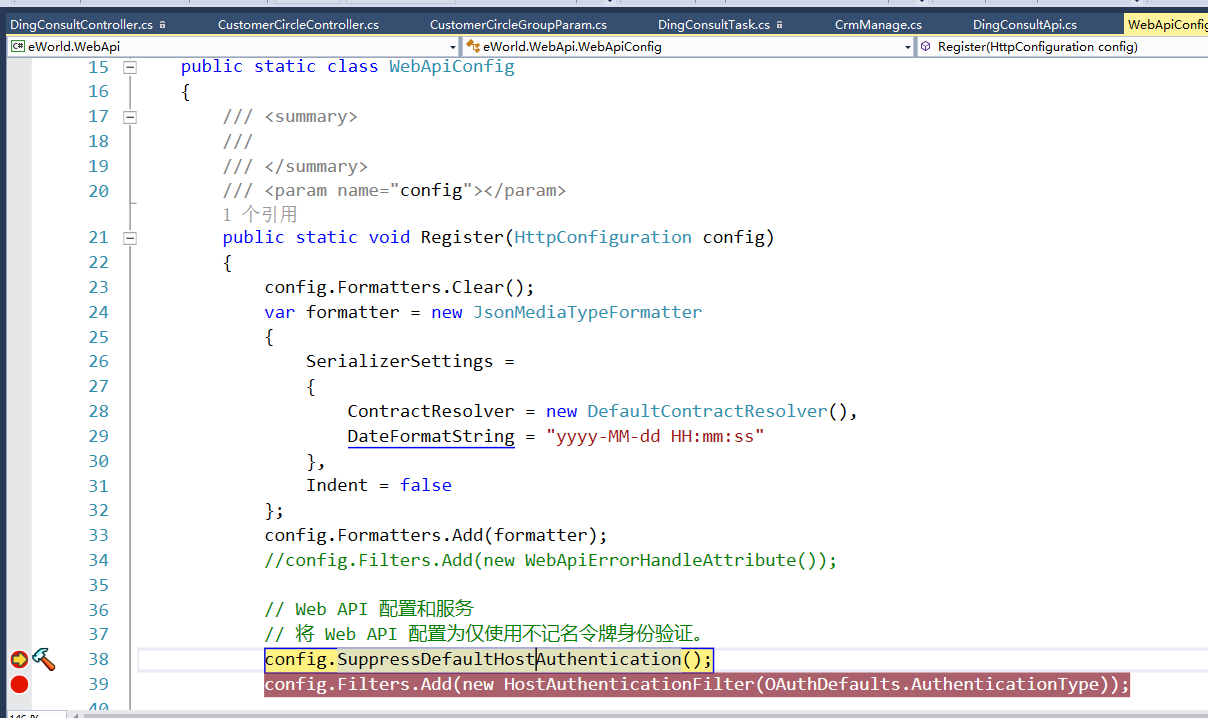
再走一次接口：

1.

重启iis，代表第一次请求过来，

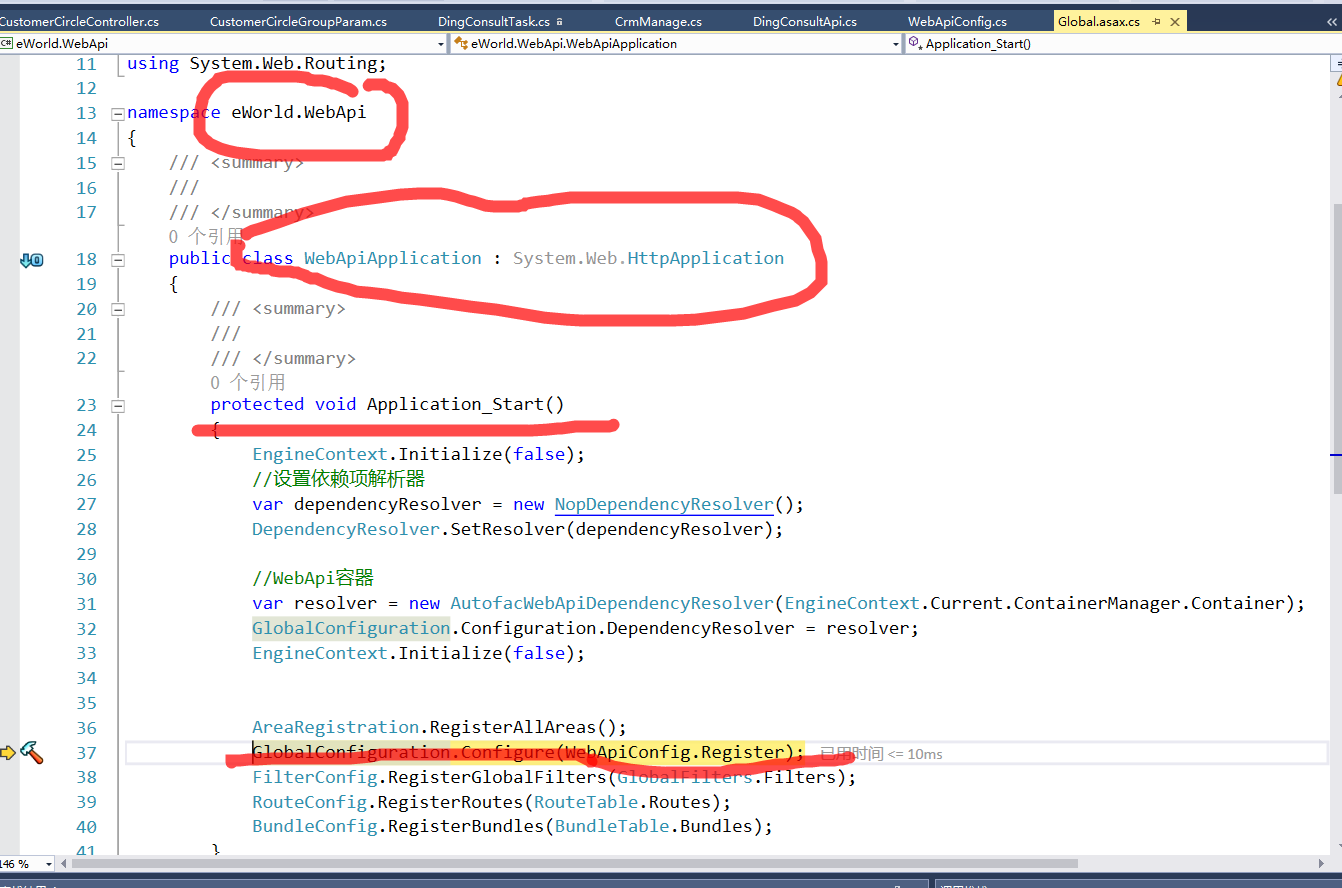
（1）

跳到了WebApiConfig中的auth配置。



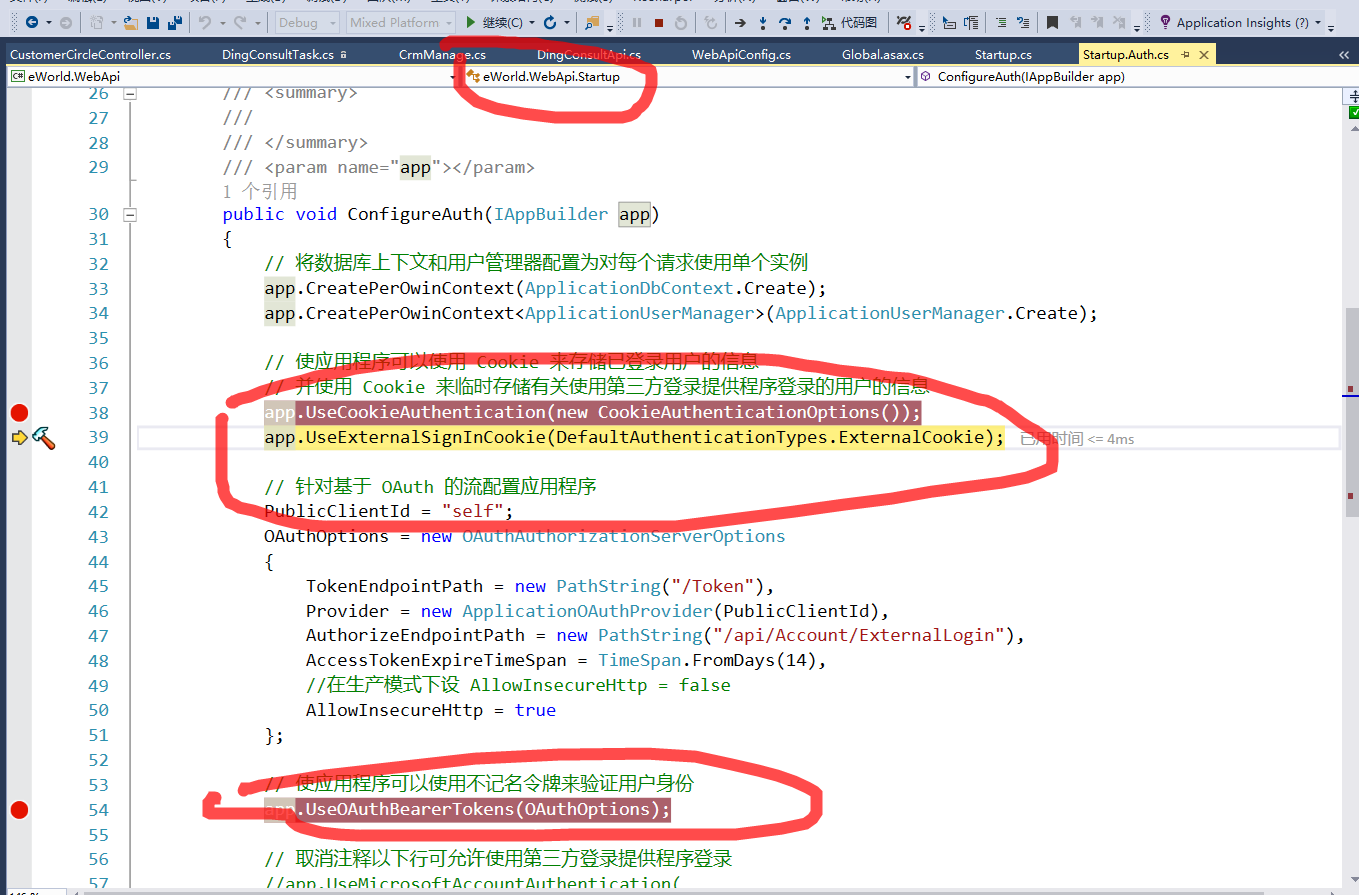
（2）

跳到了Application\_Start()中，生命周期只执行一次。应该是将auto正式地加入到了整个流程当中。



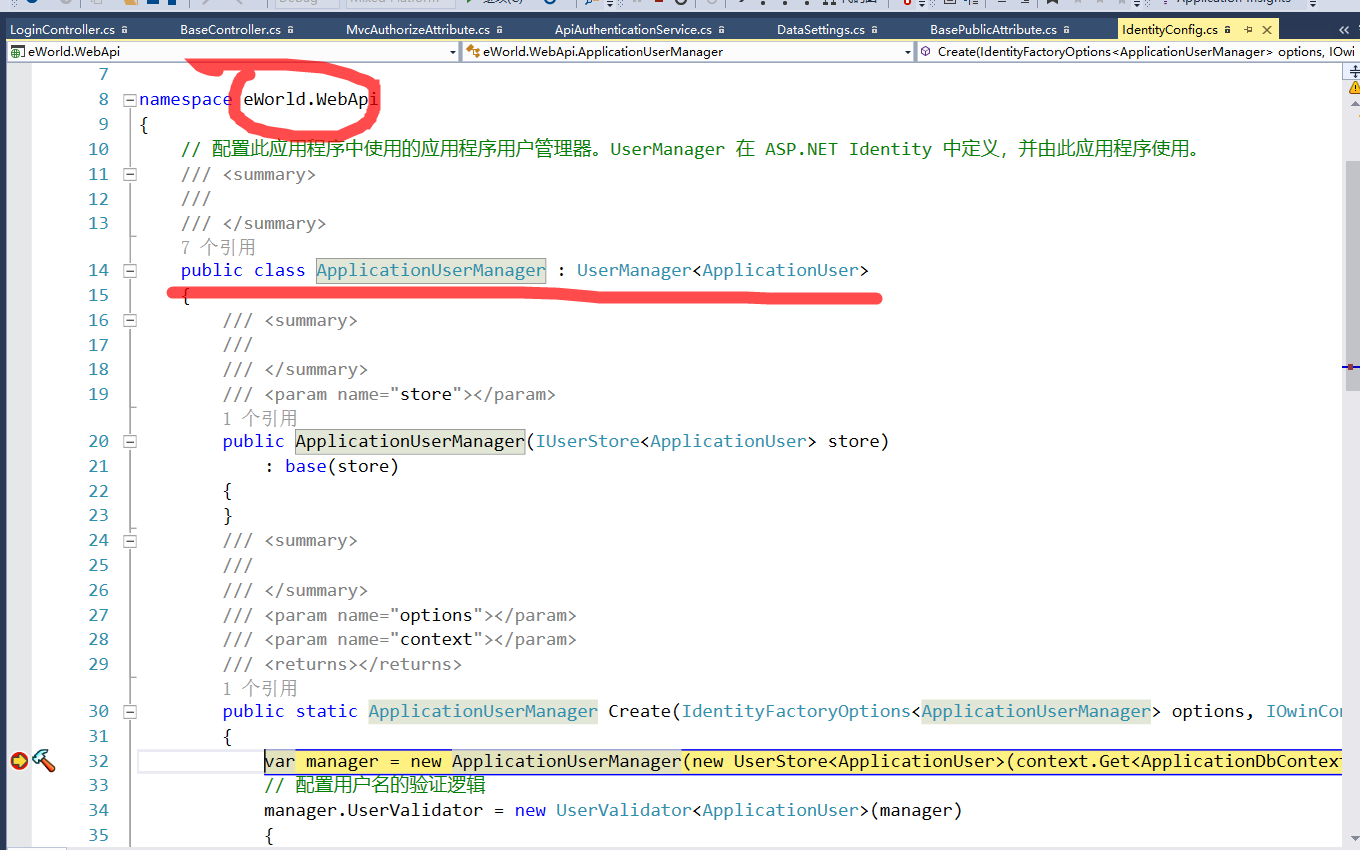
（3）

跳到了Startup中。也就是引用了auth后多了的类。



（4）

跳到了WebApi中Identity的定义中。



public static ApplicationUserManager Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)

{

var manager = new ApplicationUserManager(new UserStore<ApplicationUser>(context.Get<ApplicationDbContext>()));

// 配置用户名的验证逻辑

manager.UserValidator = new UserValidator<ApplicationUser>(manager)

{

AllowOnlyAlphanumericUserNames = false,

RequireUniqueEmail = true

};

// 配置密码的验证逻辑

manager.PasswordValidator = new PasswordValidator

{

RequiredLength = 6,

RequireNonLetterOrDigit = true,

RequireDigit = true,

RequireLowercase = true,

RequireUppercase = true,

};

var dataProtectionProvider = options.DataProtectionProvider;

if (dataProtectionProvider != null)

{

manager.UserTokenProvider = new DataProtectorTokenProvider<ApplicationUser>(dataProtectionProvider.Create("ASP.NET Identity"));

}

return manager;

}

（5）

接下来就走到了接口



2.

（1）

上面是第一次请求，第二次请求就跳过Config，App\_Start()，直接跳到上面的第四步，然后第五步，还是不知道User是在上面时候赋值的。

也许是在CRM那边吧，但感觉不可能啊，先放着吧。

（2）

（3）

3.

4.