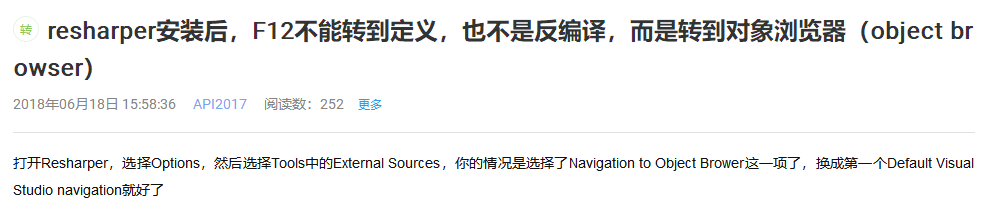
记录some

是

终于搞定了resharper



[http://localhost:3410/Image/Viewer?DicomDirPath=E:\work\ct-dhw\DICOMDIR](http://localhost:3410/Image/Viewer?DicomDirPath=E:/work/ct-dhw/DICOMDIR)

1. 访问：http://localhost:3410/Image/Viewer?DicomDirPath=E:\work\ct-dhw\DICOMDIR
2. 过滤器验证[TokenAuthentication]
3. 到家后，访问失败。应该是没有accessKey



全解决方案查找后，发现在js文件中。

自己的感觉是，前台的所有操作都风筝成了js操作。然后在view中导入。







这个Front应该就是前端封装好的js代码。

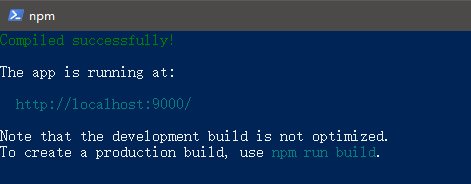
是

综合点：

1. @Styles.Render("~/Front/style/ImageStyle")调用css
2. @Scripts.Render("~/bundles/ImageViewer")

是

启动代码 npm start



在这个目录当中，

E:\ywbFromCdm\eWorldWebApplication

右键：

Vs2015的设置



React:

<http://localhost:9000/?DicomDirPath=E:/work/ct-dhw/DICOMDIR>

Vs2015：

[http://localhost:3410/Image/Viewer?DicomDirPath=E:\work\ct-dhw\DICOMDIR](http://localhost:3410/Image/Viewer?DicomDirPath=E:/work/ct-dhw/DICOMDIR)

逻辑都在business中。

Api就在api中。

重点要看dicom那块。



这个Dicomdir是一个索引文件。

第一天：

熟悉PACS，熟悉整个项目的需求——对外提供查询和下载Dicom文件的接口。

第二天：

熟悉ReactJs和Convas。30%

第三天：

熟悉WebConfig配置，了解Study和Dicom接口的流程。请求Study检查信息，返回检查信息。请求Dicom文件，以及文件流返回。

20%

NAVH-F6IC-SQ3U-6LW5

NAVA-52HL-7CWZ-KPXJ

（1）

log4Net的配置没有的Application\_Start中，而是在静态方法中，这样调用Log4Helper 的时候，自动调用：

namespace eWorld.Web.Framework.AppClass

{

/// <summary>

/// 日志帮助类

/// </summary>

/// <typeparam name="T">当前异常对象</typeparam>

public static class Log4Helper<T> where T : class

{

static Log4Helper()

{

log4net.Config.XmlConfigurator.Configure(new System.IO.FileInfo(System.Web.Hosting.HostingEnvironment.ApplicationPhysicalPath + @"\log4net.config"));//System.Configuration.ConfigurationManager.AppSettings["log4config"]

}

（2）

过滤器问题，之前都是Add进去，全局过滤器。这里是申明静态的全局过滤器集合：

以前：

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

//filters.Add(new HandleErrorAttribute());

filters.Add(new MyExceptionFilterAttribut());

//filters.Add(new LoginCheckFilterAttribute(){IsCheck = true});

//ActionFilter ResultFilter

//第三种filter：异常过滤器。

}

}

现在：

public static class GlobalFilters

{

/// <summary>

/// Gets or sets the global filter collection.

/// </summary>

///

/// <returns>

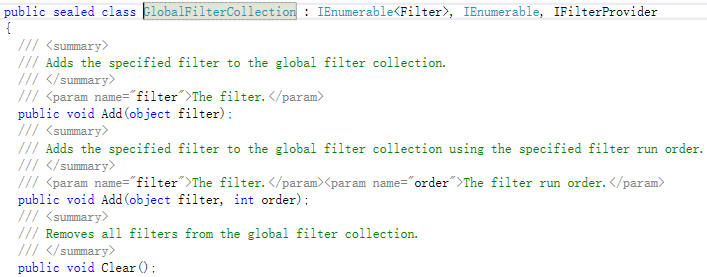
/// The global filter collection.

/// </returns>

public static GlobalFilterCollection Filters { get; }

}

其中这个全局过滤器集合，里面就包含了Add，Clear这些方法：



暂时不知道是怎么加进去的，在想，也许是autoFac。

（3）

居然还涉及到了引擎的初始化，先全部删除，再Add所需要的，



（4）

远程医疗协同访问地址：

<http://192.168.1.202:9500/portal/PlatformIndexPage>

（5）

本地一直架不起来，错误。交了喻永红帮忙。

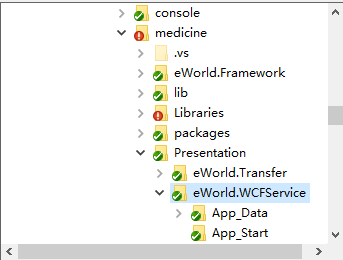


是wcf没开的缘故，在iis中重新建了wcfService。

端口9501.

Web是在9500。

wcf的物理路径地址是：



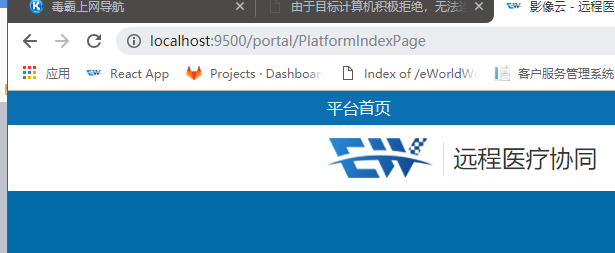
wcf能访问了：



重新生成项目之后搞定：



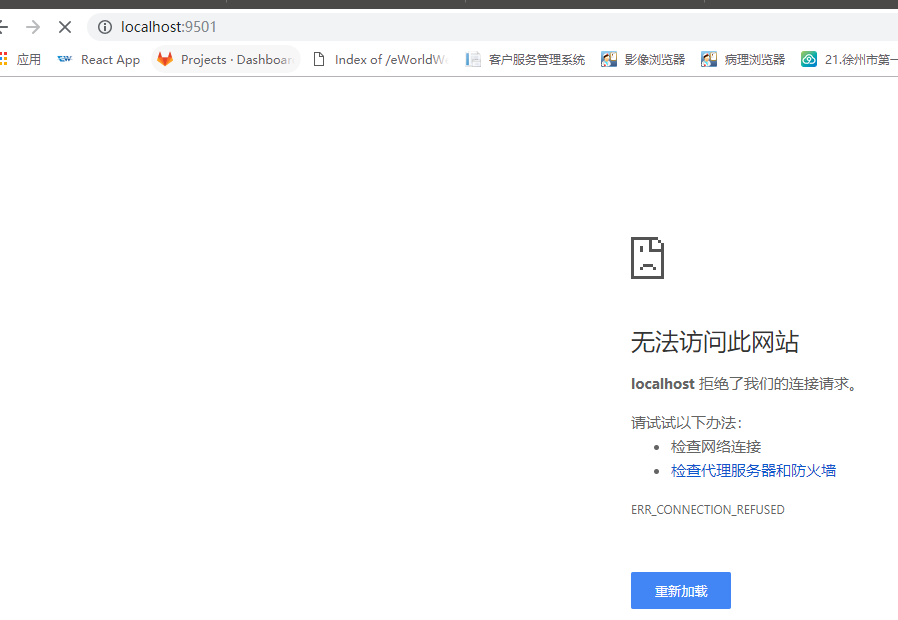
搞定，本地能跑了。



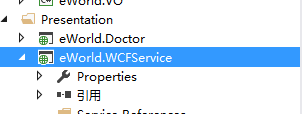
测试将iis中的wcf服务关闭，然后试试用iis express单独启动。

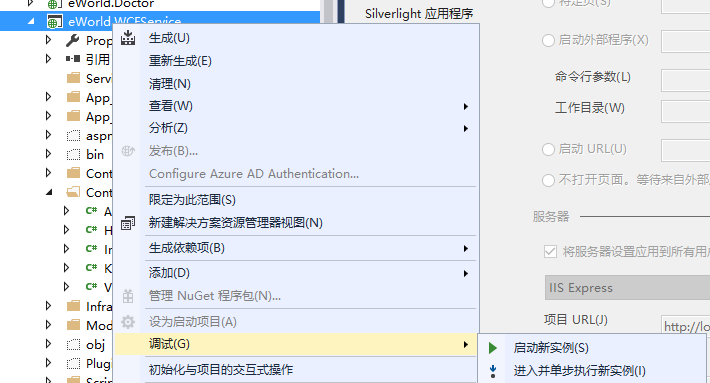


关闭后就看不到了：



然后用iis单独调试wcf





发现也是ok的。



所以正常步骤是先单例调试生成wcf，然后去访问Portal入口的首页Action方法。

现在就用自带的iis时刻保持wcf服务的开启中。

测试完毕后，又重新启动这个wcf服务。



（6）

任务：

   本周熟悉代码    
  
  还有一个事情  也需要你跟我们一起弄  
  
研发部-李阳 2018/10/15 11:07:20  
  
   就是安全性测评  
  
研发部-李阳 2018/10/15 11:07:23  
  
   黄老板 咋样  
  
研发部-李阳 2018/10/15 11:07:24  
  
   哈哈  
  
研发部-李阳 2018/10/15 11:08:45  
  
  安全性测评  估计大部分都是前端的事情   
11:08:55  
研发部-李阳 2018/10/15 11:08:55  
  
   因为后端 测不出啥  安全性的问题

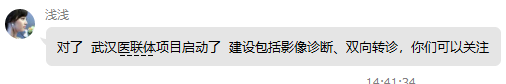
（7）

展开的快捷键：

Ctrl+M,L: 将所有过程设置为相同的隐藏或展开状态

（8）

qq记录：



（9）

正片，开始访问首页了。

a.

访问首页的时候，先看有没有cookie，如果errormsg为null，则新建一个改cookie，值为空。并且过期时间为一个小时。

#region view[平台首页]

public ActionResult PlatformIndexPage()

{

if (HttpContext.Request.Cookies["errormsg"] == null)

{

HttpCookie mycookie = new HttpCookie("errormsg");

mycookie.Value = "";

mycookie.Expires = DateTime.Now.AddHours(1);

Response.Cookies.Add(mycookie);

}

return View();

}

感觉没什么逻辑啊，看样子逻辑都去了View中了。

b.

diagnosis：诊断

View，就是模板页和当前页的整合，可以通过查看网页源代码去看。

可以看到模板页面和当前页面的整合效果，感觉low low的。和WebForm的方式很像，而且只有一个body坑，只需要填一个坑就可以了。



搞了半天，前台代码我们不用管的，我们只管后台代码，以固定的json格式发给前端，前端自己拿。

需要格外说明的，比方下面的验证啊，登陆信息啊，这些要我们写在前台，并且给前端的人。

@{

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

var version = (new ProgramVersion()).MainVersion;

var customerIdentity = ViewData["CustomerIdentity"] as CustomerIdentity ?? new CustomerIdentity();

Guid? headPicGuid = Guid.Empty;

var headPicImg = "";

if(customerIdentity.HeadPicGuid != null)

{

headPicGuid = customerIdentity.HeadPicGuid;

if (headPicGuid != null && headPicGuid != Guid.Empty)

{

try

{

headPicImg = BaseCommon.GetImage(headPicGuid).FileUrl;

}

catch { }

}

}

}

<!DOCTYPE html>

c：

就是上面的验证用户这一块，它是直接通过view页面，在开头，通过@razor代码，ViewData[“索引”]传给前端的。然后前端直接拿到这个值了，在模板中需要的地方直接加入。

var customerIdentity = ViewData["CustomerIdentity"] as CustomerIdentity ?? new CustomerIdentity();

基本每个view页面的抬头都有这段用户信息的代码了。唉。居然不是强类型传参。

终于有看到一个用实体，model的了，欣慰。



终于找到登录用户的处理逻辑了，

namespace eWorld.Web.Framework.Controllers

{

/// <summary>

///

/// </summary>

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);

if (principal != null)

{

requestContext.HttpContext.User = principal;

ViewData["CustomerIdentity"] = principal.Identity;

}

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

}

d：

原有的控制器controller类是抽象abstract的。

public abstract class Controller : ControllerBase, IActionFilter, IAuthenticationFilter, IAuthorizationFilter, IDisposable, IExceptionFilter, IResultFilter, IAsyncController, IController, IAsyncManagerContainer

{

建立abstract抽象的BaseController类去继承抽象Controller类，把其中的空的virtual虚方法给部分实现，继续定义为virtual虚方法。

namespace eWorld.Web.Framework.Controllers

{

/// <summary>

/// Base controller

/// </summary>

[WebSiteIpAddress]

[CustomerLastActivity]

[WebSiteLastVisitedPage]

public abstract class BaseController : Controller

{

/// <summary>

/// 呈现局部视图串

/// </summary>

/// <returns>Result</returns>

public virtual string RenderPartialViewToString()

{

return RenderPartialViewToString(null, null);

}

/// <summary>

/// 呈现局部视图串

/// </summary>

/// <param name="viewName">View name</param>

/// <returns>Result</returns>

public virtual string RenderPartialViewToString(string viewName)

{

return RenderPartialViewToString(viewName, null);

}

/// <summary>

/// 呈现局部视图串

/// </summary>

/// <param name="model">Model</param>

/// <returns>Result</returns>

在接下来的类的继续继承中，防火墙控制器FirewallMvcController的是普通类，是可以实例化的，再彻底去重写这些虚方法。

namespace eWorld.Web.Framework.Controllers

{

/// <summary>

///

/// </summary>

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);

if (principal != null)

{

requestContext.HttpContext.User = principal;

ViewData["CustomerIdentity"] = principal.Identity;

}

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

}

private IPrincipal GetCustomerPrincipal(RequestContext requestContext)

{

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

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

if (customerCookie?.Value != null)

{

if (customer == null)

{

return CustomerPrincipal.GuestCustomerPrincipal;

}

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

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

{

return CustomerPrincipal.GuestCustomerPrincipal;

}

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

if (!StackExchangeRedisHelper.Exists(key))

{

return CustomerPrincipal.GuestCustomerPrincipal;

}

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

customerIdentity.Authorized = true;

return new CustomerPrincipal(customerIdentity);

}

return CustomerPrincipal.GuestCustomerPrincipal;

}

/// <summary>

/// 用户信息

/// </summary>

public CustomerIdentity CustomerIdentity => User.Identity as CustomerIdentity;

}

}

查找mvc5控制器controller的异步调用。

我们发现MvcHandler不止实现的IHttpHandler即接口，还实现了异步的IHttpAsyncHandler接口，那么如果程序不调用同步的ProcessRequest方法，那就一定是使用的异步的BeginProcessRequest方法。

　　这是正确的，MVC5使用的异步的BeginProcessRequest方法，接下来我们去BeginProcessRequest方法中看看有哪些秘密吧。

IMG_256

[IMG_257](https://www.cnblogs.com/zxj159/archive/2014/11/22/javascript:void(0);)

protected virtual IAsyncResult BeginProcessRequest(HttpContext httpContext, AsyncCallback callback, object state)

{

HttpContextBase httpContextBase = new HttpContextWrapper(httpContext);

return BeginProcessRequest(httpContextBase, callback, state);

}

[IMG_258](https://www.cnblogs.com/zxj159/archive/2014/11/22/javascript:void(0);)

　　向下找

[IMG_259](https://www.cnblogs.com/zxj159/archive/2014/11/22/javascript:void(0);)

protected internal virtual IAsyncResult BeginProcessRequest(HttpContextBase httpContext, AsyncCallback callback, object state)

{

IController controller;

IControllerFactory factory;  
 **//创建控制器**

**ProcessRequestInit(httpContext, out controller, out factory);**

IAsyncController asyncController = controller as IAsyncController;

if (asyncController != null)

{

// asynchronous controller

// Ensure delegates continue to use the C# Compiler static delegate caching optimization.

BeginInvokeDelegate<ProcessRequestState> beginDelegate = delegate(AsyncCallback asyncCallback, object asyncState, ProcessRequestState innerState)

{

try

{  
  **// Action**

**return innerState.AsyncController.BeginExecute(innerState.RequestContext, asyncCallback, asyncState);**

}

catch

{  
 **// 释放控制器**

**innerState.ReleaseController();**

throw;

}

};

EndInvokeVoidDelegate<ProcessRequestState> endDelegate = delegate(IAsyncResult asyncResult, ProcessRequestState innerState)

{

try

{

innerState.AsyncController.EndExecute(asyncResult);

}

finally

{

innerState.ReleaseController();

}

};

ProcessRequestState outerState = new ProcessRequestState()

{

AsyncController = asyncController, Factory = factory, RequestContext = RequestContext

};

SynchronizationContext callbackSyncContext = SynchronizationContextUtil.GetSynchronizationContext();

return AsyncResultWrapper.Begin(callback, state, beginDelegate, endDelegate, outerState, \_processRequestTag, callbackSyncContext: callbackSyncContext);

}

else

{

// synchronous controller

Action action = delegate

{

try

{

controller.Execute(RequestContext);

}

finally

{

factory.ReleaseController(controller);

}

};

return AsyncResultWrapper.BeginSynchronous(callback, state, action, \_processRequestTag);

}

}

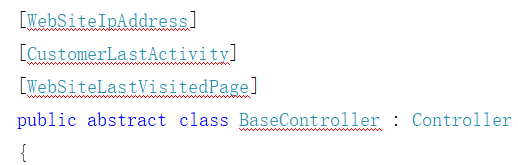
访问的时候会查找这个cookie——eWorldToken。



访问的时候进行解码：



调试过程找到这个WebSiteIpAddressAttribute过滤器了。



namespace eWorld.Web.Framework

{

/// <summary>

///

/// </summary>

public class WebSiteIpAddressAttribute : ActionFilterAttribute

{

/// <summary>

///

/// </summary>

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

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

//更新IP地址

string currentIpAddress = webHelper.GetCurrentIpAddress();

if (!String.IsNullOrEmpty(currentIpAddress))

{

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

var customer = workContext.CurrentCustomer;

if (customer != null && !currentIpAddress.Equals(customer.LastIpAddress, StringComparison.InvariantCultureIgnoreCase))

{

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

//customer.LastIpAddress = currentIpAddress;

//customerService.UpdateCustomer(customer);

}

}

}

}

}

第二个过滤器也执行：

namespace eWorld.Web.Framework

{

/// <summary>

/// 记录用户最后登录时间

/// </summary>

public class CustomerLastActivityAttribute : ActionFilterAttribute

{

/// <summary>

///

/// </summary>

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

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

if (!DataSettingsHelper.DatabaseIsInstalled())

return;

if (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.AddSeconds(30) < DateTime.Now)

{

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

customerService.UpdateLastActivityDateUtc(customer.UserId);

}

}

}

}

第三个过滤器来了：

namespace eWorld.Web.Framework

{

/// <summary>

///

/// </summary>

public class WebSiteLastVisitedPageAttribute : ActionFilterAttribute

{

/// <summary>

///

/// </summary>

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

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;

//仅GET请求

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

return;

var customerSettings = EngineContext.Current.Resolve<CustomerSettings>();

if (!customerSettings.WebSiteLastVisitedPage)

return;

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

var pageUrl = webHelper.GetThisPageUrl(true);

if (!String.IsNullOrEmpty(pageUrl))

{

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

var genericAttributeService = EngineContext.Current.Resolve<IGenericAttributeService>();

var previousPageUrl = workContext.CurrentCustomer.GetAttribute<string>(SystemCustomerAttributeNames.LastVisitedPage);

if (!pageUrl.Equals(previousPageUrl))

{

genericAttributeService.SaveAttribute(workContext.CurrentCustomer, SystemCustomerAttributeNames.LastVisitedPage, pageUrl);

}

}

}

}

}

打上标签，依次执行，这几个控制器方法过滤器。

三个过滤器执行完了，才执行这个：

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);

if (principal != null)

{

requestContext.HttpContext.User = principal;

ViewData["CustomerIdentity"] = principal.Identity;

}

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

}

接下来进行平台跳转：

#region 平台跳转

public ActionResult PlatePortalTransfer()

{

var url = Request.Url;

var requestDomain = url == null ? "" : url.Authority;

var centerId = 0;

//根据访问页的域名查询服务中心ID begin

if (!string.IsNullOrWhiteSpace(requestDomain))

{

var result = \_crmService.Invoke(t => t.GetCenterInfoByServiceDNS(requestDomain));

if (result.code == ResultStatusCode.OK)

{

if (result.data != null)

{

centerId = result.data.ServiceCenterId;

}

}

}

//根据访问页的域名查询服务中心ID end

if (centerId == 0)

return RedirectToAction("PlatformIndexPage");

else

return Redirect("portal/PlatformIndex?centerid=" + centerId);

}

接下来走EndRequest方法（应该就是23个公开事件的最后一个了）：

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

{

//忽略静态资源

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

if (webHelper.IsStaticResource(this.Request))

return;

if (CanPerformProfilingAction())

{

//停止MvcMiniProfiler.MiniProfiler

MiniProfiler.Stop();

}

//配置注册资源

//我们不为IHttpModule注册AutofacRequestLifetimeHttpModule

//因为其配置资源，这Application\_EndRequest方法之前被调用

//在这种情况下，上面的代码将抛出一个异常

//UPDATE: we cannot do it. For more info see the following forum topic - http://www.nopcommerce.com/boards/t/22456/30-changeset-3db3868edcf2-loaderlock-was-detected.aspx

//AutofacRequestLifetimeHttpModule.ContextEndRequest(sender, e);

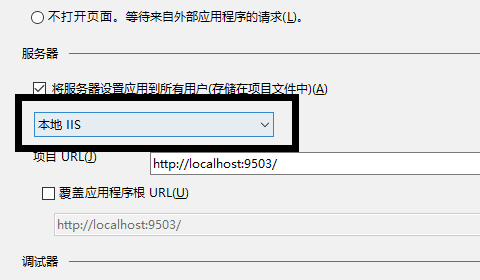
}

（10）

web api不显示，



李阳让我本地建立iis

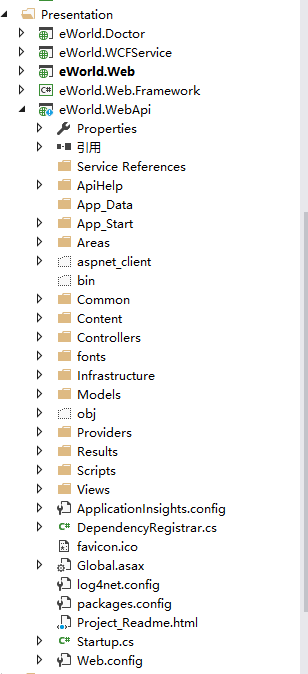
  
  
研发部-李阳 2018/10/16 10:20:12  
  
   因为这里用的是本地IIS  
  
研发部-李阳 2018/10/16 10:20:29  
  
    你可以在本地IIS 里面创建一个网站

我创建：



发现没有权限，用管理员运行了vs。

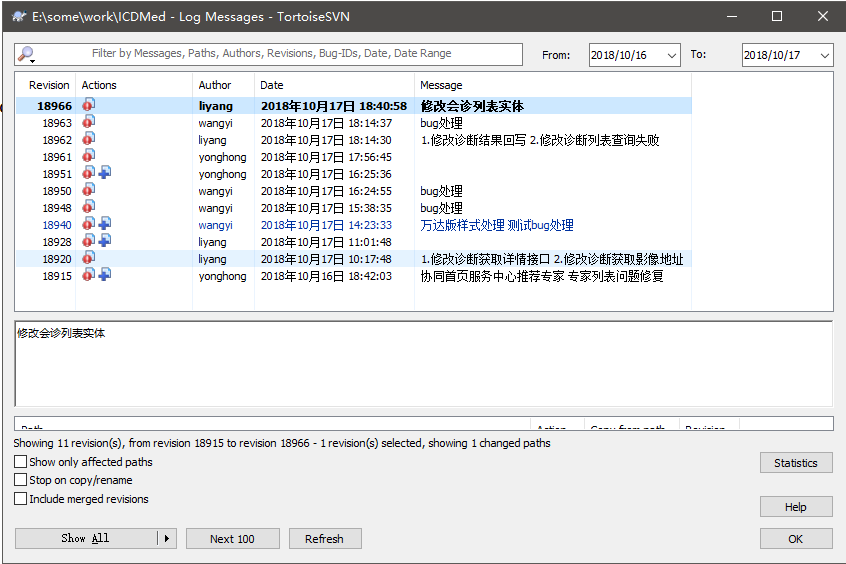
搞定！开森！



看了下属性：



看到了svn日志：

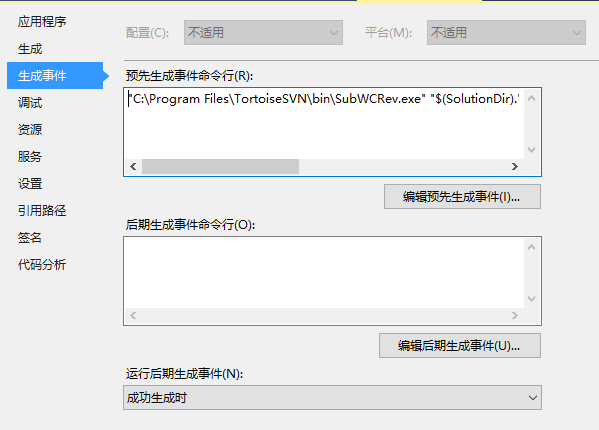


（2018年10月17日17:23:13）

老孔在和王毅说前后端分离，问会不会node.js，说起要真正前后端分离，就要会这个，让他了解一下，因为以后准备实现真正的前后端分离。前端通过web api去调用数据。

之前在苗哥那里，苗哥已经这么做了。

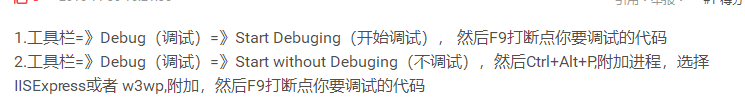
说起淘宝什么的。我发现孔哥真的很能讲。让搜索前后端分离，去看。



报错，svn重新装到默认目录。

搞定。

如何调试问题：



在日报那里倒是看到了模块：

