一些整理

是

1.

公司的数据库层，eWorld.Data这种是封装好的，应该是单独抽出来一层，然后加入到所有的各个团队，比方十个团队，十个团队对于数据库的需求是不一样的，那不同团队自由去访问，但所有的数据库入口都给团队了。

自己写的DAL层就可以对各个数据库进行操作，通过ef。各个数据库仓库已经封装ok。

BLL层是各种逻辑的判断。事务这些都是在这里。

其中各个层之间，参数走的是DTO，接口层是VO，需要MapTo转到DTO中的类。

WCF契约层，契约，接口写在这里。

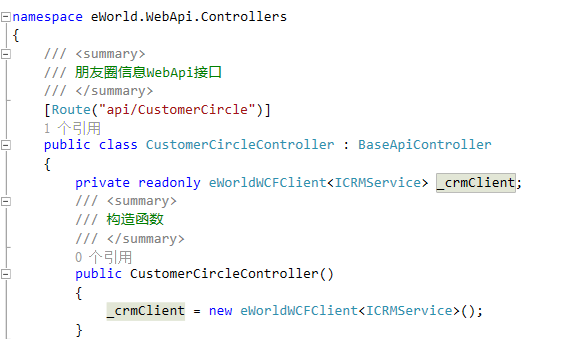
在实现中，统一用\_crmManage去操作方法。其实这些方法都是BLL层的方法。

这应该就是老马讲的，新加一层wcf服务层，之前写的DAL层，BLL层不变，本来是UI层去操作BLL层，这里是在WCF层中去操作BLL层中所有的方法，再把结果对外开放。

return \_crmManage.DeleteFriend(param);

客户端这边，其实我现在开发这边，我这台电脑又是服务端，自己写契约，又是客户端，我的WebApi是配置在本地9503端口的，之前就配置好的，默认就是wcf client客户端。因为API端可以默认地去调用wcf服务端的任何方法。

看了，WebApi这里就是wcf的客户端client：



然后别人访问我的api，就可以通过各种方法来实现通信了。

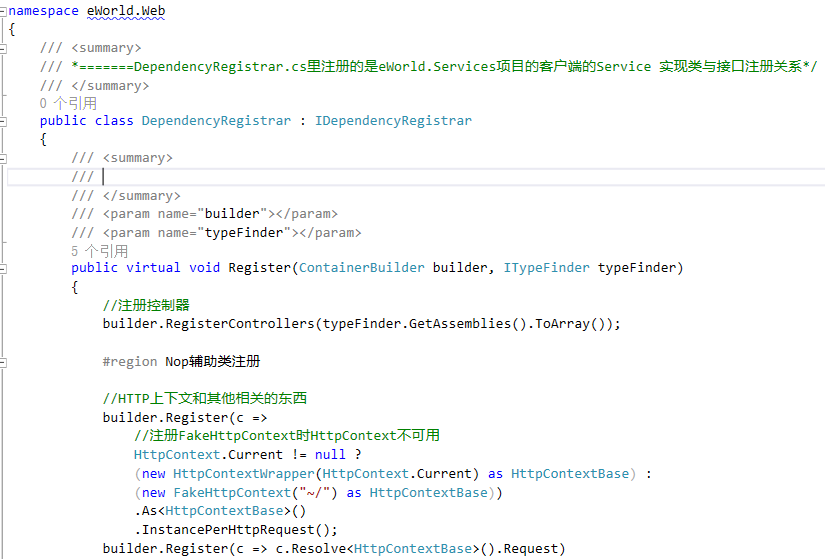




eWorld.WebApi中又一个注入的类。



我自己猜测就是和wcf配置了契约，可以进行通信什么的。



不是的，适合Web端一样，调用的同样的wcf client对象。都是指向——

eWorld.WCFServerContract

这里我自己写的朋友圈接口，是只用到了CRM，所以只创建了一个crmClient，多个的话，就创建多个wcf client客户端。



继续看到Web端，发现Web端也是wcf的客户端Client，和WebApi是一样的，都是通过wcf服务去和bll层，dal层，以及数据库进行交互：



通过eWorld.WCFServerContract去和wcf服务端取得通信：





这里可以看出，Web端和Web Api端都是UI端，调用WCF服务和BLL，DAL，数据库进行交互。现在不确定数据库是否独立，但由svn来说，似乎是分开的。先假设是分开的。

三层：

UI：Web和WebApi

服务层：WCF服务+BLL+DAL

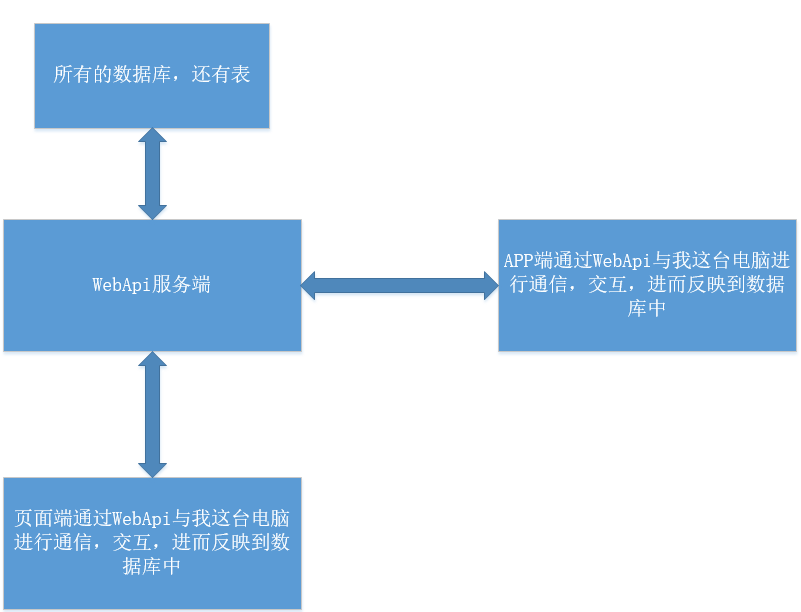
数据：数据库



衍生下去就是UI端随便加，比方普通的Winform客户端，也不对，不像以前，每个端，比方Winform或者手机端，都来调用wcf服务端，现在单独部署了WebApi端，那手机端，Winform端等等，都直接调用api接口就可以了。更方便。

所以，这里的WebApi端其实是对wcf端的一个封装，一个升级。连web端都可以直接去调WebApi端。

喻大神也说了，以后的趋势是把服务端全部更新为WebApi，这个wcf服务端要渐渐地抛弃，不去维护了。



如上图，之后服务端变成了WebApi，而所有的应用端，UI端，都通过WebApi服务端与数据库进行交互。

这样，WebApi服务端就把之前的DAL，BLL都包裹在了其中。

另外，wcf服务和web api服务都是单独部署在本地的iis中，一个是9501，一个是9503.

2.

pojo类和vo类分别是什么：

还有个保存在收藏夹了。

实际上没区别 他们的功能都一样 不过他们使用地点不一样

po是在持久层所使用的 用来封装原始数据 而VO则主要在视图层活动 两个JAVABEAN的使用范围不同 假如你从数据库取出来的原始数据 你可以通过业务层将数据进行封装 再通过VO发到页面上去 比方说

数据库表示性别的方式 为 1 和 0 1为男 0为女

获得的时候我们获得的就是原始数据 那么 经过业务层的时候我们可以进行判断 并把他们所对应的参数放到VO里 也就是说

po里为性别属性为1

经过业务层之后 判断性别 并根据原始数据给VO赋值 比如根据1 给VO里的属性赋 ‘男’

也就是说po里的属性 主要是用来给数据库用的

而VO里的属性 就是纯给用户显示的

不明白可以补充问题

3.

看来老杨的视频，发现针对接口的各种参数啊，身份啊，等等的验证是放在filter中，而当前项目是，接口，也就是ui层这类是不验证的，唯一验证的就是当前登陆用户，除此之外的验证都是放到了具体项目的bll层。

我在想，如果是验证没通过，那还要放进api么，在第一扇门应该就让它滚蛋了啊。

但是李阳这边说是，所有验证都放在bll层，自己现在的感觉，api层的逻辑判断，就应该在api，也就是ui层去搞定，可以放方法体里面，也可以直接就用filter。

4.

接下来的接口，打算Dal层全部返回IQueryable<>的类型，然后在bll层进行组装，利用延时加载去优化数据库，减少和数据库的交互次数，但是同样的问题，因为每次操作，都自动SaveChanges()，这个会有影响吗，不对，这个是查询，增删改才是SaveChanges()。无关的。

5.

我替换是替换在27，但真正发布是发布在202那台机子上的。

6.

换回主干，postMan无法访问，李阳把wcf属性设置为本地iis后，好像就好了。

7.

亮哥跟别人说起：

服务中心开通服务。

机构开通云存储功能，剩下的功能都是问服务中心开通的。因为影像的存储是以机构的名字存储的。

8.

李阳和亮哥的聊天。

Token我们后台给前端，

前端给我们token，我们解密，解密之后就有key，拿key去redis中去找值。

再将前端传来的Token跟redis中的比较，如果是，就放行，否则就不行。

这样也解决了单点登陆问题。

因为我登陆了，token变了。而后面同样存了这个token 的电脑登陆了，但是token和redis中的已经不一样了，就登陆不了了。

9.

李阳说，前端传给我的时间是，比方今天，（2018年11月17日11:33:26）回事11.17 000000和11.17 000000，所以后者默认加一天

/// 提供自定义Json

/// </summary>

public class CustomizeJsonResult : JsonResult

{

/// <summary>

///

/// </summary>

/// <param name="context"></param>

public override void ExecuteResult(ControllerContext context)

{

if (context == null)

{

throw new ArgumentNullException("context");

}

HttpResponseBase response = context.HttpContext.Response;

if (Data != null)

{

JsonSerializerSettings jsonSerializerSettings = new JsonSerializerSettings

{

DateFormatString = "yyyy-MM-dd HH:mm:ss",

ContractResolver = new DefaultContractResolver()

};

response.Write(JsonConvert.SerializeObject(Data, Formatting.Indented, jsonSerializerSettings));

}

}

}

我自己的理解，



比方前台，王毅用的控件传给我，传的是，这里时间是11.17 00：00：00，两个都是这个。

10.

当把List<short?>放入到类中，初始化之后，是null的。而new一个，反而不是null 的。

class Person

{

public List<short?> Shorts { get; set; }

}

Person p = new Person();

if (p.Shorts==null)

{

Console.WriteLine("true");

}

else

{

Console.WriteLine("false");

}

这里输出的是true。

即便是list<shorts>没有？，不是可空值类型也是一样的，类的初始化，值类型就是null。

但如果这样。

List<short?> list = null;

if (list == null)

{

}

list.Add(null);

list.Add(null);

if (list != null && list.Count > 0)

{

Console.WriteLine("true");

}

else

{

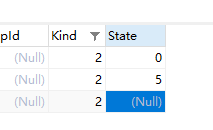
Console.WriteLine("false");

}

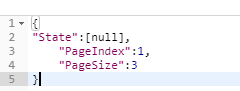
这里的list增加了null的值后，默认值并不是null了。

如果你是需要查状态里的值包含null的数据，就用short?

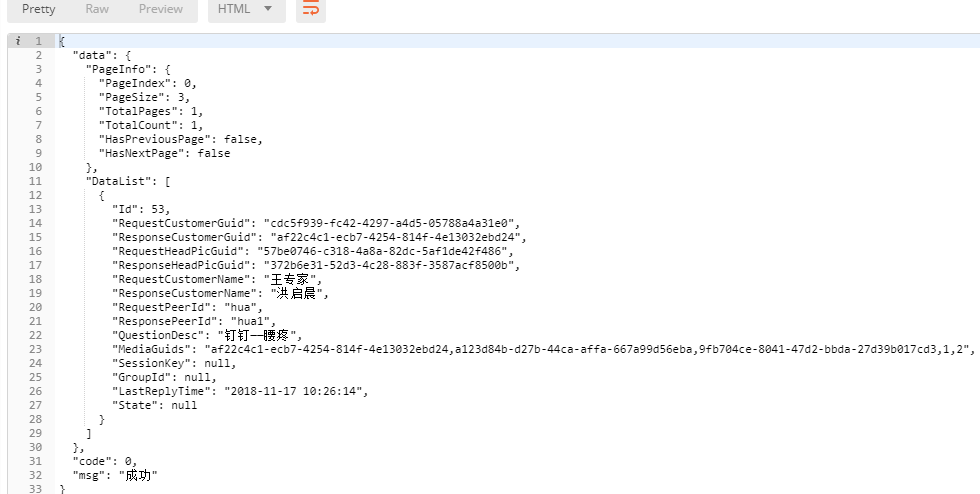
比方我把数据中的state变成了null



查询的时候传入null



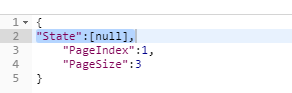
结果查到数据了，尼玛嗨。

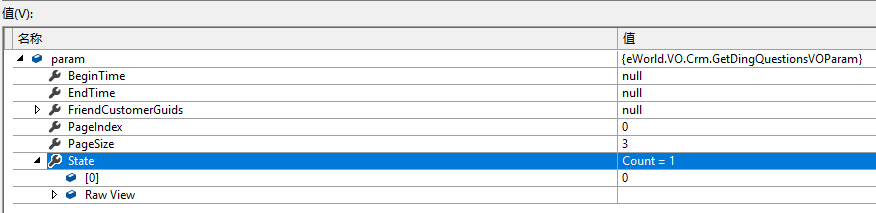


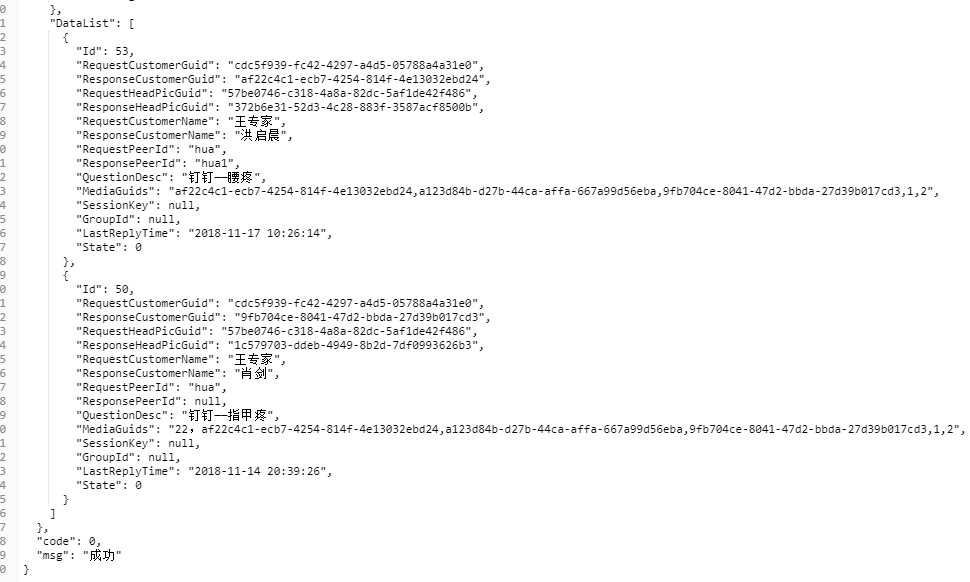
所以啊，如果需要查包括null值的数据，才用可空类型，这是我自己的一种理解。

但是改为short之后

当我仍旧传的是null之后，state的长度变成了1，而默认的第一个值就是0，变成了默认查0的方法。







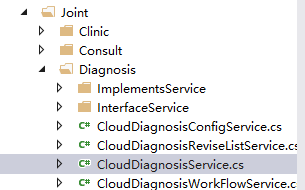
还是用这个，默认传了0，不搜索为null的值。以后如果要包含null的值，可以用这个。

而DateTime要变为可空类型的原因是因为不然默认是这个值。如果不可空的话。



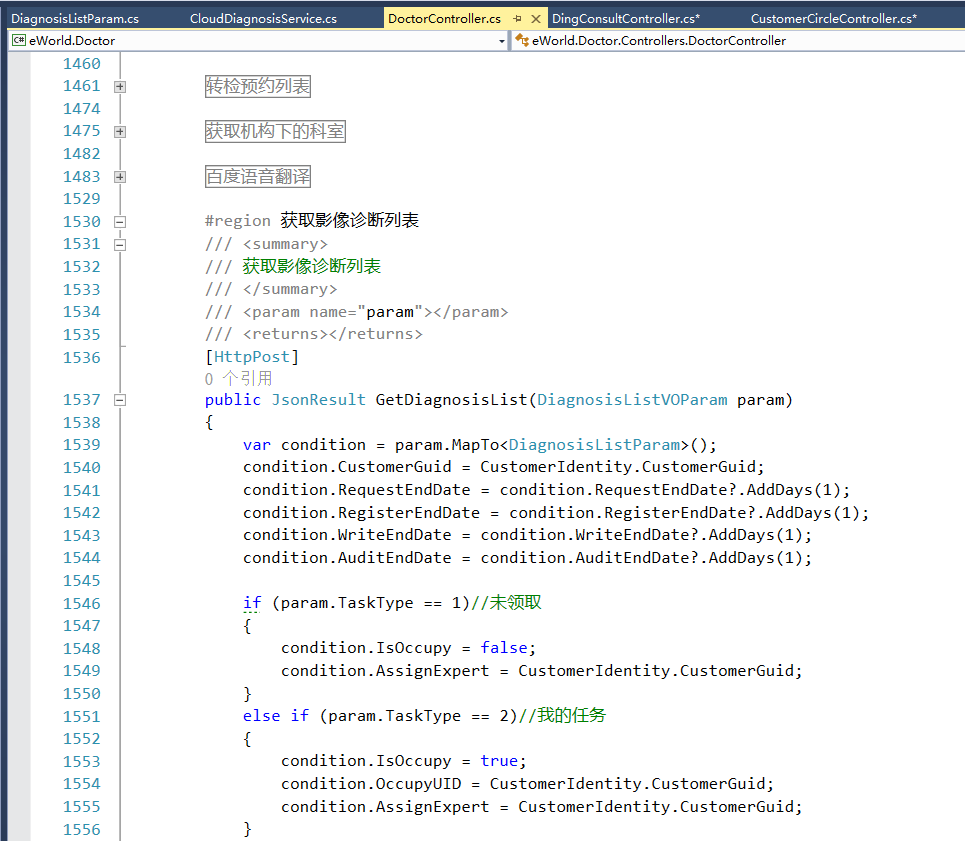
11.

以后分页抄这个。





在Doctor控制器中。



自己的总结，如果是Iqueryable类型，用彪哥的这种方式就很方便，但如果先要ToList()，就是先把数据全部查出来，再分页，就坑爹了。

12.

运维负责后台的加机构，加服务中心。加服务。这种。

13.



通过27和200的时间发现，原来27只是网络连到了202，所以在里面更改的文件夹，其实就是直接在202上面操作，最终还是部署在202的。

最终200发布的外网地址是：<http://218.108.111.166:9503>

通过查看，编辑网站绑定中的ip地址为全部未分配，这就内外网，局域网都可以访问了，也印证了我这里本机可以访问，局域网可以访问，阳湛局域网可以访问，张晨萌那里外网也可以访问。

查找：

最好是选择“全部未分配”，用这个选项时，服务器本机，在IE浏览器地址栏输入http：//localhosts/ 或127.0.0.1 可以打开本机架设的网站的主页，也可以输入内网IP地址打开内网的网站，做端口映射后，还可以从外网打开网站；就是说，选择这个选项，可以用所有的方法打开网站；

如果选择服务器的内网IP地址，就可以在局域网内部的其他电脑上输入服务器内网IP地址打开网站，不能在服务器本机上输入127.0.0.1打开网站，做端口映射后，也可以从外网打开网站；

如果设置为外网IP地址，就只能从外网打开网站，或者从内网以域名方式打开网站（需要端口映射）。不能从服务器本机上以127.0.0.1或内网IP地址方式打开网站。

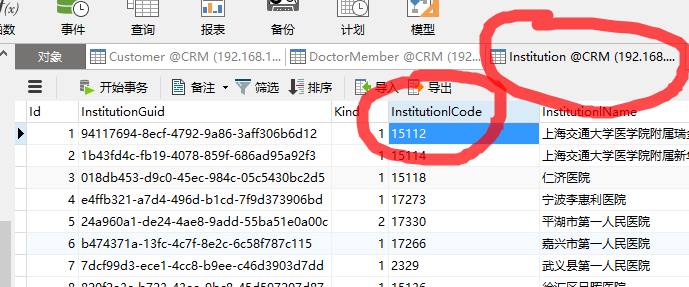
14.

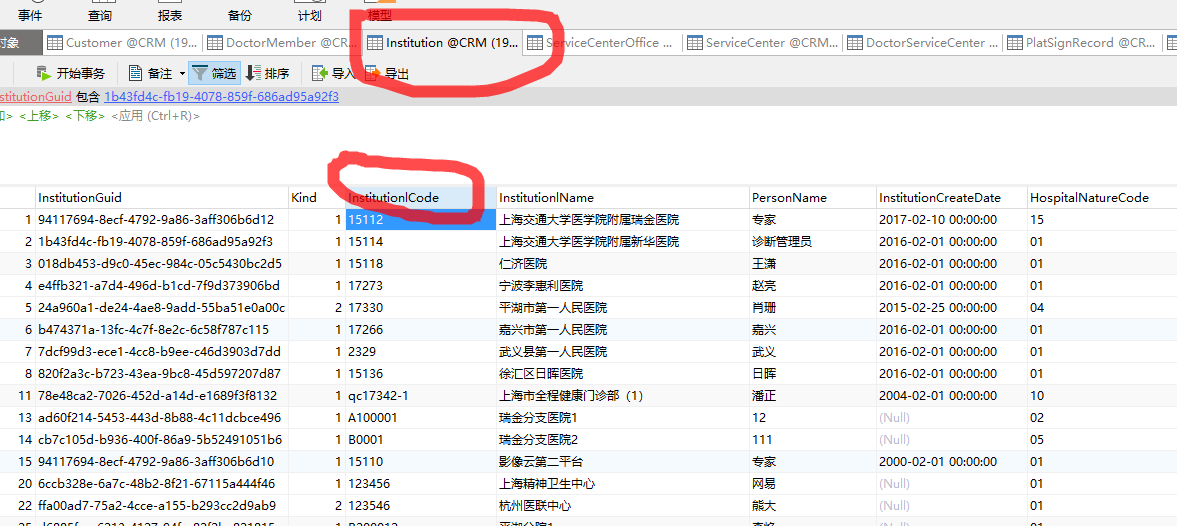
我直接在控制器这里，ui层调用第三方的接口，李阳说没问题的，不一定一定要放BLL层。

15.

会员医生的HospitalCode对应的其实是机构的InstitutionCode

结构再跟服务中心签约，多对一。然后，其实服务中心也是一个大的机构。





16.注入问题

（1）

//朋友圈服务

builder.RegisterType<CustomerCircleService>().As<ICustomerCircleService>().InstancePerHttpRequest();

这里的前面就是注入了，而InstancePerHttpRequest()表示：

InstancePerHttpRequest在一次Http请求上下文中,共享一个组件实例。仅适用于asp.net mvc开发。

（2）

构造函数注入：

namespace eWorld.WCFBLL.Joint.Crm

{

public class CrmManage : ICrmManage

{

#region 全局变量

private readonly IInstitutionService \_institutionService;

private readonly IDoctorMemberService \_doctorMemberService;

private readonly IServiceCenterService \_serviceCenterService;

private readonly IServiecOpensService \_serviceOpensService;

private readonly ICustomerService \_customerService;

private readonly IDataDicTypeService \_dataDicTypeService;

private readonly IDataDicInfoService \_dataDicInfoService;

private readonly IDiagnosticTemplateCatalogService \_diagnosticTemplateCatalogService;

private readonly IDiagnosticTemplateService \_diagnosticTemplateService;

private readonly IMemberCenterServiceService \_memberCenterServiceService;

private readonly IDoctorServiceService \_doctorServiceService;

private readonly IDoctorServiceCenterService \_doctorServiceCenterService;

private readonly IInstitutionOpenService \_institutionOpenService;

private readonly IApplyConsultService \_applyConsultService;

private readonly IServiceCenterManagerService \_serviceCenterManagerService;

private readonly ICustomerConfigService \_customerConfigService;

private readonly IAccessControlService \_accessControlServiceService;

private readonly IServiceCenterSetupService \_serviceCenterSetupService;

//收藏分类

private readonly IMyCatalogService \_myCatalogService;

//收藏

private readonly IMyCollectionService \_myCollectionService;

private readonly IEncryptionService \_encryptionService;

private readonly ICommonManage \_commonMange;

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

//远程医疗协同设置

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

private readonly ICloudDiagnosisConfigService \_cloudDiagnosisConfigService;

private readonly ICloudDiagnosisService \_cloudDiagnosisService;

private readonly IEcgRequestService \_ecgRequestService;

private readonly IFundusDiagnosisService \_fundusRequestService;

private readonly IRemoteClinicService \_remoteClinicService;

private readonly IMemberDoctorServiceService \_memberDoctorServiceService;

private readonly IMediaManage \_mediaManage;

private readonly IPermissionService \_permissionService;

private readonly ICustomerCircleService \_customerCircleService;

private readonly IQuestionService \_questionService;

private readonly ICustomerRegistrationService \_customerRegistrationService;

private readonly CustomerSettings \_customerSettings;

private static readonly ServiceManageSettings \_serviceSettings = EngineContext.Current.Resolve<ServiceManageSettings>(); //

#endregion

#region 构造函数

public CrmManage(IInstitutionService institutionService,

IDoctorMemberService doctorMemberService,

IServiceCenterService serviceCenterService,

IServiecOpensService serviceOpensService,

IDataDicTypeService dataDicTypeService,

IDataDicInfoService dataDicInfoService,

IDiagnosticTemplateCatalogService diagnosticTemplateCatalogService,

IDiagnosticTemplateService diagnosticTemplateService,

IMemberCenterServiceService memberCenterServiceService,

IDoctorServiceService doctorServiceService,

IDoctorServiceCenterService doctorServiceCenterService,

IInstitutionOpenService institutionOpenService,

IApplyConsultService applyConsultService,

IServiceCenterManagerService serviceCenterManagerService,

ICustomerConfigService customerConfigService,

IAccessControlService accessControlServiceService,

IServiceCenterSetupService serviceCenterSetupService,

ICommonManage commonMange,

IMyCatalogService myCatalogService,

IMyCollectionService myCollectionService,

ICloudDiagnosisConfigService cloudDiagnosisConfigService,

ICloudDiagnosisService cloudDiagnosisService, IEcgRequestService ecgRequestService,

IFundusDiagnosisService fundusRequestService, IRemoteClinicService remoteClinicService,

IMemberDoctorServiceService memberDoctorServiceService,

ICustomerService customerService,

IMediaManage mediaManage,

IPermissionService permissionService,

ICustomerCircleService customerCircleService,

IQuestionService questionService,

ICustomerRegistrationService customerRegistrationService,

IEncryptionService encryptionService,

CustomerSettings customerSettings)

{

\_institutionService = institutionService;

\_doctorMemberService = doctorMemberService;

\_serviceCenterService = serviceCenterService;

\_serviceOpensService = serviceOpensService;

\_dataDicTypeService = dataDicTypeService;

\_dataDicInfoService = dataDicInfoService;

\_diagnosticTemplateCatalogService = diagnosticTemplateCatalogService;

\_diagnosticTemplateService = diagnosticTemplateService;

\_memberCenterServiceService = memberCenterServiceService;

\_doctorServiceService = doctorServiceService;

\_doctorServiceCenterService = doctorServiceCenterService;

\_institutionOpenService = institutionOpenService;

\_applyConsultService = applyConsultService;

\_serviceCenterManagerService = serviceCenterManagerService;

\_customerConfigService = customerConfigService;

\_accessControlServiceService = accessControlServiceService;

\_serviceCenterSetupService = serviceCenterSetupService;

\_commonMange = commonMange;

\_myCatalogService = myCatalogService;

\_myCollectionService = myCollectionService;

\_cloudDiagnosisConfigService = cloudDiagnosisConfigService;

\_cloudDiagnosisService = cloudDiagnosisService;

\_ecgRequestService = ecgRequestService;

\_fundusRequestService = fundusRequestService;

\_remoteClinicService = remoteClinicService;

\_memberDoctorServiceService = memberDoctorServiceService;

\_customerService = customerService;

\_mediaManage = mediaManage;

\_permissionService = permissionService;

\_customerCircleService = customerCircleService;

\_questionService = questionService;

\_customerRegistrationService = customerRegistrationService;

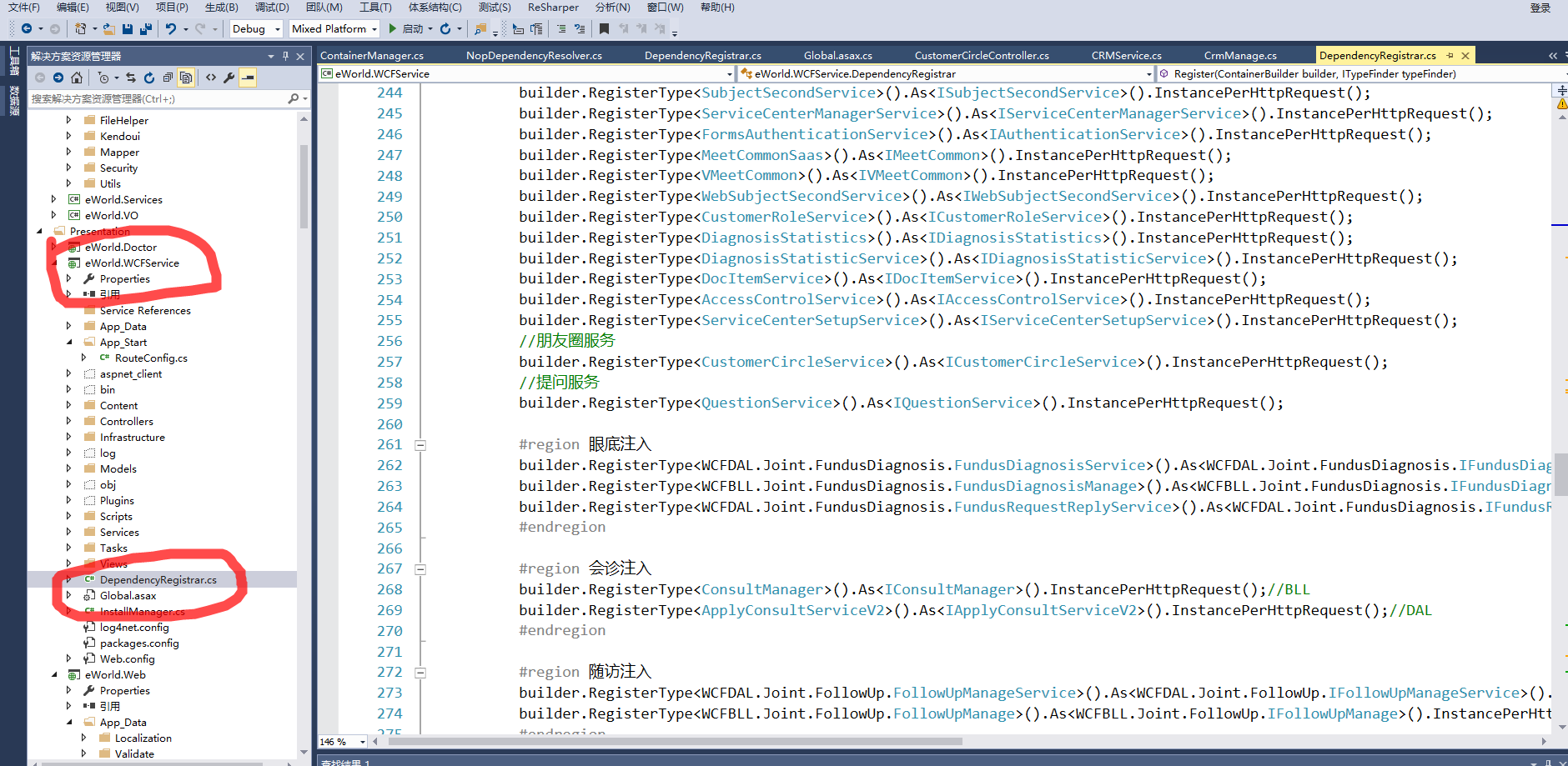
\_encryptionService = encryptionService;

\_customerSettings = customerSettings;

}

#endregion

注入过程：



17.

webapi的全部文件夹



wcf：





bin，Doctor，Views，这三个要替换，其余全都不变。

自己改了哪个，就另外替换哪个。

发现王毅这边前端改了，那Script也要改。

另外，content也要。

18.

吃饭说起，每个都要布这么多iis，协同三个，wcf，web，webapi，其他也是，common 要两个（公共的东西），其他crm，云归档都是。

总共要15个以上。

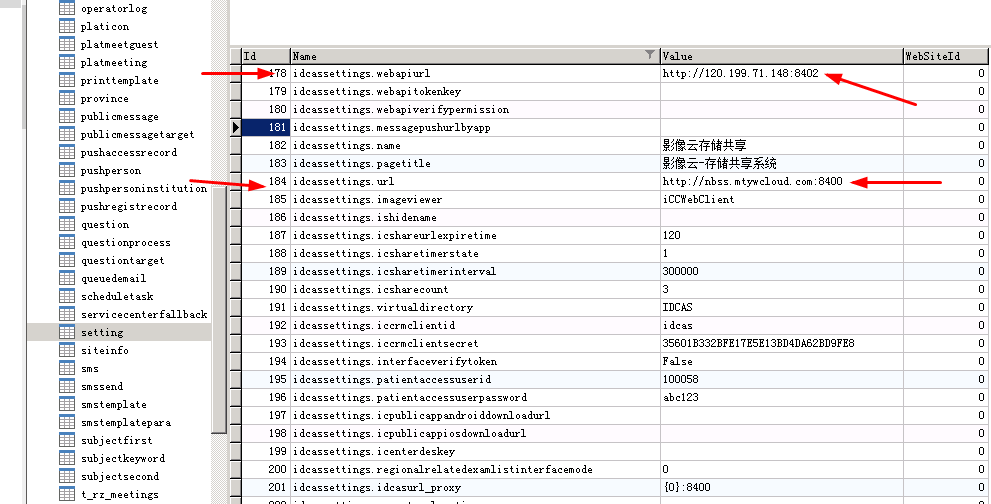
19.

开发环境的配置文件和77（测试环境）以及202（发布环境）的对比，就是配置数据库的地址在变，log4Net的配置不同。

后来的发布版本，现在通过调用军哥那边的初始化工具获取。

20.

果然最后的配置，是不同的医院配置不同的地址。



这是运维那边，可能邵逸夫那边的配置，web api的url，还有web的url都是根据实际情况去配置的。所以，在我这里的能力开放平台，我最后会用数据库配置中的url去替换当前的localhost，以便获取文件读取的真实的web地址。

21.