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。

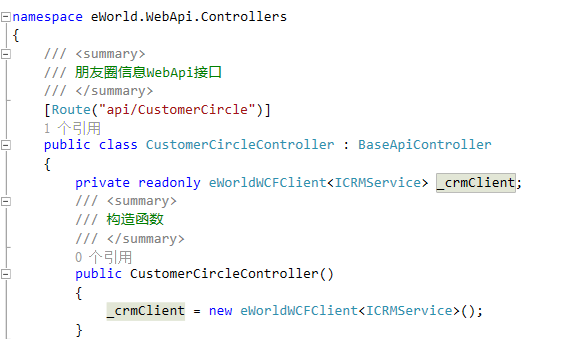
在实现中，统一用\_crmManage去操作方法。其实这些方法都是BLL层的方法。

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。

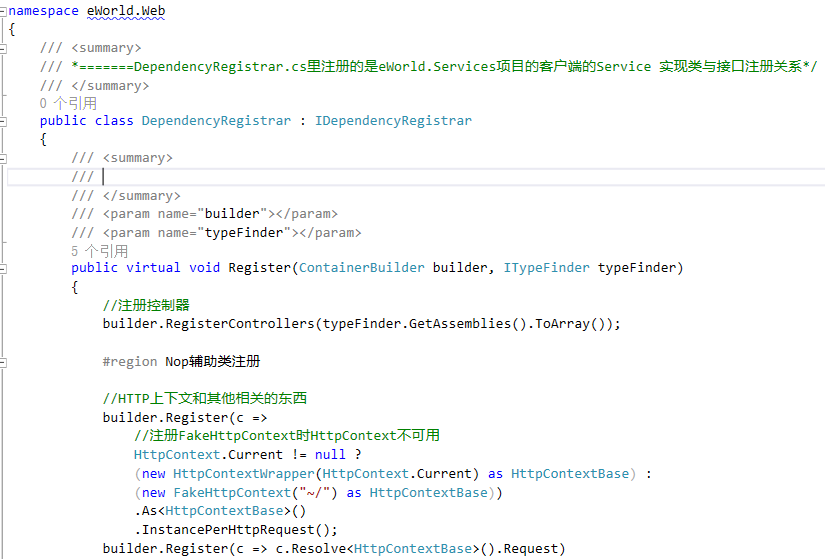




eWorld.WebApi中又一个注入的类。



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

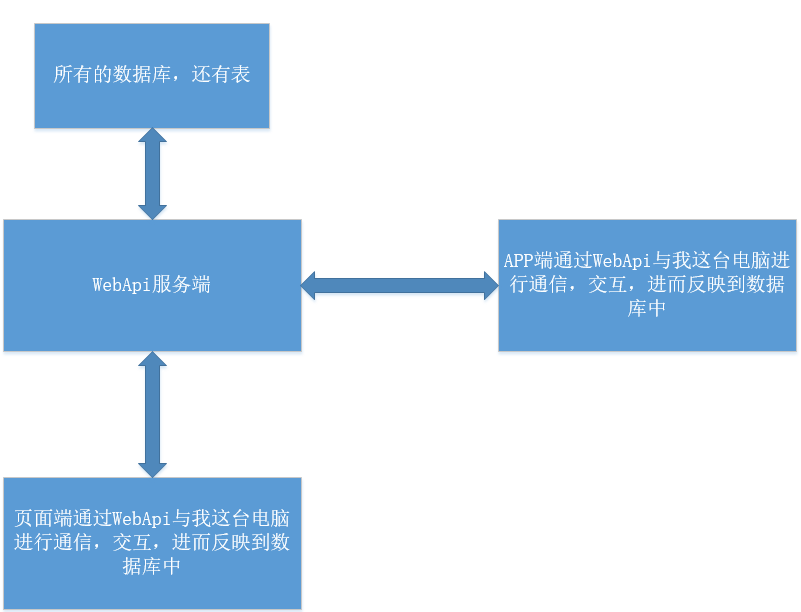
数据：数据库



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.

6.