**[plz叫我红领巾[IMG_256](https://juejin.im/book/5c90640c5188252d7941f5bb/section/5c9065385188252da6320022)](https://juejin.im/user/5ca038336fb9a05e790a3638" \t "https://juejin.im/post/_blank)**

2019年05月10日阅读 1596

关注

# **领域驱动设计之实战权限系统微服务**

做一个租户系统下的权限服务，接管用户的认证和授权，我们取名该服务为oneday-auth-server

### 写在前面

​ DDD（领域驱动设计）中涉及到几个概念，实体，值对象，聚合，限定上下文。本篇只涉及实践，概念讲解将放在下一篇，同时上一篇[为什么我们需要领域驱动设计](https://juejin.im/post/5ca16166e51d456708675612" \t "https://juejin.im/post/_blank)作为科普帖，大家可以在看完代码之后再回头理解一下，同时对比一下现有项目，知其然更要知其所以然，你经常遇到了什么问题，为什么DDD能够更好的解决软件负责的问题。

### 需求描述

​ 认证功能即登录功能，登录成功登录态的设定，登录失败的处理方式例如IP锁定，失败超过次数锁定等方式

​ 授权功能即对认证通过的用户，进行角色和权限授予，同时开启资源保护，未具备访问该资源权限的用户将无法访问。

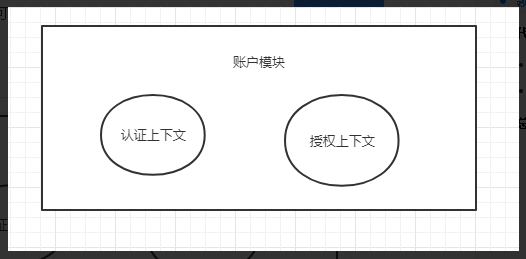
本篇将详细介绍如何在DDD的指导下实现第一点功能。

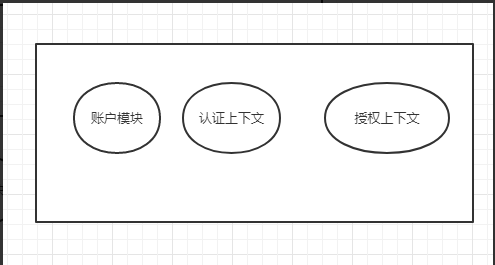
### 领域、子域和界限上下文

​ 我们先明白的一点是****领域****这个词语承载了太多的含义，既可以表示整个业务系统，也可以表示其中的某个核心域或者支撑子域。举个不是很恰当的例子，假设我们原本想要在一个叫账户模块实现了这个功能，同时还有用户信息功能，这个时候，****账户就是一个大的领域，一块的大蛋糕****，而oneday-auth则是这块大蛋糕的某一块，用户信息又是另一块，这被分出的一块一块蛋糕，我们称之为由账户领域分成的子域，****权限子域****和用户信息子域。子域下还可以再接着划分出子域，没有最小的子域，只有最合适的子域。

​ 你会觉得这个微服务的拆分很像，是的，微服务的拆分是遵循DDD的思想，但是你再仔细思考下，****你是不是只学了一个形式而已****可以对比一下下面的了两张图片和你的思路是不是不谋而合。

​





​ 本文中我将权限子域再划分出了****认证上下文和授权上下文****。对于界限上下文，我们把重点放在界限上，摘抄实现领域驱动设计的一段话：

比如，“顾客”这个术语可能有多种含义。在浏览产品目录的时候，“顾客”表示一种意思；而在下单的时候，“顾客”又表示另一种意思。原因在于：当浏览产品目录时，“顾客”被放在了先前购买情况、忠诚度、可买产品、折扣和物流方式这样的上下文中。而在上下单时，“顾客”的上下文包括名字、产品寄送地址、订单总价和一些付款术语

​ 我在oneday-auth中设计了一个类LoginUserE，用来代表登录用户实体类，包含的信息仅仅跟认证和授权相关，而用户信息子域中，肯定也有一个用户类UserInfo,但是这里的代表的含义是跟业务系统相关信息，比如说性别，昵称。我相信大多数读者肯定经历过一个类中承担过多功能，试图去创建一个全功能的类，最终导致的结果各位也可想而知，贪一时之方便带来的是不断拆东墙补西墙。

​ 用户进入认证界限上下文，他在这里只会被认为 一个待认证，而且只具备认证相关的信息，用户进入授权界限上下文，他在这里只会被认为一个认证成功，等待授权或者具备权限的用户。认证上下文和授权上下文我们可以

​ 于是在代码里，我划分了两个包模块：

> one.day.auth:

>> authentication ：认证即用户登录，身份识别等功能

>> authorization ：授权上下文：给予用户身份，角色，权限，并判断用户是否具备访问某个功能的权限等功能复制代码

​ 看到这里，请读者自己思考一个问题，如果按照原来的做法，你会不会分出两个包，你的大致做法是不是如下

> one.day.auth.service

>> authenticationServiceImpl

>> authorizationServiceImpl复制代码

​ 如果你看到这里突然有了一种思维的自我斗争，甚至有一种恍然大悟的感觉，那么恭喜你，你已经开始培养了DDD的思维。

​ 小结：代码目录的不同，就从一开始决定了你的开发思维。传统的MVC分层注定无法真正有效的划分领域，从而实现面向对象开发

### 代码实践

#### 代码分层

​ [为什么我们需要领域驱动设计](https://juejin.im/post/5ca16166e51d456708675612" \t "https://juejin.im/post/_blank)提到了两个架构，四层架构和六边形架构（又称端口-适配器）。其中六边形架构是从四层架构进一步发来而来的，是逻辑意义上的，代码的物理分层是做不到所谓六边形的。我们暂时抛开这一切，只关注我们想要的目的。

领域对象要做到只关心业务逻辑，不能出现丝毫技术细节，即不直接依赖任何外部，通过接口去依赖

​ 应用层：非业务相关处理；领域层：业务相关处理；基础设施：持久化，缓存等技术细节实现。代码目录分层如下：

> one.day.auth.authentication

> > app 应用层

> > client 二方包，这里方便起见放在了同一个Maven项目中

> > domain 领域层

> > > entity 实体包，具备行为，不具备数据状态

> > > port 端口定义，外部依赖统一定义为端口

> > > service 领域服务

> > infrastructure 基础设施层

> > > adapt 适配器，实现领域层定义的端口接口

> > > converter DTO，DO，Entity互相转换的工具类

> > > dataobject 表映射包 不具备行为，具备数据状态

> > > repository 仓储

> > > tunnel 通道复制代码

#### 功能实现

​ 我们来看看登录这一个功能具体是如何实现的。

**@Componentpublic** **class** **AuthenticationApp** {

/\*\*

\* 领域层，登录领域服务

\*/

**@Autowired**

**private** LoginService loginService;

/\*\*

\* 登录

\*

\* @param loginCmd

\*/

**public** **void** **login**(LoginCmd loginCmd) {

//调用领域层进行登录校验

String userId = loginService.login(loginCmd);

//session中存放userId已证明登录

//由于领域层主要负责登录，或者校验密码，登录成功之后的登录态设定不关心，交由应用层负责

ProjectUtil.setSession("userId", userId);

}

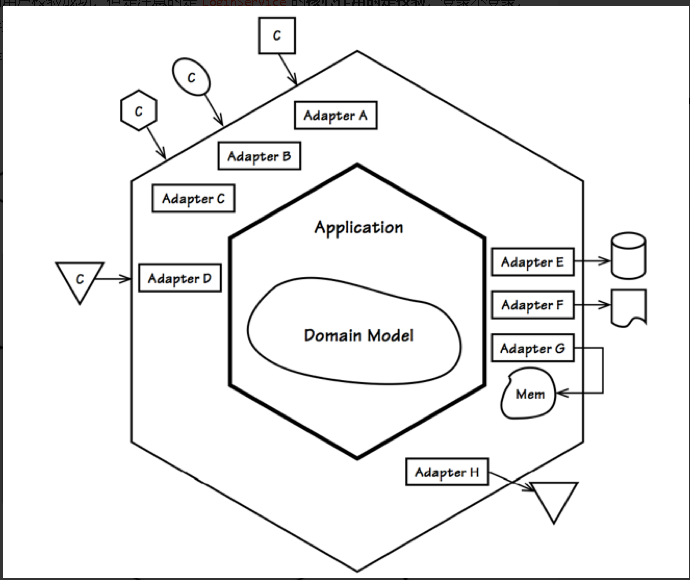
**public** **void** **addLoginUser**(AddLoginUserCmd addLoginUserCmd) {

loginService.addLoginUser(addLoginUserCmd);

}

}复制代码

​ 我们可以看到，应用层AuthenticationApp先调用了领域层的领域服务LoginService，当该方法没有抛出异常则证明用户校验成功，但是注意的是LoginService的****核心作用的是校验****，登录不登录，即登录态的设定并不是他所关心的，并不是他的业务逻辑。领域层只保证用户和密码是正确的，而其他一切东西都是外围，应用层，甚至是上游服务得知校验成功之后再来设定登录态。



​ 我们接着看看领域层，领域服务是如何工作的。

​ 我们先介绍两个类，LoginUserRepositoryPort和LoginUserConverter。读者可能会有一个疑惑是，怎么可能会没有技术细节呢，我怎样都需要将数据保存到数据库中，这肯定就涉及到持久化技术，这个时候六边形架构就应运而生了。我们的口号是“领域层不掺杂任何技术细节”，任何的外部依赖，我们都定义成一个端口类，而具体的实现交由各个层的适配器去实现，通过依赖注入实现相应的依赖功能。如何检验这一点，就是要看你的领域层能不能做到****拷贝不走样****，即如果你单纯复制domain目录到其他的项目中，是否能够正常编译。

​ LoginUserConverter存在的意义是什么，DTO，Entity，DataObject之间总会互相转换，将这一部分代码统一放到Converter类中。我相信读者的不少项目，各种转换都是很随意的，开心就好：)

**@Servicepublic** **class** **LoginServiceImpl** **implements** **LoginService** {

**private** **final** LoginUserRepositoryPort loginUserRepositoryPort;

**private** **final** LoginUserConverter loginUserConverter;

**@Autowired**

**public** **LoginServiceImpl**(LoginUserRepositoryPort loginUserRepositoryPort, LoginUserConverter loginUserConverter) {

**this**.loginUserRepositoryPort = loginUserRepositoryPort;

**this**.loginUserConverter = loginUserConverter;

}

**@Override**

**public** String **login**(LoginCmd loginCmd) {

Optional<LoginUserE> optionalLoginUserE = loginUserRepositoryPort.findByUsername(loginCmd.getUsername());

optionalLoginUserE.orElseThrow(() -> **new** BaseException(GlobalEnum.NON\_EXIST));

LoginUserE loginUserE = optionalLoginUserE.get();

loginUserE.login(loginCmd.getPassword());

//todo 登录成功，异步通知观察者

**return** loginUserE.getUserId();

}

**@Override**

**public** **void** **addLoginUser**(AddLoginUserCmd addLoginUserCmd) {

LoginUserE loginUserE = loginUserConverter.convert2Entity(addLoginUserCmd);

loginUserE.prepareToAdd();

loginUserRepositoryPort.add(loginUserE);

}

}复制代码

​ 领域服务LoginServiceImpl的第一件事是通过依赖注入获取的LoginUserRepositoryPort去查询获取登录用户LoginUserE，如果存在则调用login方法。我们看看LoginUserE究竟是什么玩意。

**@Datapublic** **class** **LoginUserE** **extends** **Unique** {

**public** **static** **final** String COMMON\_SALT = "commonSalt";

/\*\*

\* 登录用户名

\*/

**private** String username;

/\*\*

\* 登录密码

\*/

**private** String password;

/\*\*

\* 盐

\*/

**private** String salt;

/\*\*

\* 加密算法

\*/

**private** EncryptionAlgorithmV encryptionAlgorithmV;

/\*\*

\* 业务唯一ID

\*/

**private** String userId;

**private** TenantIdV tenantIdV;

/\*\*

\* 比较密码

\*

\* @param sendPwd 传入的密码

\* @return true/false

\*/

**public** **boolean** **login**(String sendPwd) {

//检查available

//错误次数限制

//锁号 ip

**return** StringUtils.equals(password, encryptionAlgorithmV.getPasswordEncoder().encoder(sendPwd, salt));

}

/\*\*

\* 密码加密

\*/

**public** **void** **encryptPassword**() {

**this**.setSalt(RandomStringUtils.randomNumeric(8));

**this**.setPassword(encryptionAlgorithmV.getPasswordEncoder().encoder(password, salt));

}

}复制代码

​ 代码逻辑其实很简单，留着几个扩展功能没有实现，一个是针对登录失败的各种场景操作，第二个是，对不同的租户下的用户系统实现不同的加密器。功能从上帝Service类转移到具备真正意义的实体类上，具备真正的行为，符合类的单一职责标准。

​ 到这里登录功能讲解就算是结束，但其中我留有一个功能未开发，即****登录成功，异步通知观察者****，DDD中同时倡导事件驱动开发和最终一致性。这其实也是跟****类的单一职责原则****有关。在整个登录功能中，****校验****是第一步，校验成功紧接着是进行****授权****，两者是上下游关系，核心业务逻辑不应该写在一块，这在传统MVC项目中两者是绝对的耦合在一起。而采用事件驱动可以将两者分离，无论是异步或者同步，简单起见的话可以直接使用guava的EventBus。

​ 持久化层的设计和特点本篇暂不涉及，不可一步而就，事实上如果你还关心这一点的话则证明你还未能理解DDD。重点是业务逻辑，无技术细节。持久化只是一种存储技术，不要因为用了这一个技术反而被绑架了你的思路。

### 总结

​ 业务层执行非业务逻辑，领域层只执行业务逻辑，使用端口-适配器模式隔离外部依赖，检验的标准是拷贝不走样。第一步的界限上下文划分很关键。一开始的划分就决定了你是面向对象还是面向过程。不要被持久化技术绑架了我们的开发思路。我们的口号是“****领域层不掺杂任何技术细节****”，我们的目标是****真正的面向对象开发****，我们的理想是****永不加班****！！！'

​ 走过路过不要错误，您的点赞是支持我写作最好的动力

​ 源码地址：[github.com/iamlufy/one…](https://github.com/iamlufy/oneday-auth" \t "https://juejin.im/post/_blank)

​ 作者：[plz叫我红领巾](https://juejin.im/user/5ca038336fb9a05e790a3638" \t "https://juejin.im/post/_blank)

​ 出处：[juejin.im/post/5cd3d1…](https://juejin.im/post/5cd3d1a8f265da034c7042c6" \t "https://juejin.im/post/_blank)

  本博客欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接，否则保留追究法律责任的权利。