



链滴

从零开始领域驱动 - 划分代码层次

作者: [xiaodaojava](#)

原文链接: <https://ld246.com/article/1563896978369>

来源网站: 链滴

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



开头说两句

博客地址: <https://www.lixiang.red>

公众号: 程序员心情站

项目背景

不知从何时开始,业内慢慢开始有了领域驱动的声音,刚好公司有些新项目要做,就借着DDD的浪潮,新项目,新技术选型

学习调研时遇到的问题

国内关于DDD的书和视频真心不那么多,听说阿里盒马对DDD有着深入的使用,但也没什么资料流出,网为数不多的两本书,一本红皮书,像天书一样,啃了几天,啃不下去,还有本稍好一点,也不是太好理解,大致了一遍后还是有点蒙,还是动手写代码吧,边写边思考,边改进,在学习过程中主要有以下思考阶段

1. 在实际编码中,如果代码如何分层
 2. DDD是以业务为主,那么如果去划分业务上下界
 3. DDD提倡了一种专家和开发人员都能理解的语言,那么如何去把这种新语言沉淀下来
- 等等...

对领域驱动的理解

有句笑谈是,没有加个中间表解决不了的业务,如果有,就加两张. 虽说领域驱动是以业务为主,但是落叶根也是要建库建表的.那么我们新引入的领域驱动的概念就是加在数据库表和Manage层的中间件.

以小刀现在负责的用户模块为例,我们先业务建模,小刀是按经典的三户模型进行设计的:客户-用户-账户.

1. 客户

是指自然人,只要点了我们系统的,就是我们的客户,不管是看了一眼还是点了一下.

2. 用户

指不仅仅看了我们的系统,还和我们系统产生了业务数据

3. 账户

原指金额行业中用来为用户销账的账户,在我们系统中,小刀修改成通行证的概念.即,一个通行证,可以对多个用户.

举个例子就是:小刀我是一个客户,我手里面两个通行证,一个是公司APP的通行证,一个是公司后台管理系统的通行证. APP通行证下面有我的微信账号,手机账号. 管理系统通行证下面有A管理系统,B管理系统权限这样

那么按照这三户的业务,他们实际上是属于同一个领域,但是有三张数据库表. 对领域驱动的理解总结起来就是: 一个领域对应多张表. 外层(service , manager) 从dao层拿了数据之后,业务都要通过领域层来处理.

领域驱动对应的代码分层

为什么修改代码分层呢,那肯定是以前的不好才会改,那以前的怎么不好了呢,我们来一起看下以前的代分层

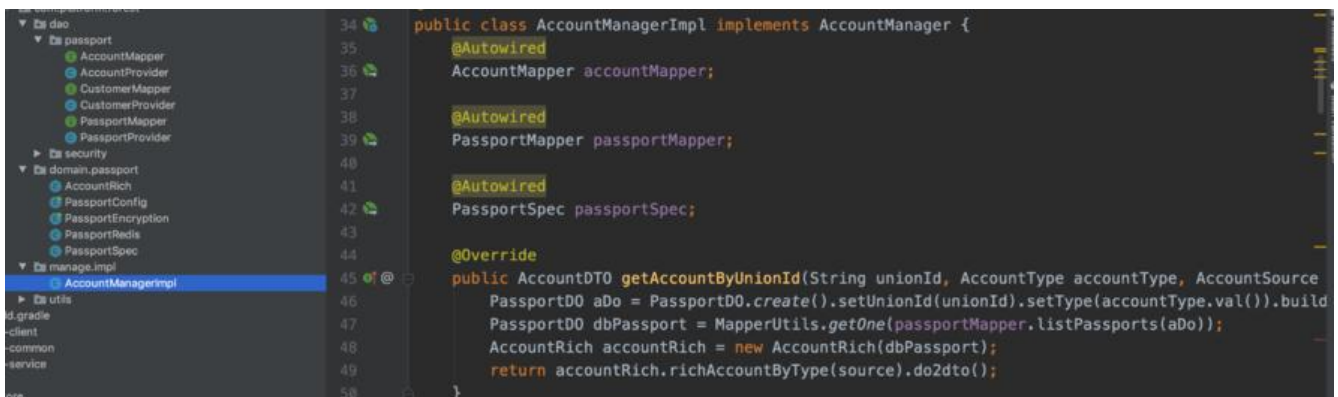
经典的代码分层结构

以前的经典分层是 controller 调用 manager/service , 然后service调用dao , dao层去数据库取数据, 后service里面对数据进行业务加工,然后返回给controller , controller 给包一个BaseResponse之类的加上状态码,信息之类的然后返回给调用方.

这样写也没什么不好的, 但有时候逻辑写的时候清楚,过段时间回来再看就无从下手了,最主要的原因是码没有根着业务走,只是拼出了业务想要的数据而已

新的代码分层结构

其实新也新不到哪去,下面这个分层结构是小刀结合看的书自己琢磨出来的,有不全之处,欢迎大家一起讨论



```
public class AccountManagerImpl implements AccountManager {
    @Autowired
    AccountMapper accountMapper;

    @Autowired
    PassportMapper passportMapper;

    @Autowired
    PassportSpec passportSpec;

    @Override
    public AccountDTO getAccountByUnionId(String unionId, AccountType accountType, AccountSource
        PassportDO aDo = PassportDO.create().setUnionId(unionId).setType(accountType.val()).build();
        PassportDO dbPassport = MapperUtils.getOne(passportMapper.listPassports(aDo));
        AccountRich accountRich = new AccountRich(dbPassport);
        return accountRich.richAccountByType(source).do2dto();
}
```

如上图所示,其实这里已经欠下了技术债(passport,account概念混乱了),但是代码一期已经上线了,jar也打出去了,不好再更改了.

DAO层

这层和以前代码一样,采用Mybatis实现的,只不过小刀这里用的是mybatis新的provider方式,没有用xm去写sql,这里其实有点小纠结,就是 redis/缓存 算不算DAO层的范围.因为我们也是从redis中去做数据存取.考量再三还是没有把redis归到DAO层,而是归到了领域层里面,因为redis里面都是业务数据

domain层

这一层是重点了,在这一层抽象出了四个主要的概念

1. 领域配置对象: PassportConfig

这里从apollo上获取一些项目业务配置信息

2. 领域规则对象: PassportSpec

这里面主要是对参数,验证码,节点值做检验,业务可以进行的,返回 true/false ,业务不能进行的,就抛出 RuntimeException

3. 领域缓存对象: PassportRedis

如名,对redis的操作和存取

4. 领域充血对象: AccountRich(现在想应该是PassportRich)

这里面封装了以上三个对象来进行业务操作

manage层

在这一层,如上图所示,每个方法都是大体这几步

1. 通过mapper从数据库取出业务实体对象

2. 把业务实体对象做为根传入充血对象

3. 对充血对象进行各种业务操作

4. 最后调用 2dto方法,返回一个值对象给调用方

最后说两句

领域驱动这块小刀还在不停的实践中,大家有什么想法和小刀一起讨论的可以加小刀微信: best39697502