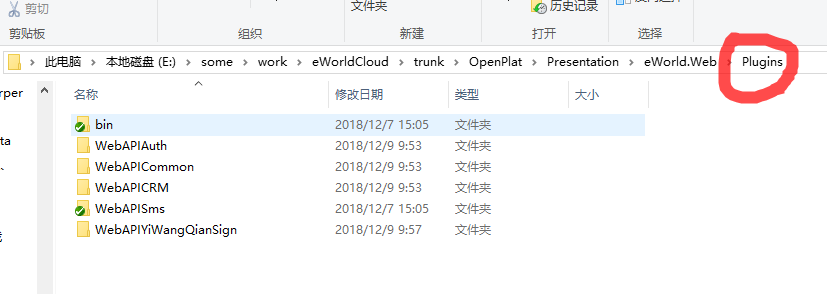
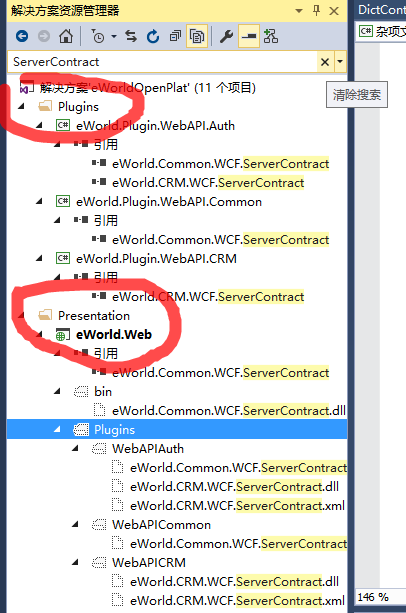
探索一些

1.



彪哥这里的接口，Plugins，似乎最后还都是包在web里面的，虽然在vs中看着是并列一般，只有也解释了为什么这些web api应用只是类库，而不是可以发布的web api应用。

不对，仔细看了下，只是最终编译后，生成在了这唯一的一个web应用当中。



2.

似乎是动态引用的，在变的。刚刚弹出来，让我重新加载。说明crm那边修改了



3.

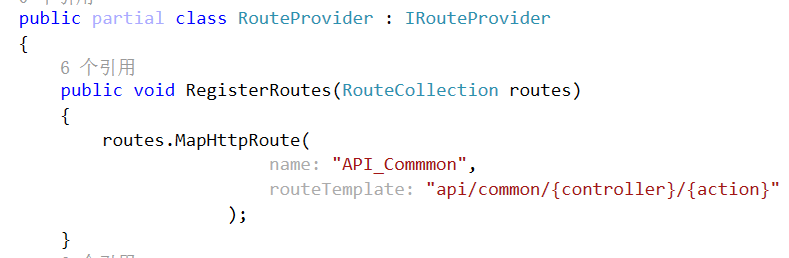
可参照网上找的autofac api注入：

api的注入：

api的实现：

builder.RegisterApiControllers(typeFinder.GetAssemblies().ToArray());

各个接口的地址：



4.

访问获取一级科室的接口

<http://localhost:7000/api/common/Dict/Get_Plat_SubjectFirst>

（0.1）

public ActionResult Index()

{

var model = new AuthorizeModel();

model.MachineCode = AuthorizationCode.getMNum();

return View(model);

}

（1）

namespace eWorld.Web.Framework.Mvc

{

/// <summary>

/// Base nopCommerce model

/// </summary>

public partial class BaseNopModel

{

public BaseNopModel()

{

this.CustomProperties = new Dictionary<string, object>();



}

public virtual void BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext)

{

}

/// <summary>

/// 使用这个属性来存储任何自定义值的模型。

/// </summary>

public Dictionary<string, object> CustomProperties { get; set; }

}

（2）

namespace eWorld.Web.Controllers

{

public class AuthorizationController : Controller

{

private readonly IInstallationLocalizationService \_locService;

public AuthorizationController(IInstallationLocalizationService locService)

{

this.\_locService = locService;

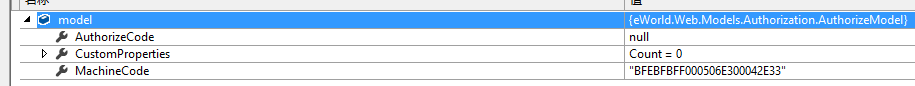
}

public ActionResult Index()

{

var model = new AuthorizeModel();

model.MachineCode = AuthorizationCode.getMNum();



return View(model);

}

这里的getMNum()方法寻找：

/// <summary>

/// 生成机器码

/// </summary>

/// <returns></returns>

public static string getMNum()

{

string strNum = getCpu() + GetDiskVolumeSerialNumber();//获得24位Cpu和硬盘序列号

string strMNum = strNum.Substring(0, 24);//从生成的字符串中取出前24个字符做为机器码

return strMNum;

}

（3）

namespace eWorld.Web.Framework

{

public class NopValidatorFactory : AttributedValidatorFactory

{

//private readonly InstanceCache \_cache = new InstanceCache();

public override IValidator GetValidator(Type type)

{

查找Type：



if (type != null)

{

var attribute = (ValidatorAttribute)Attribute.GetCustomAttribute(type, typeof(ValidatorAttribute));

结果为null



if ((attribute != null) && (attribute.ValidatorType != null))

{

//验证可以依赖于某些客户的特殊设置（如工作语言）

//这就是为什么我们不缓存验证器

//var instance = \_cache.GetOrCreateInstance(attribute.ValidatorType,

// x => EngineContext.Current.ContainerManager.ResolveUnregistered(x));

var instance = EngineContext.Current.ContainerManager.ResolveUnregistered(attribute.ValidatorType);

return instance as IValidator;

}

}

return null;

}

}

}

原始接口：

namespace FluentValidation

{

/// <summary>

/// Gets validators for a particular type.

///

/// </summary>

public interface IValidatorFactory

{

/// <summary>

/// Gets the validator for the specified type.

///

/// </summary>

IValidator<T> GetValidator<T>();

/// <summary>

/// Gets the validator for the specified type.

///

/// </summary>

IValidator GetValidator(Type type);

}

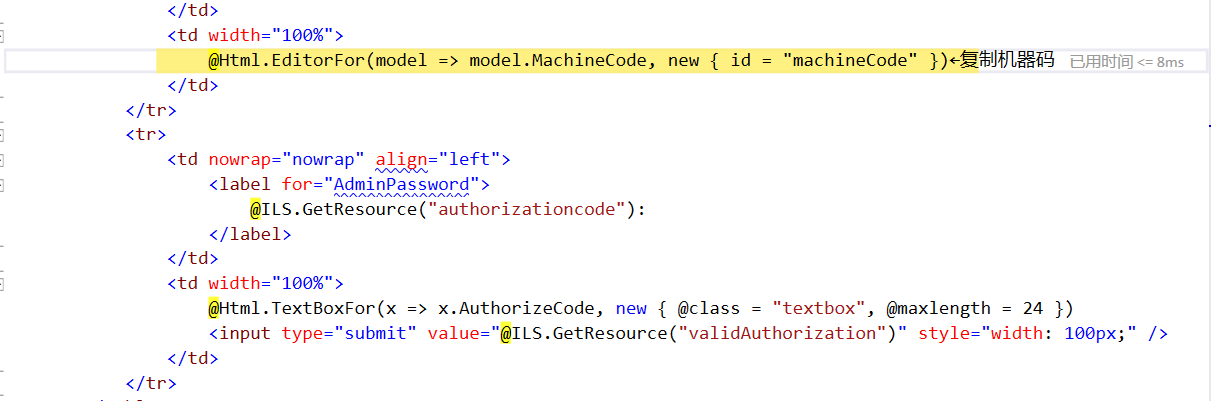
}

（4）



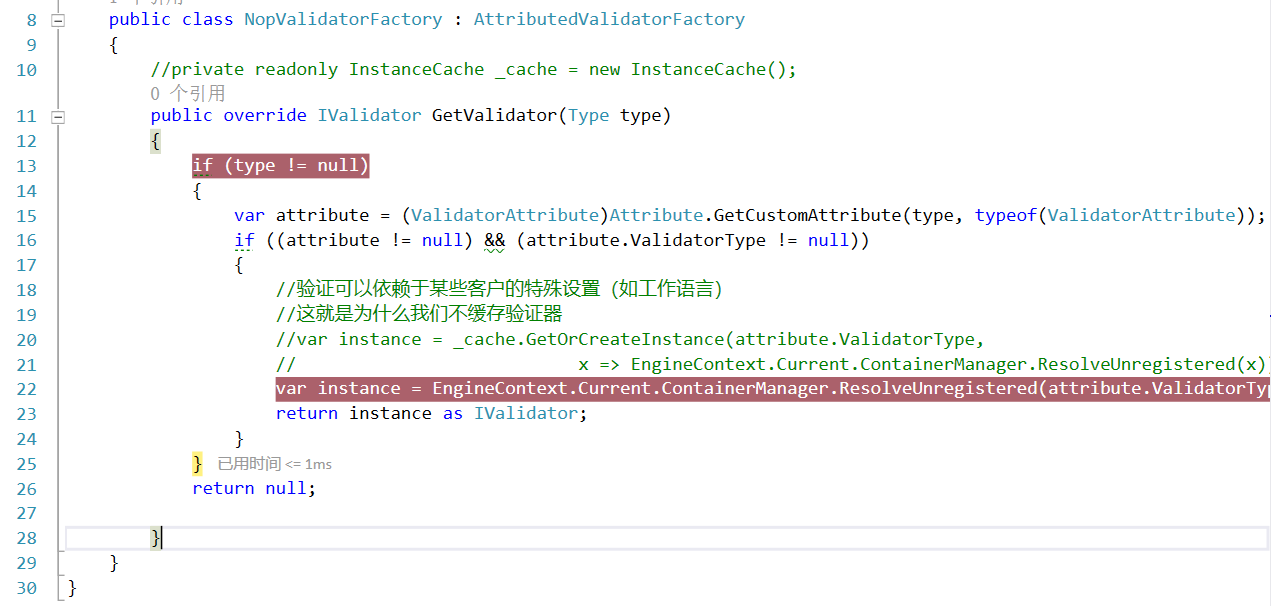
（5）

转跳授权的Index页面：



（6）

又继续跳回来。



（7）

继续跳到这个页面：

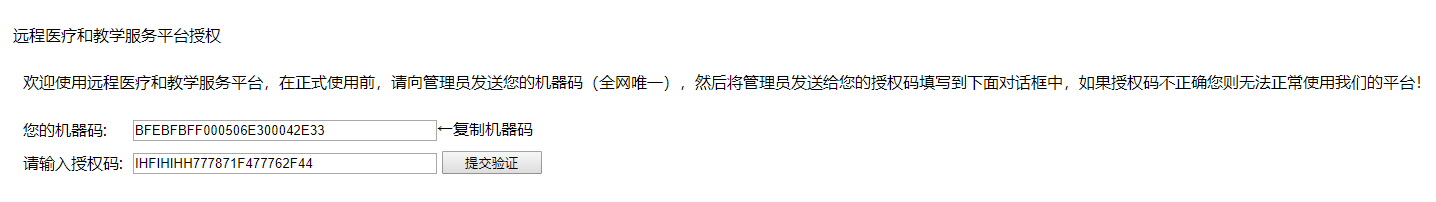


（8）

最后跳回授权的首页



5.



获取了授权码：

IHFIHIHH777871F477762F44

6.

在action中返回的代码：

return this.LowerCaseJson(data);

查找（也是类似jsonAction的重写）：

（1）

namespace eWorld.Web.Framework.Mvc

{

public static class MvcControllerExtension

{

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

{

return new LowerCaseJsonResult(data);

}

}

}

（2）

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);

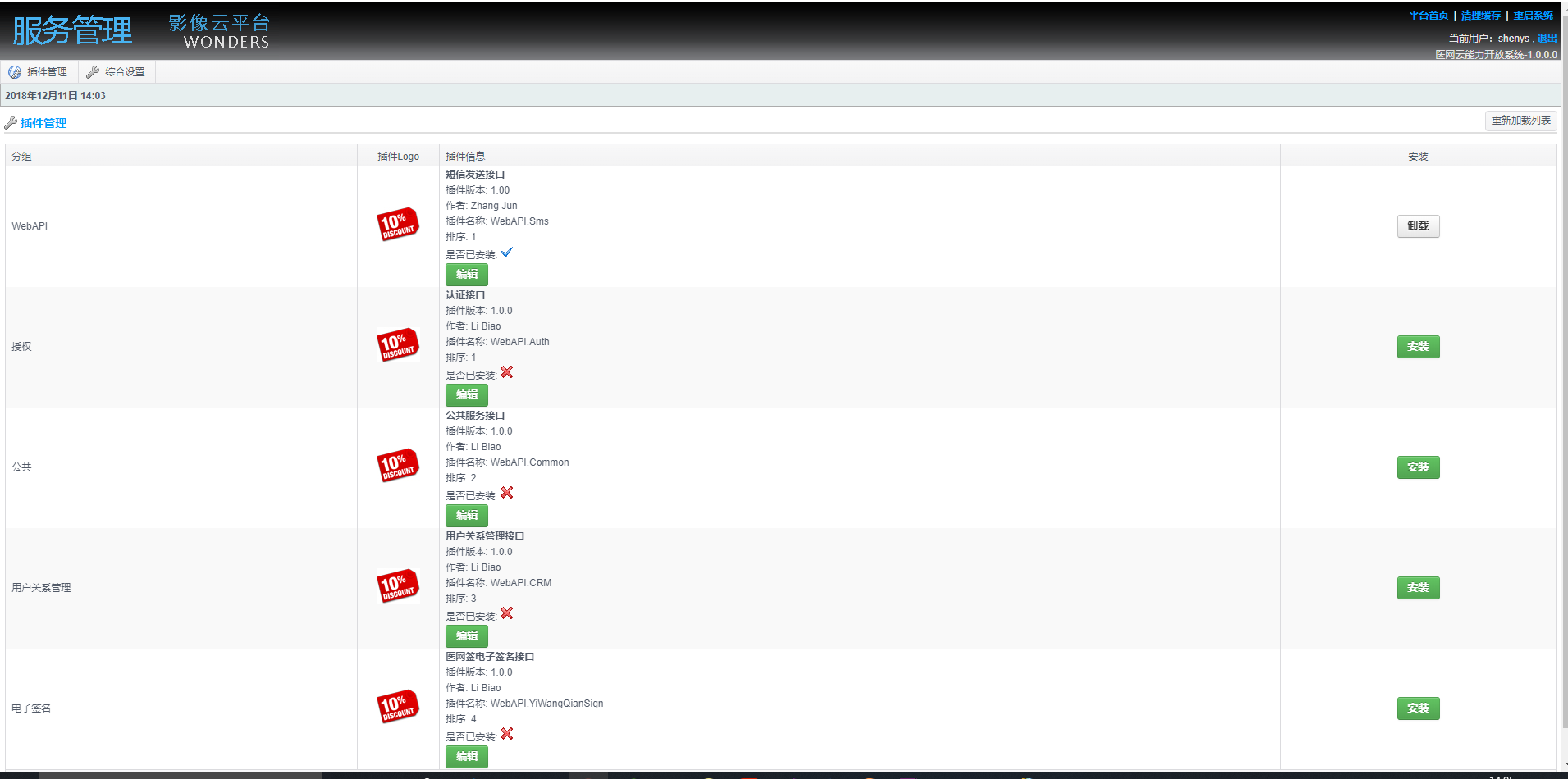
}

}

}

7.

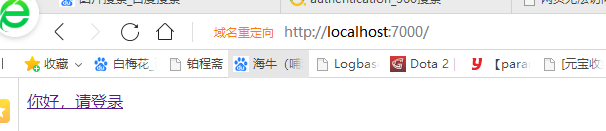
彪哥让我去这里下载插件。



8.

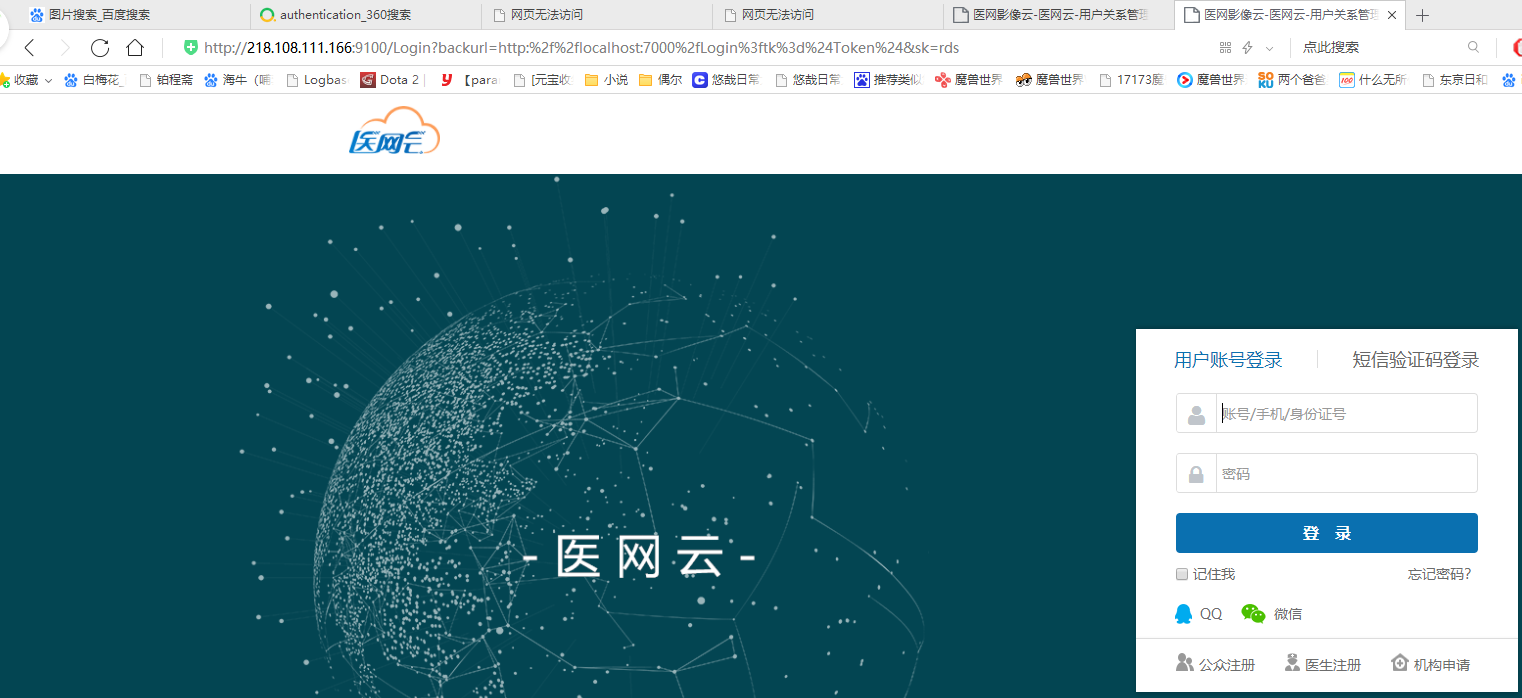
（1）

换成360访问7000，先是让登陆



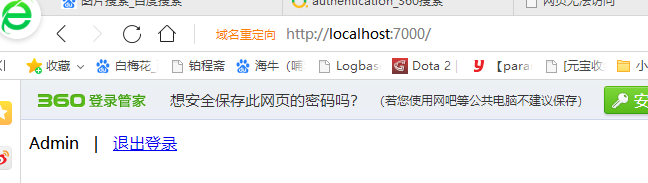
（2）

点击登陆之后，依旧是跳到张军的crm登陆界面



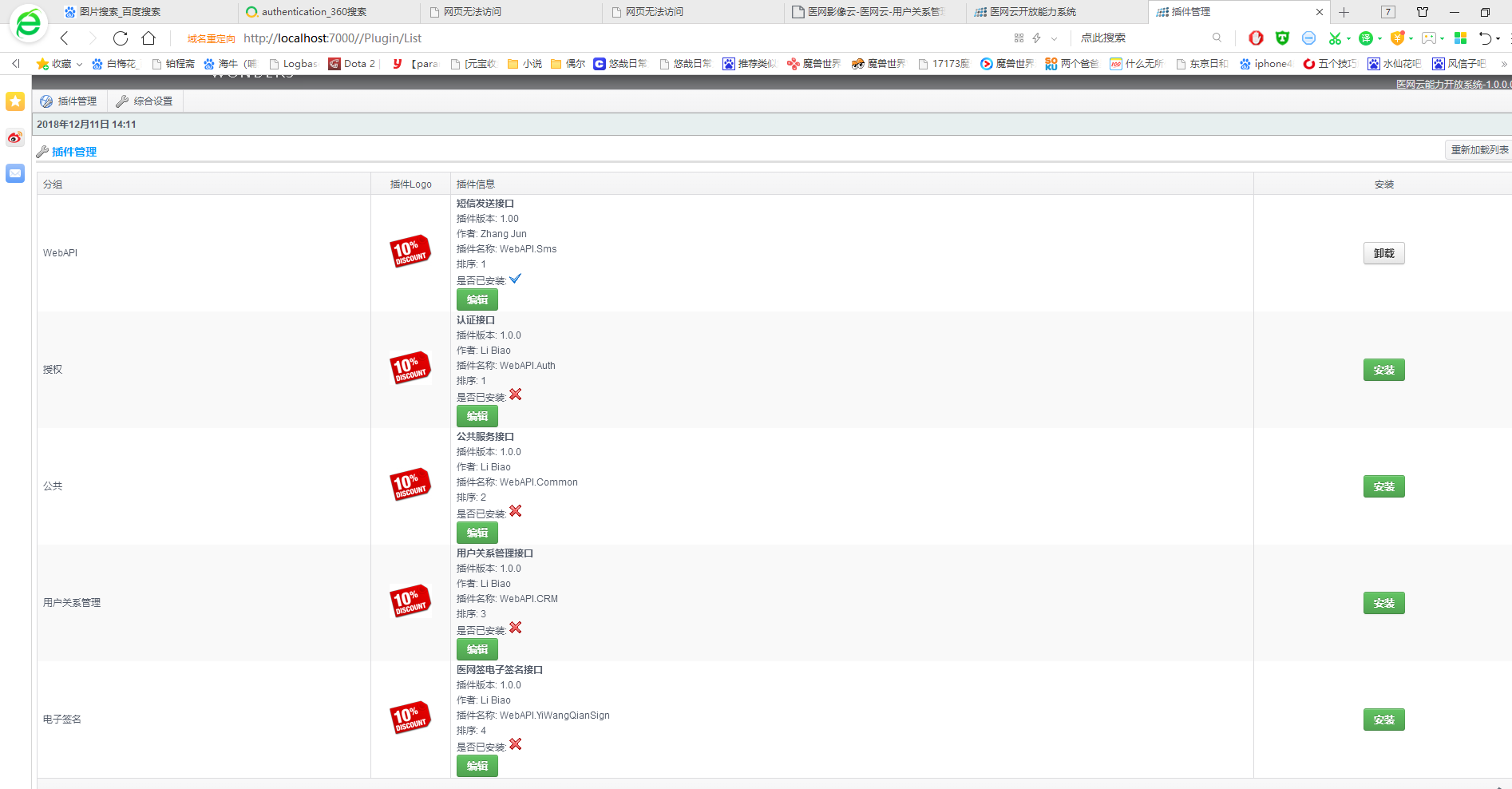
（3）

登陆了管理员后转跳回来



（4）

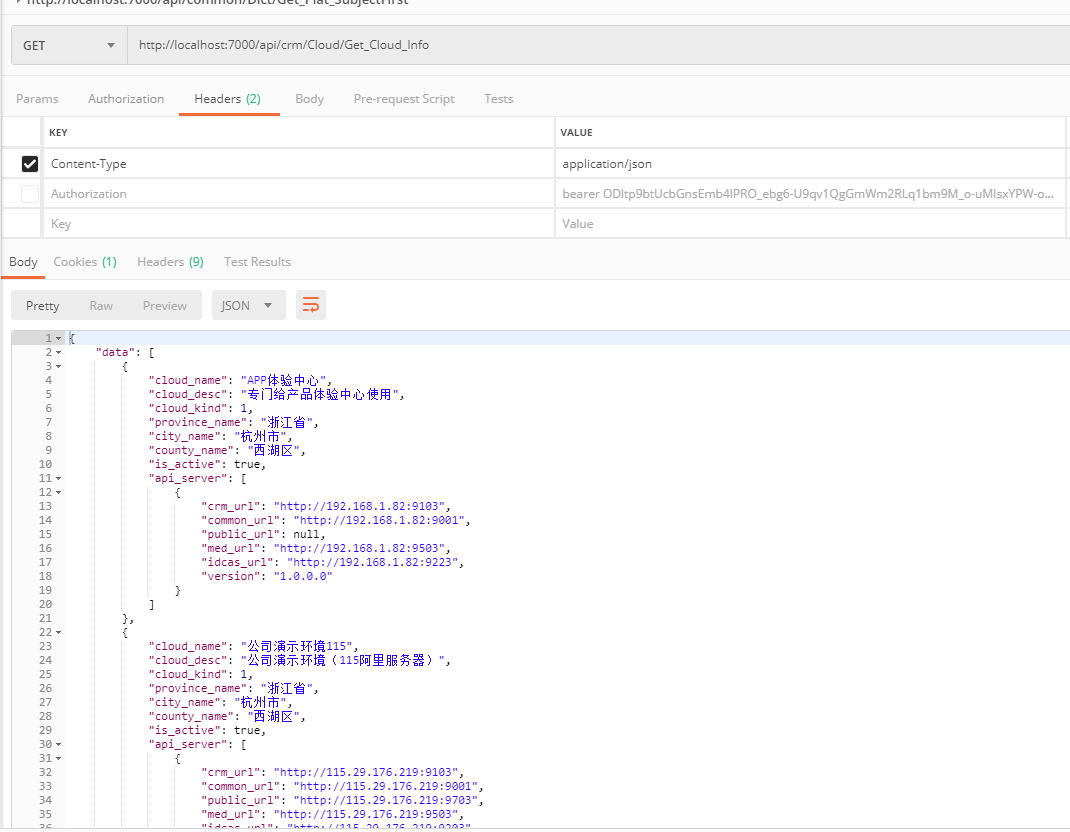
接下来进入插件页面





（5）

插件下载好了之后，接口就通了。神奇。



（6）

/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

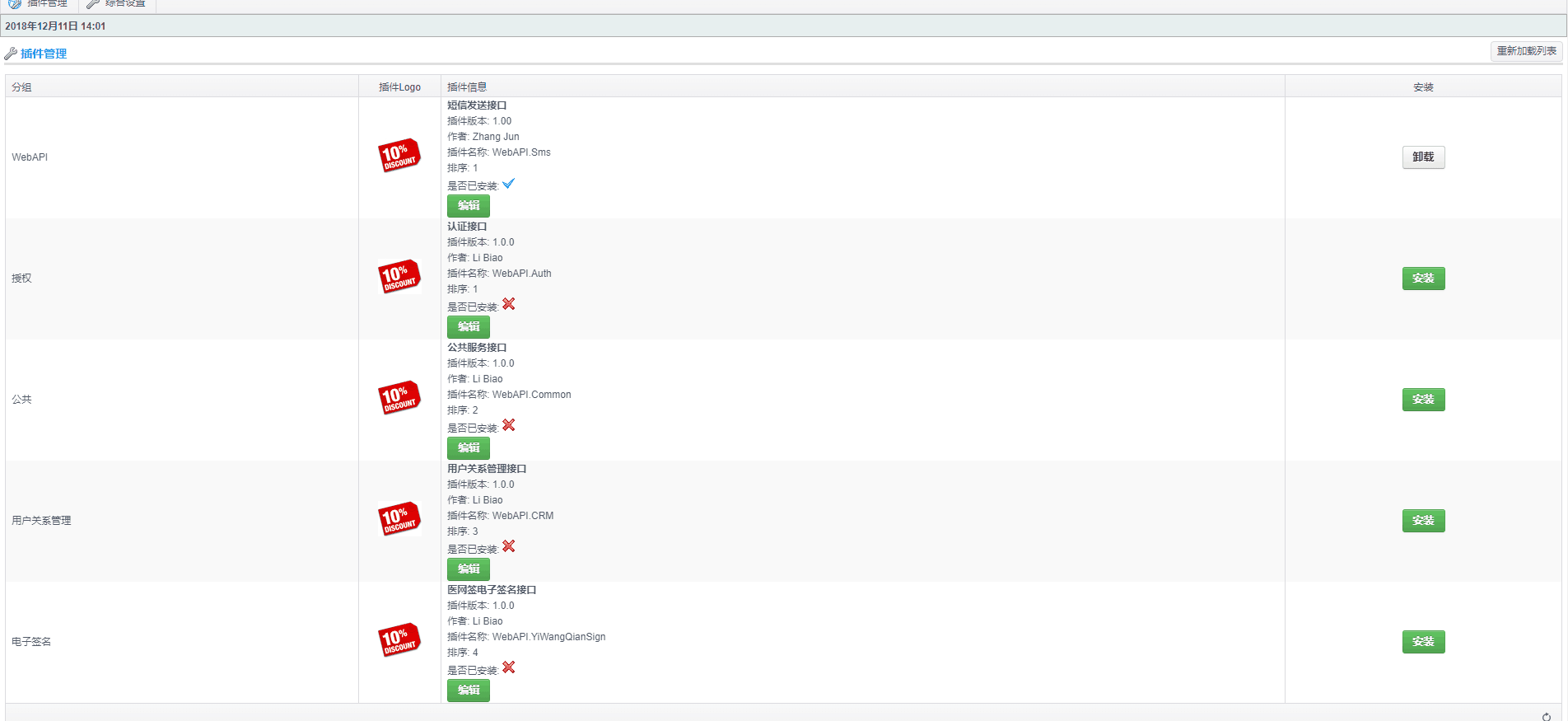
tomtaw

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



都要装？

浙江-蜗牛（彪哥） 2018/12/11 14:01:43

用哪个装哪个

（7）

搜索这个。

ssoProvider.IsLogin

（8）

（9）

9.

（1）

重新调试Home/Index路径

跳到方法：

public ActionResult Index()

{

return View();

}

（2）

跳到前台视图：

@using eWorld.Sso

@using eWorld.Core.Infrastructure

@{

Layout = null;

var ssoProvider = EngineContext.Current.Resolve<ISsoProvider>();

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>医网云开放能力系统</title>

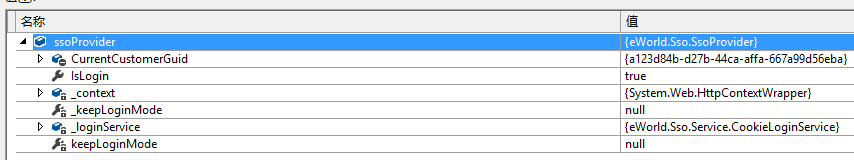
</head>

<body>

<div>

@if (ssoProvider.IsLogin)

ssoProvider的值



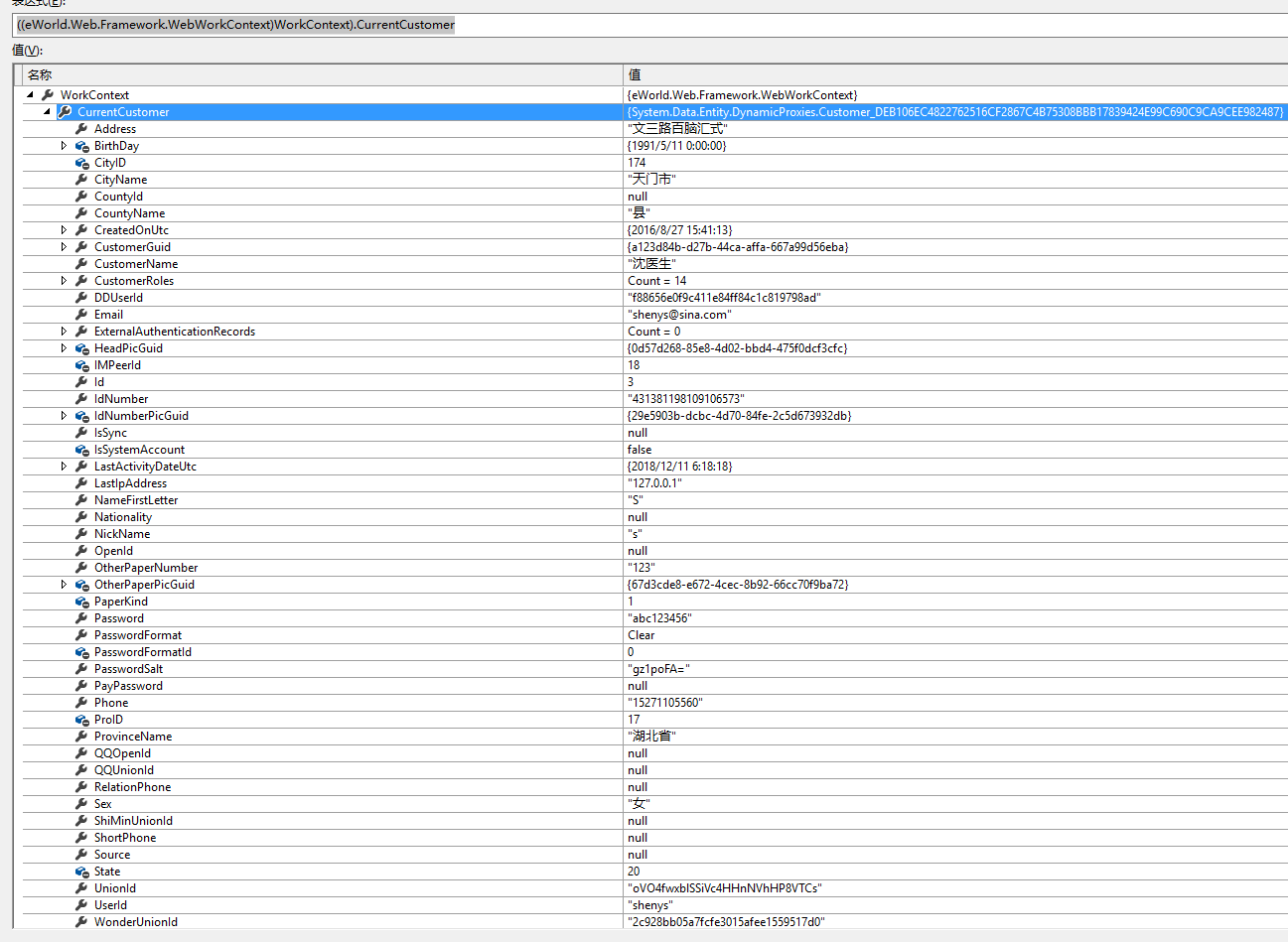
{

<div class="x3 text-right icon\_head" style="line-height:30px;">

<span class="margin-small-right head\_img">

<img src="~/Content/img/head.png" class="radius-circle" alt="" width="28" height="28">@WorkContext.CurrentCustomer.UserId

通过这个代码就能拿到当前登陆用户的信息



其中WorkContext的具体实现：

namespace eWorld.Core

{

/// <summary>

/// Work context

/// </summary>

public interface IWorkContext

{

/// <summary>

/// Gets or sets the current customer

/// </summary>

Customer CurrentCustomer { get; set; }

继续查找注入：

//work context

builder.RegisterType<WebWorkContext>().As<IWorkContext>().InstancePerHttpRequest();

继续查找WebWorkContext：

namespace eWorld.Web.Framework

{

/// <summary>

/// 工作上下文的Web应用程序

/// </summary>

public partial class WebWorkContext : IWorkContext

{

#region Const

private const string CustomerCookieName = "Nop.customer";

#endregion

视图余下的代码：

</span>&nbsp;&nbsp;<span>|</span>&nbsp;&nbsp;

<span class="margin-small-left Logout"><a href="/Logout">退出登录</a></span>

</div>

}

else

{

<div class="x3 text-right" style="line-height:30px;">

<span class="margin-left">

<a href="/Login">你好，请登录</a>

</span>

</div>

}

</div>

</body>

</html>

（3）

（4）

（5）

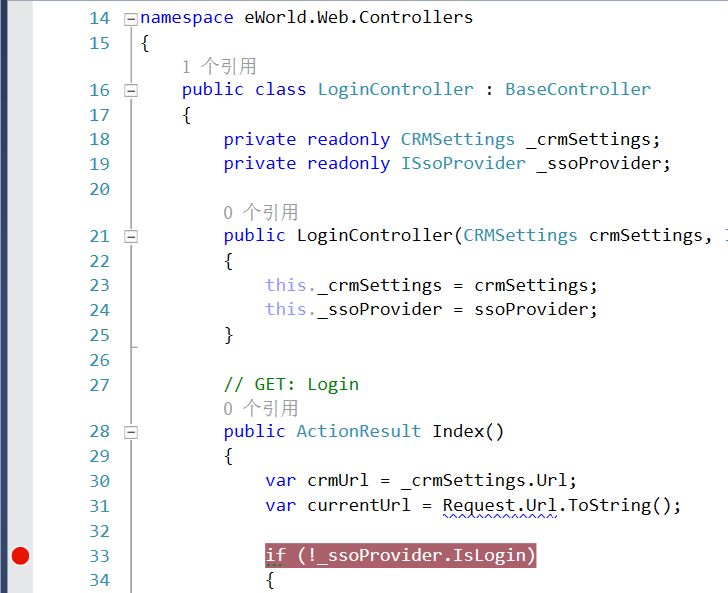
（6）

10.

登陆Login/Index

在获取UserId那里设置了断点后：

目标是进入到这里的CA中。



（1）

跳到UserId

namespace eWorld.Entity.Domain.CRM

{

/// <summary>

/// 用户实体，平台用户基本信息表

/// </summary>

public partial class Customer : BaseEntity

{

private ICollection<ExternalAuthenticationRecord> \_externalAuthenticationRecords;

private ICollection<CustomerRole> \_customerRoles;

/// <summary>

/// 构造函数

/// </summary>

public Customer()

{

this.CustomerGuid = Guid.NewGuid();

this.PasswordFormat = PasswordFormat.Clear;

}



拿到了登陆用户的值



（2）

跳到了



namespace eWorld.Services.Customers

{

/// <summary>

/// Customer service

/// </summary>

public partial class CustomerService : ICustomerService

{

public virtual Customer GetCustomerByGuid(Guid customerGuid)

{

if (customerGuid == Guid.Empty)

return null;

var query = from c in \_customerRepository.Table

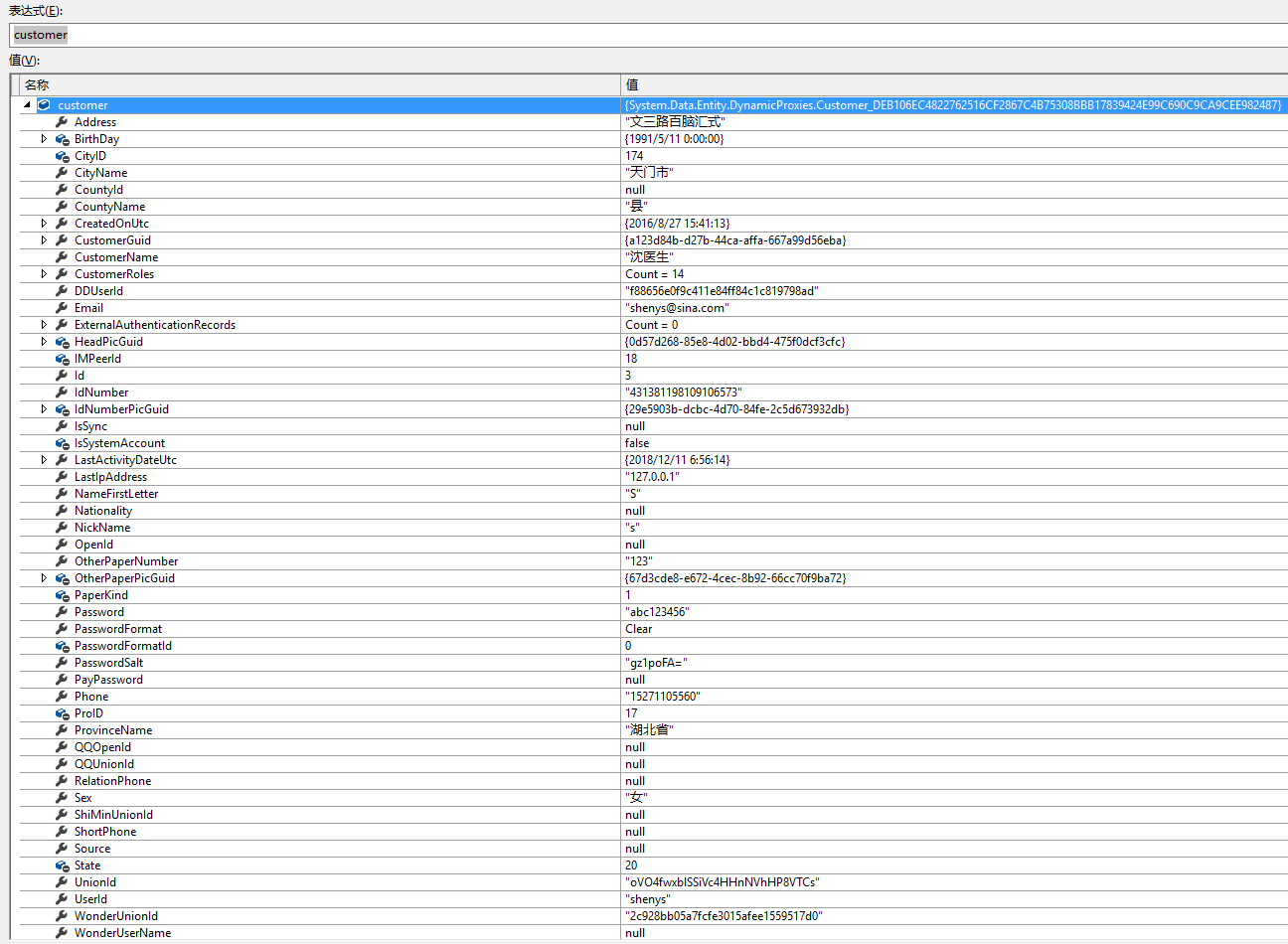
where c.CustomerGuid == customerGuid

orderby c.Id

select c;

var customer = query.FirstOrDefault();

这里就拿到了登陆用户的信息：



return customer;

}

（3）

其实这里应该是前置，

/// <summary>

/// 从Cookie凭据中获取授权用户

/// </summary>

public virtual Customer GetAuthenticatedCustomer()

{

if (\_cachedCustomer != null)

return \_cachedCustomer;

if (\_httpContext == null || \_httpContext.Request == null)

return null;

if (!\_ssoProvider.IsLogin)

return null;

Guid? CustomerGuid = \_ssoProvider.CurrentCustomerGuid;

拿到guid

if (CustomerGuid == null)

return null;

var customer = \_customerService.GetCustomerByGuid(CustomerGuid.GetValueOrDefault());

if (customer != null && customer.State >= Convert.ToInt32(CustomerState.Activated) && customer.IsRegistered())

这里应该是把当前登录用户信息存储到缓存中

\_cachedCustomer = customer;

return \_cachedCustomer;

}

（4）

跳到了Global文件的Application\_AuthenticateReques（）方法中：

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

{

//我们不这样做它的 Application\_BeginRequest ,因为用户还未通过身份验证

SetWorkingCulture();

}

关于这个方法的查询：

Application\_AuthenticateRequest：在安全模块建立起当前用户的有效的身份时，该事件被触发。在这个时候，用户的凭据将会被验证。

protected void SetWorkingCulture()

{

if (!AuthorizationSettingsHelper.ProgramAuthorzation())

return;

if (!DataSettingsHelper.DatabaseIsInstalled())

return;

//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 = string.Format("{0}keepalive/index", webHelper.GetWebSiteLocation());

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

return;

if (webHelper.GetThisPageUrl(false).StartsWith(string.Format("{0}admin", webHelper.GetWebSiteLocation()),

StringComparison.InvariantCultureIgnoreCase))

{

//admin area

CommonHelper.SetTelerikCulture();

}

else if (webHelper.GetThisPageUrl(false).StartsWith(string.Format("{0}service", webHelper.GetWebSiteLocation()),

StringComparison.InvariantCultureIgnoreCase))

{

}

else

{

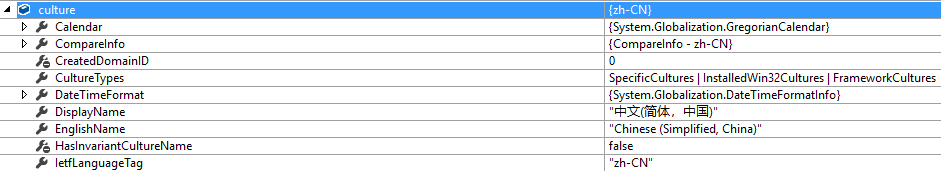
//public WebSite

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

if (workContext.CurrentCustomer != null && workContext.WorkingLanguage != null)

{

var culture = new CultureInfo(workContext.WorkingLanguage.LanguageCulture);



Thread.CurrentThread.CurrentCulture = culture;

Thread.CurrentThread.CurrentUICulture = culture;

}

}

}

（5）

又走到了这里：

namespace eWorld.Web.Framework

{

public class CustomerLastActivityAttribute : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

if (!AuthorizationSettingsHelper.ProgramAuthorzation())

return;

if (!DataSettingsHelper.DatabaseIsInstalled())

return;

if (filterContext == null || filterContext.HttpContext == null || filterContext.HttpContext.Request == null)

return;

//不要将过滤器应用于子方法

if (filterContext.IsChildAction)

return;

//只有GET请求

if (!String.Equals(filterContext.HttpContext.Request.HttpMethod, "GET", StringComparison.OrdinalIgnoreCase))

return;

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

var customer = workContext.CurrentCustomer;

//更新最后活动日期

if (customer != null && customer.LastActivityDateUtc.AddMinutes(1.0) < DateTime.UtcNow)

{

var customerService = EngineContext.Current.Resolve<ICustomerService>();

customer.LastActivityDateUtc = DateTime.UtcNow;

customerService.UpdateCustomer(customer);

}

}

}

（6）

终于跳到了Action中：

public ActionResult Index()

{

var crmUrl = \_crmSettings.Url;



var currentUrl = Request.Url.ToString();



if (!\_ssoProvider.IsLogin)

{

var queryToken = HttpContext.Request["tk"];

if (string.IsNullOrEmpty(queryToken))

{

//跳转到GetToken界面

return Redirect(\_ssoProvider.GetTokenUrl(crmUrl, currentUrl));

}

else//有token或为$Token$字符串

{

if (queryToken == "$Token$")

{

return Redirect(\_ssoProvider.GetLoginUrl(crmUrl, currentUrl));

}

else

{

currentUrl = Regex.Replace(currentUrl, @"\?tk=[^&]\*&?", string.Empty);

currentUrl = Regex.Replace(currentUrl, @"&tk=[^&]\*&?", string.Empty);

//Redis凭据已删除，但CRM的Cookie仍在

if (\_ssoProvider.GetCustomerGuidByQueryToken(queryToken) == null)

return Redirect(\_ssoProvider.GetLoginUrl(crmUrl, currentUrl));

//Token写入Cookie中

\_ssoProvider.SubSiteLogin(queryToken);

return Redirect(currentUrl);

}

}

}

因为已经验证到登陆，直接跳到这里，然后进入到Home的首页：



return RedirectToRoute("HomePage");

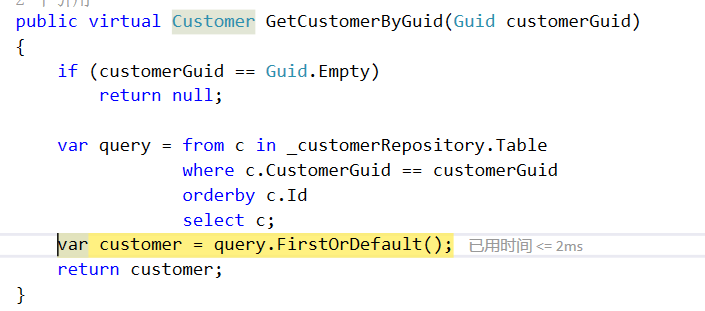
}

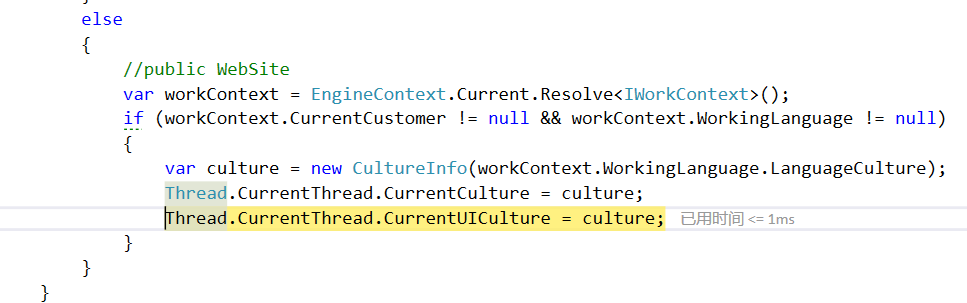
（7）

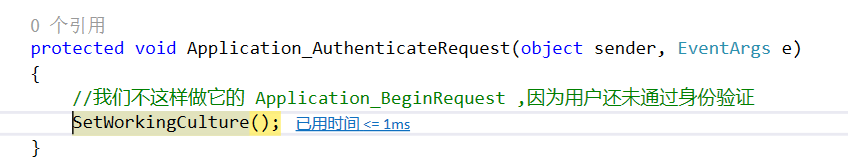
接下来就是跳转去Home/Index了。又进行循环了，跳到UserId，跳到customer



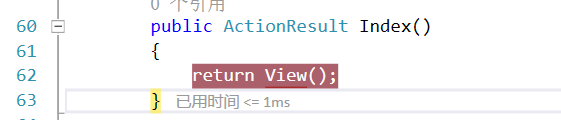








最后跳到了Home/Index





（8）

11.

Cookie: crm\_token=8FF2D13B6F60EB9C5233262F94CC918A7A0DFBD8E9F40B1819721C5AFF38C2E4C881EFB92AA778D5; eWorldToken=8F729D6F63E2A696A4E2FF62B97B42C1DDDBBF3746C1B80775B949596594D86DE6F735934C2413098C54C0B8F6BF2EA845BF9CF46F0BAB830578C851159A2D7CDC43CB076FE77CB92325B7268993189964A22C2FCB94728FFFA5A8F9E67892D14F68BC4666CF620045E4611FE9A3C18CA60BEF0529F589C0;

ASP.NET\_SessionId=t5yjtselbleyczf2sch1xuj3; \_\_RequestVerificationToken=KMyj9Dvi7NFSxbOfeKNmdK\_CL1\_nejbI5YkvoKhnDLQkUUxRs-2DbC-Rrg9\_mS-wp3Dotznzy8pQ4vumhWsUFsc0QcxtJCFxe3Yrxlk4Zi01

12.

单点登陆，以特性的形式，最后打在了BaseController上。

namespace eWorld.Web.Framework.Controllers

{

/// <summary>

/// 权限过滤器 根据一角色一过滤器 主要在个人中心或是后台管理的时候用上

/// </summary>

[AttributeUsage(AttributeTargets.Method | AttributeTargets.Class, Inherited = true, AllowMultiple = true)]

public class AdminAuthorizeAttribute : FilterAttribute, IAuthorizationFilter

/// <summary>

/// SSO单点登录

/// </summary>

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

public void SSOHandle(AuthorizationContext filterContext)

{

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

var currentCustomer = workContext.CurrentCustomer;

//未登录

if (!IsLogin(currentCustomer))

{

var queryToken = filterContext.HttpContext.Request["token"];

if (queryToken != null)

{

if (queryToken == "$Token$")

{

filterContext.Result = new RedirectResult(getTokenURL(filterContext));

return;

}

//持有令牌

//调用WebService获取主站凭证

string webServiceUrl = string.Format("{0}/TokenService.asmx?wsdl", FormsAuthentication.LoginUrl);

WebServiceProxy proxy = new WebServiceProxy(webServiceUrl);

object[] para = new object[1] { queryToken };

var o = proxy.ExecuteQuery("TokenGetCredence", para);

if (o != null)

{

var \_authenticationService = EngineContext.Current.Resolve<IAuthenticationService>();

\_authenticationService.SignIn((string)o, false);

//令牌正确

string sourceUrl = filterContext.HttpContext.Request.Url.AbsoluteUri;

sourceUrl = Regex.Replace(sourceUrl, @"\?Token=[^&]\*&?", string.Empty);

sourceUrl = Regex.Replace(sourceUrl, @"&Token=[^&]\*&?", string.Empty);

filterContext.Result = new RedirectResult(sourceUrl);

return;

}

else

{

//令牌错误

filterContext.Result = new RedirectResult(GetLoginURL(filterContext));

return;

}

}

else

{

filterContext.Result = new RedirectResult(getTokenURL(filterContext));

return;

}

}

//登录后检验权限

var permissionService = EngineContext.Current.Resolve<IPermissionService>();

bool result = permissionService.Authorize(StandardPermissionProvider.AdministratorAccessPanel);

if (!result)

{

filterContext.Result = new RedirectResult("~/NoPermission");

return;

}

}

上面的SSO单点登录在public void OnAuthorization(AuthorizationContext filterContext)方法中执行。

/// <summary>

/// 关于授权

/// </summary>

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

public void OnAuthorization(AuthorizationContext filterContext)

{

if (\_dontValidate)

return;

if (filterContext == null)

throw new ArgumentNullException("filterContext");

if (OutputCacheAttribute.IsChildActionCacheActive(filterContext))

throw new InvalidOperationException("您无法使用[AdminAuthorize]属性，当一个子动作缓存处于活动状态");

if (IsAdminPageRequested(filterContext))

{

SSOHandle(filterContext);

}

}

查找授权在什么时候执行：

查找：

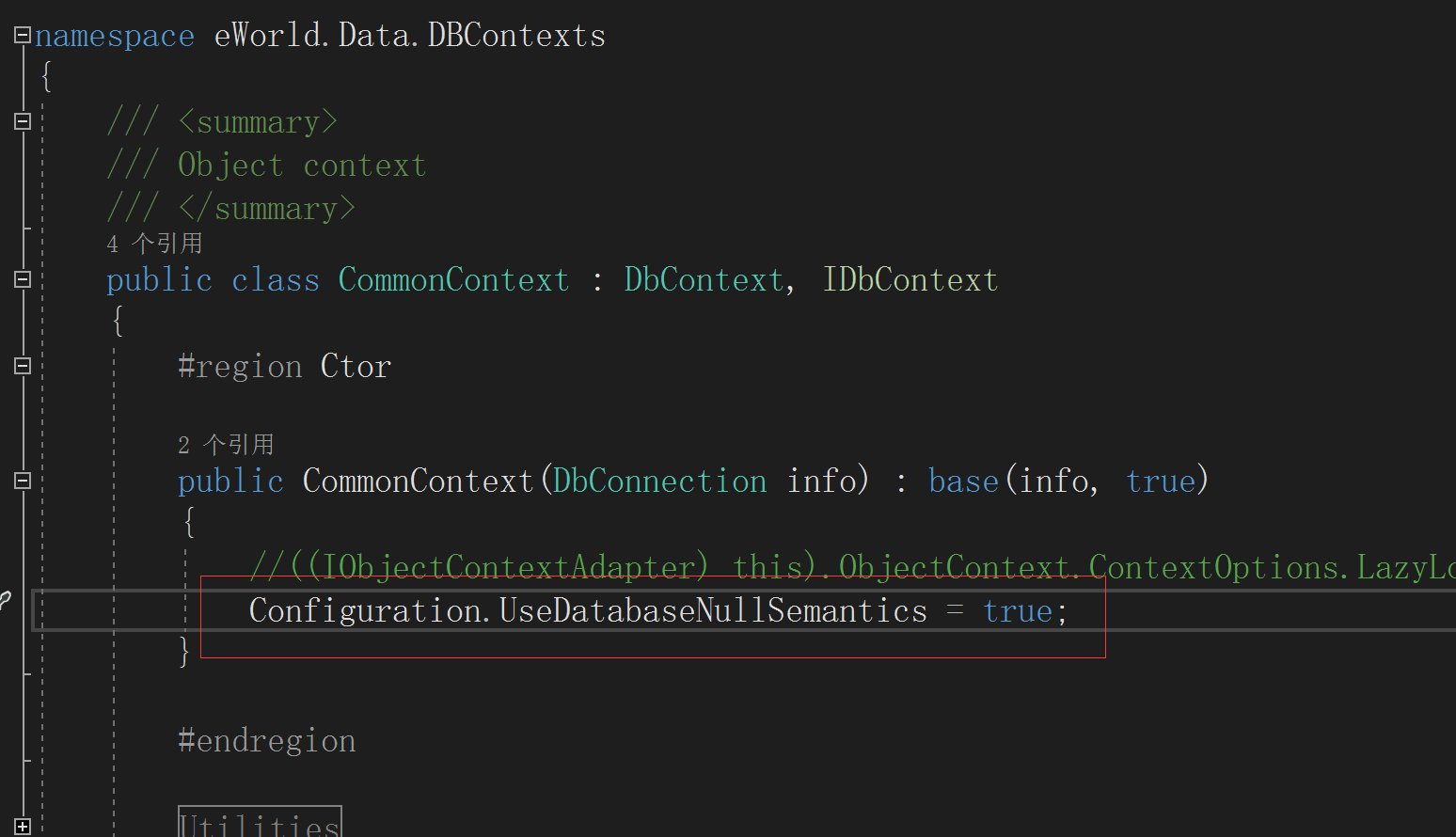
执行顺序，OnAuthorization在AuthorizeAttribute特性种，

先执行OnAuthorization，然后执行AuthorizeCore函数；

OnActionExecuting方法在ActionFilterAttribute特性中，

同时存在OnAuthorization和OnActionExecuting，在授权检查完毕之后执行OnActionExecuting，也就是说OnActionExecuting后执行

13.



产品-孔钧

利用值查询在WHERE条件上没有过多的条件过滤，而利用参数查询则是生成过多的条件筛选，到这里是不是就到此为止了呢，如果您对于参数查询不想生成对空值的过滤，我们在上下文构造函数中可关闭这种所谓【语义可空】判断

14.

15.

16.

17.

18.