DDD领域驱动设计初探（6）：领域服务 - 文章 - 伯乐在线

前言：之前一直在搭建项目架构的代码，有点偏离我们的主题（DDD）了，这篇我们继续来聊聊DDD里面另一个比较重要的知识点：领域服务。关于领域服务的使用，书中也介绍得比较晦涩，在此就根据博主自己的理解谈谈这个知识点的使用。

一、领域服务的引入

在《领域驱动设计：软件核心复杂性应对之道.Eric.Eva》这本书中，作者这样定义领域服务：有些重要的领域操作，不适合归到实体(Entity)和值对象(Value Object)的类别中，这些操作从本质上讲是有些活动或动作，而不是事物，当领域中的某个重要过程或转换操作不属于实体或值对象的自然职责时，应该在模型中添加一个作为独立接口的操作，并将其申明为Service。

Service是作为接口提供的一种操作，它在模型中是独立的，与实体和值对象不同的是，它强调的是与其他对象的关系，只定义能够为客户做什么。一般情况下，对于那些放在哪个实体和值对象里面都不合适，并且它更加偏向的是实体和实体之间的关系，这种场景下，我们就需要使用领域Service。看到这里，有人就郁闷了，既然领域服务的作用是处理实体与实体之间的关系，那么为什么不把它放在应用层里面去呢，应用层的作用不就是协调任务的吗？的确，理论上来讲，放在应用层也能够解决，但博主的理解是，DDD的设计原则之一是尽量丰满领域模型，主张充血的领域模型，也就是说和领域模型相关的领域逻辑最好是方法领域层，很显然，处理实体和实体之间的关系一般来说领域逻辑，所以最好还是放在领域层。当然，这个也并不绝对，只是博主自己的理解。

我们还是来举个例子说明，比如我们权限管理里面，如果需要一个功能，将指定的用户赋予制定的权限，比如接口这样定义

C#

1

```
void AssignPower(TB_USERS oUser, TB_ROLE oRole)
```

按照我们前面聚合的划分，这个接口是应该放在用户这个聚合里面还是角色聚合里面呢？很显然，感觉都不合适，这个接口包含两个聚合的实体，所以像这种情况，可以考虑用领域服务去解决。如果你非要说，方法不这样设计，放在应用层里面也是可以的。我只能说，呵呵，别闹。

二、领域服务的使用

还记得我们介绍仓储的时候我们领域层里面的Services文件夹么？下面，我们就根据给指定的用户赋予制定的权限的功能一步一步写一个领域服务功能。

1、聚合划分的修改

这里需要提出一个问题，对于博主在DDD领域驱动设计初探（1）：聚合这篇里面聚合的划分，现在想来存在不合理的地方，上次说了应该划分为4个聚合 ，可是没有考虑到用户和角色是多对多的关系，所以需要将用户角色表TB_USERROLE单独划分为一个聚合，所以总共是5个聚合，这也就是前面说的对于DDD里面聚合的划分是比较考量程序员经验的，对于聚合划分有误造成的误解表示抱歉。我需要在相应的地方做下修改。

领域实体要继承聚合根基类：

C#

```
        public interface IPowerManagerDomainService
        {
                void AssignPower(TB_USERS oUser, TB_ROLE oRole);
        }
        [Export(typeof(IPowerManagerDomainService))]
        public class PowerManagerDomainService:IPowerManagerDomainService
        {
                private IUserRepository _userRepository = null;
                private IRoleRepository _roleRepository = null;
                private IUserRoleRepository _userroleRepository = null;
                [ImportingConstructor]
                public PowerManagerDomainService(IUserRoleRepository oUserRoleRepository)
                {
                        _userroleRepository = oUserRoleRepository;
                }
                public void AssignPower(TB_USERS oUser, TB_ROLE oRole)
                {
                        if (oUser == null || oRole == null)
                        {
                                return;
                        }
                var oUserRole = _userroleRepository.Find(x => x.USER_ID == oUser.USER_ID &
x.ROLE_ID == oRole.ROLE_ID)                                                     .FirstOrDefault();
                        if (oUserRole == null)
                        {
                                oUserRole = new TB_USERROLE();
                                oUserRole.ROLE_ID = oRole.ROLE_ID;
                                oUserRole.USER_ID = oUser.USER_ID;
                                oUserRole.ID = Guid.NewGuid().ToString();
                                _userroleRepository.Insert(oUserRole);
                        }
                }
        }
```

在领域服务的实现类PowerManagerDomainService里面，我们使用了MEF的[ImportingConstructor]导入

构造函数特性，使用这个特性的前提是构造函数里面的参数类型IUserRoleRepository必须标注了导出Export，由前面我们看到IUserRoleRepository的实现类是标记了导出的，所以通过该特性就可以顺利将用户角色的仓储实现类的实例导入进来。

3、调用测试

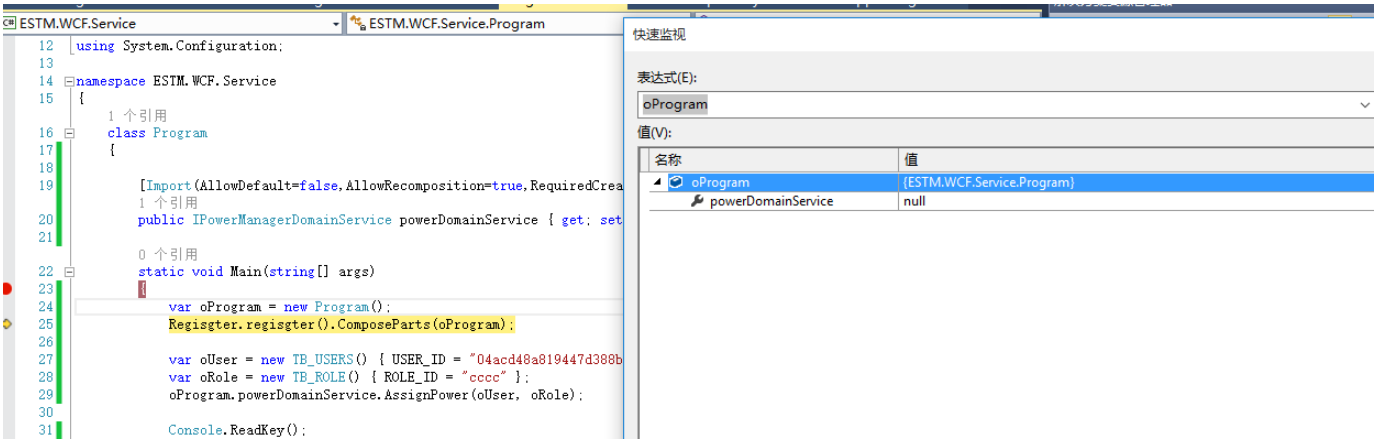
在应用层里面我们来写测试代码：

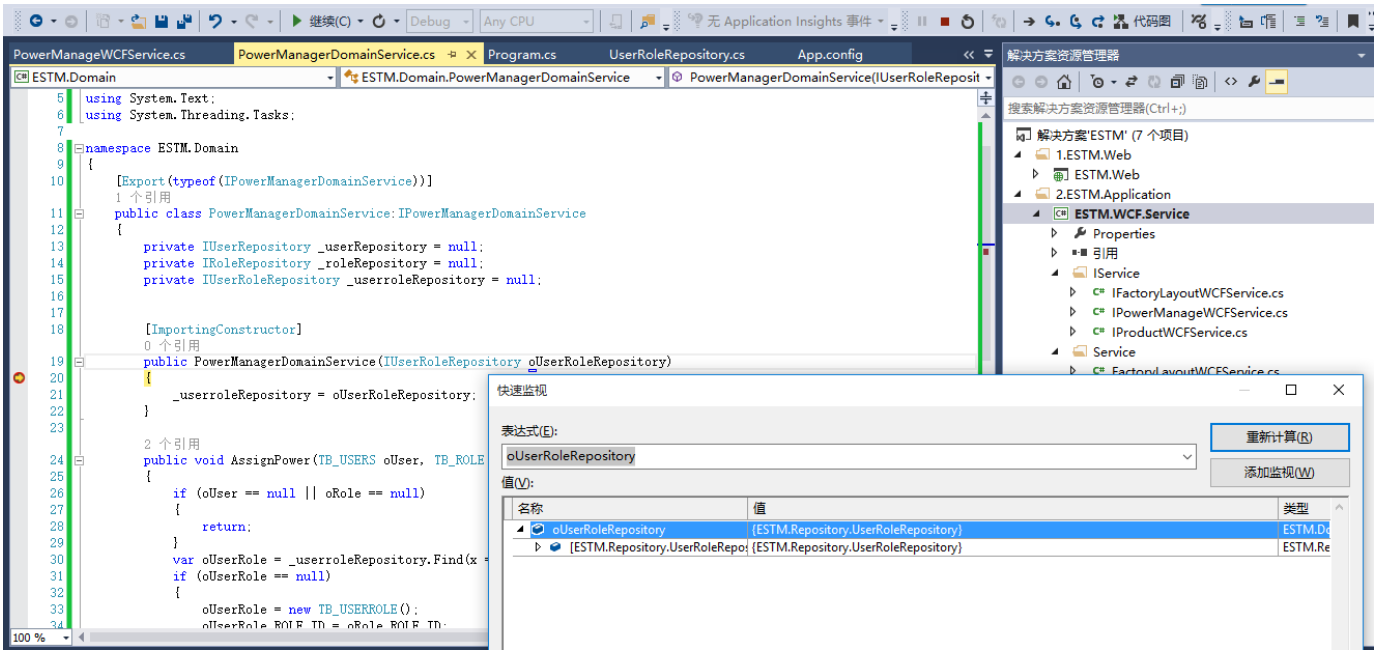
C#

```
    class Program
        {
                [Import(AllowDefault=false,AllowRecomposition=true,RequiredCreationPolicy=CreationPolicy.Any,Source=ImportSource.Any)]
                public IPowerManagerDomainService powerDomainService { get; set; }static void Main(string[] args)
                {
                        var oProgram = new Program();
                        Regisgter.regisgter().ComposeParts(oProgram);
                        var oUser = new TB_USERS() { USER_ID = "04acd48a819447d388b20dffb15f672e" };
                        var oRole = new TB_ROLE() { ROLE_ID = "cccc" };
                        oProgram.powerDomainService.AssignPower(oUser, oRole);
                        Console.ReadKey();
                }
        }
```

这里需要说明一点：虽然领域服务的实现类PowerManagerDomainService定义了一个有参的构造函数，如果不适用IOC注入的方式，我们通过new这个对象的时候需要传入仓储对象，但由于使用了构造函数的导入，参数通过构造函数的导入而传进去了，所以在应用层使用的时候也不用关心构造函数参数的问题了。这一点博主也是纠结了半天，后台调试程序才知道。是不是这样的，我们来看看：

我们在有参的构造函数里面打一个断点来看看





构造函数的参数就是这样传进去的。方法的执行都是很简单的逻辑。

4、总结

（1）MEF对于有参构造函数的导入要使用[ImportingConstructor]特性，参数类型要可导入。

（2）由于领域服务的接口和实现都是放在领域层里面，而且领域服务里面调用了仓储的实现逻辑，那么这样看领域层又和仓储的实现揉到一起了，这样是否又会造成领域层的不纯洁了呢？答案是不会，我们看领域服务实现类PowerManagerDomainService里面的代码可知，里面的逻辑都是通过仓储的接口对象IUserRoleRepository _userroleRepository去处理的，也就是说领域服务里面并没有调用具体的仓储实现，还是和仓储的接口在打交道，仓储的实现是在运行的时候通过MEF动态注入进去的。所以这样设计依然可以保持领域层的纯洁，不会依赖于具体的仓储实现。

（3）在上面的例子中，既然只用到IUserRoleRepository这一个仓储接口，那么这个功能是否可以直接放在用户角色的仓储实现里面而不用使用领域服务呢？刚开始博主也有这种疑惑，可是后来想想还是不行，因为对于权限模块，UI前端是不会和像DTO_TB_USERROLE这种对象打交道的，因为它里面只有用户ID和角色ID，对于UI来说没有实际的意义。所以UI里面只会传递用户和角色这种对象，而这两种对象隶属于不同的聚合，这种情况下最好还是使用领域服务了。当然你也可以将TB_USERROLE、TB_USERS、TB_ROLE这3个划分为一个聚合，把TB_USERROLE作为聚合根，另外两个当做实体，然后利用TB_USERROLE的导航属性来处理TB_USERS和TB_ROLE，这种做法理论上也行，但是博主觉得像导航属性这种东西很多项目考虑到效率问题可能会关闭掉，所以使用起来也有一定的风险。还是那句话，视情况而定，如果你的项目利用仓储基本能搞定需求，或者说你不想又引入一个什么领域服务的概念，你也可以遵照你的设计，这个都没有问题，毕竟最好的架构是适合项目的架构！

DDD领域驱动设计初探系列文章：