14 | 代码模型（下）：如何保证领域模型与代码模型的一致性？

2019-11-15 欧创新 来自北京

《DDD实战课》

你好，我是欧创新。

在 [第 12 讲] 中，我们了解了如何用事件风暴来构建领域模型，在构建领域模型的过程中，我们会提取出很多的领域对象，比如聚合、实体、命令和领域事件等。到了 [第 13 讲]，我们又根据 DDD 分层架构模型，建立了标准的微服务代码模型，为代码对象定义好了分层和目录结构。

那要想完成微服务的设计和落地，这之后其实还有一步，也就是我们今天的重点——将领域对象映射到微服务代码模型中。那为什么这一步如此重要呢？

DDD 强调先构建领域模型然后设计微服务，以保证领域模型和微服务的一体性，因此我们不能脱离领域模型来谈微服务的设计和落地。但在构建领域模型时，我们往往是站在业务视角的，并且有些领域对象还带着业务语言。我们还需要将领域模型作为微服务设计的输入，对领域对象进行设计和转换，让领域对象与代码对象建立映射关系。

接下来我们围绕今天的重点，详细来讲一讲。

领域对象的整理

完成微服务拆分后，领域模型的边界和领域对象就基本确定了。

我们第一个重要的工作就是，整理事件风暴过程中产生的各个领域对象，比如：聚合、实体、命令和领域事件等内容，将这些领域对象和业务行为记录到下面的表格中。

你可以看到，这张表格里包含了：领域模型、聚合、领域对象和领域类型四个维度。一个领域模型会包含多个聚合，一个聚合包含多个领域对象，每个领域对象都有自己的领域类型。领域类型主要标识领域对象的属性，比如：聚合根、实体、命令和领域事件等类型。

领域模型	聚合	领域对象	领域类型
个人客户	个人客户	个人客户	聚合根
		创建个人信息	命令
		修改个人信息	命令
		查询个人信息	命令
		客户已创建	领域事件
		地址	实体
		新增地址	命令
		修改地址	命令
	客户归并	待归并客户	实体
		创建归并客户清单	命令
		归并客户	命令
		客户已归并	领域事件
		拆分客户	命令

从领域模型到微服务的设计

从领域模型到微服务落地，我们还需要做进一步的设计和分析。事件风暴中提取的领域对象，还需要经过用户故事或领域故事分析，以及微服务设计，才能用于微服务系统开发。

这个过程会比事件风暴来的更深入和细致。主要关注内容如下：

分析微服务内有哪些服务？

服务所在的分层？

应用服务由哪些服务组合和编排完成？

领域服务包括哪些实体的业务逻辑？

采用充血模型的实体有哪些属性和方法？

有哪些值对象？

哪个实体是聚合根等？

最后梳理出所有的领域对象和它们之间的依赖关系，我们会给每个领域对象设计对应的代码对象，定义它们所在的软件包和代码目录。

这个设计过程建议参与的角色有：DDD 专家、架构师、设计人员和开发经理。

领域层的领域对象

事件风暴结束时，领域模型聚合内一般会有：聚合、实体、命令和领域事件等领域对象。在完成故事分析和微服务设计后，微服务的聚合内一般会有：聚合、聚合根、实体、值对象、领域事件、领域服务和仓储等领域对象。

下面我们就来看一下这些领域对象是怎么得来的？

1. 设计实体

大多数情况下，领域模型的业务实体与微服务的数据库实体是一一对应的。但某些领域模型的实体在微服务设计时，可能会被设计为多个数据实体，或者实体的某些属性被设计为值对象。

我们分析个人客户时，还需要有地址、电话和银行账号等实体，它们被聚合根引用，不容易在领域建模时发现，我们需要在微服务设计过程中识别和设计出来。

在分层架构里，实体采用充血模型，在实体类内实现实体的全部业务逻辑。这些不同的实体都有自己的方法和业务行为，比如地址实体有新增和修改地址的方法，银行账号实体有新增和修改银行账号的方法。

实体类放在领域层的 Entity 目录结构下。

2. 找出聚合根

聚合根来源于领域模型，在个人客户聚合里，个人客户这个实体是聚合根，它负责管理地址、电话以及银行账号的生命周期。个人客户聚合根通过工厂和仓储模式，实现聚合内地址、银行账号等实体和值对象数据的初始化和持久化。

聚合根是一种特殊的实体，它有自己的属性和方法。聚合根可以实现聚合之间的对象引用，还可以引用聚合内的所有实体。聚合根类放在代码模型的 Entity 目录结构下。聚合根有自己的实现方法，比如生成客户编码，新增和修改客户信息等方法。

3. 设计值对象

根据需要将某些实体的某些属性或属性集设计为值对象。值对象类放在代码模型的 Entity 目录结构下。在个人客户聚合中，客户拥有客户证件类型，它是以枚举值的形式存在，所以将它设计为值对象。

有些领域对象可以设计为值对象，也可以设计为实体，我们需要根据具体情况来分析。如果这个领域对象在其它聚合内维护生命周期，且在它依附的实体对象中只允许整体替换，我们就可以将它设计为值对象。如果这个对象是多条且需要基于它做查询统计，我建议将它设计为实体。

4. 设计领域事件

如果领域模型中领域事件会触发下一步的业务操作，我们就需要设计领域事件。首先确定领域事件发生在微服务内还是微服务之间。然后设计事件实体对象，事件的发布和订阅机制，以及事件的处理机制。判断是否需要引入事件总线或消息中间件。

在个人客户聚合中有客户已创建的领域事件，因此它有客户创建事件这个实体。

领域事件实体和处理类放在领域层的 Event 目录结构下。领域事件的发布和订阅类我建议放在应用层的 Event 目录结构下。

5. 设计领域服务

如果一个业务动作或行为跨多个实体，我们就需要设计领域服务。领域服务通过对多个实体和实体方法进行组合，完成核心业务逻辑。你可以认为领域服务是位于实体方法之上和应用服务之下的一层业务逻辑。

按照严格分层架构层的依赖关系，如果实体的方法需要暴露给应用层，它需要封装成领域服务后才可以被应用服务调用。所以如果有的实体方法需要被前端应用调用，我们会将它封装成领域服务，然后再封装为应用服务。

个人客户聚合根这个实体创建个人客户信息的方法，被封装为创建个人客户信息领域服务。然后再被封装为创建个人客户信息应用服务，向前端应用暴露。

领域服务类放在领域层的 Service 目录结构下。

6. 设计仓储

每一个聚合都有一个仓储，仓储主要用来完成数据查询和持久化操作。仓储包括仓储的接口和仓储实现，通过依赖倒置实现应用业务逻辑与数据库资源逻辑的解耦。

仓储代码放在领域层的 Repository 目录结构下。

应用层的领域对象

应用层的主要领域对象是应用服务和事件的发布以及订阅。

在事件风暴或领域故事分析时，我们往往会根据用户或系统发起的命令，来设计服务或实体方法。为了响应这个命令，我们需要分析和记录：

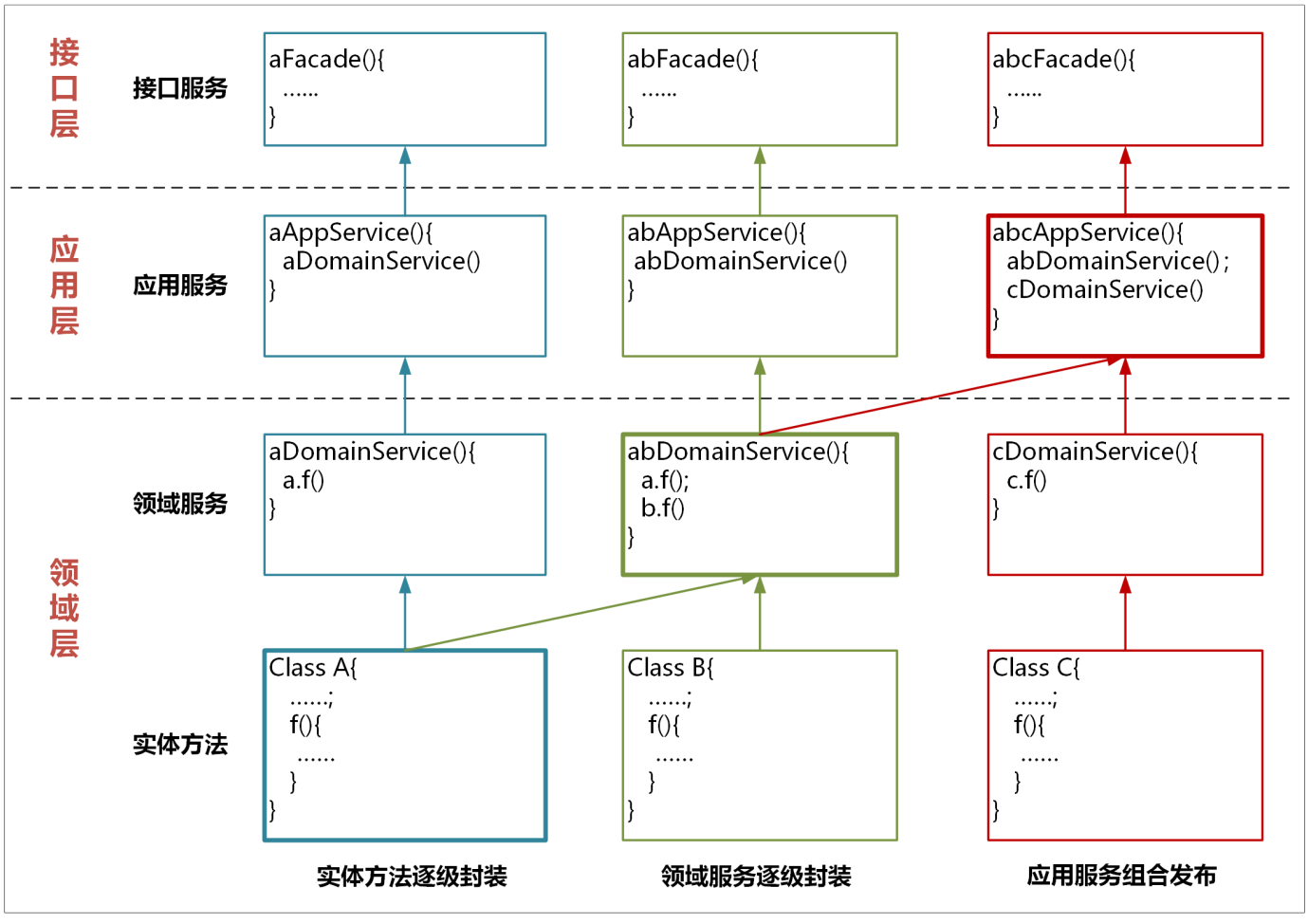
在应用层和领域层分别会发生哪些业务行为；

各层分别需要设计哪些服务或者方法；

这些方法和服务的分层以及领域类型（比如实体方法、领域服务和应用服务等），它们之间的调用和组合的依赖关系。

在严格分层架构模式下，不允许服务的跨层调用，每个服务只能调用它的下一层服务。服务从下到上依次为：实体方法、领域服务和应用服务。

如果需要实现服务的跨层调用，我们应该怎么办？我建议你采用服务逐层封装的方式。



我们看一下上面这张图，服务的封装和调用主要有以下几种方式。

1. 实体方法的封装

实体方法是最底层的原子业务逻辑。如果单一实体的方法需要被跨层调用，你可以将它封装成领域服务，这样封装的领域服务就可以被应用服务调用和编排了。如果它还需要被用户接口层调用，你还需要将这个领域服务封装成应用服务。经过逐层服务封装，实体方法就可以暴露给上面不同的层，实现跨层调用。

封装时服务前面的名字可以保持一致，你可以用 *DomainService 或 *AppService 后缀来区分领域服务或应用服务。

2. 领域服务的组合和封装

领域服务会对多个实体和实体方法进行组合和编排，供应用服务调用。如果它需要暴露给用户接口层，领域服务就需要封装成应用服务。

3. 应用服务的组合和编排

应用服务会对多个领域服务进行组合和编排，暴露给用户接口层，供前端应用调用。

在应用服务组合和编排时，你需要关注一个现象：多个应用服务可能会对多个同样的领域服务重复进行同样业务逻辑的组合和编排。当出现这种情况时，你就需要分析是不是领域服务可以整合了。你可以将这几个不断重复组合的领域服务，合并到一个领域服务中实现。这样既省去了应用服务的反复编排，也实现了服务的演进。这样领域模型将会越来越精炼，更能适应业务的要求。

应用服务类放在应用层 Service 目录结构下。领域事件的发布和订阅类放在应用层 Event 目录结构下。

领域对象与微服务代码对象的映射

在完成上面的分析和设计后，我们就可以建立像下图一样的，领域对象与微服务代码对象的映射关系了。

典型的领域模型

个人客户领域模型中的个人客户聚合，就是典型的领域模型，从聚合内可以提取出多个实体和值对象以及它的聚合根。

我们看一下下面这个图，我们对个人客户聚合做了进一步的分析。提取了个人客户表单这个聚合根，形成了客户类型值对象，以及电话、地址、银行账号等实体，为实体方法和服务做了封装和分层，建立了领域对象的关联和依赖关系，还有仓储等设计。关键是这个过程，我们建立了领域对象与微服务代码对象的映射关系。

微服务	层	聚合	领域对象	领域类型	依赖的领域对象	包名	类名	方法名
个人客户	应用层	/	创建个人客户信息	应用服务	领域服务：创建个人客户信息	*.person.application.service	CreatePersonInfoAppService	CreatePersonInfoAppService
		/	修改个人客户信息	应用服务	领域服务：修改个人客户信息	*.person.application.service	UpdatePersonInfoAppService	UpdatePersonInfoAppService
		/	查询个人客户信息	应用服务	领域服务：查询个人客户信息	*.person.application.service	GetPersonInfoAppService	GetPersonInfoAppService
		/	客户已创建	事件发布	领域事件：客户已创建	*.person.application.event.publish	PersonEventPublish	
	领域层	个人客户	个人客户	聚合根		*.person.domain.person.entity	Person	
			创建个人客户信息	方法		*.person.domain.person.entity	Person	CreatePersonInfo
			修改个人客户信息	方法		*.person.domain.person.entity	Person	UpdatePersonInfo
			查询个人客户信息	方法		*.person.domain.person.entity	Person	GetPersonInfo
			生成客户编码	方法		*.person.domain.person.entity	Person	CreatePersonId
			客户证件类型	值对象	聚合根个人客户的值对象	*.person.domain.person.entity	IdType	
			电话	实体	被聚合根个人客户引用	*.person.domain.person.entity	PhoneNumber	
			客户创建事件	事件实体		*.person.domain.person.event	PersonEvent	
			客户已创建	领域事件		*.person.domain.person.event	PersonEvent	PersonEventProcess
			地址	实体	被聚合根个人客户引用	*.person.domain.person.entity	Address	
			新增地址	方法		*.person.domain.person.entity	Address	CreateAddress
			修改地址	方法		*.person.domain.person.entity	Address	UpdateAddress
			银行账号	实体	被聚合根个人客户引用	*.person.domain.person.entity	BankAccount	
			创建银行账号	方法		*.person.domain.person.entity	BankAccount	CreateBankaccount
			修改银行账号	方法		*.person.domain.person.entity	BankAccount	UpdateBankaccount
			创建个人客户信息	领域服务	方法：创建个人客户信息	*.person.domain.person.service	PersonService	CreatePersonInfoDomService
			修改个人客户信息	领域服务	方法：修改个人客户信息	*.person.domain.person.service	PersonService	UpdatePersonInfoDomService
			查询个人客户信息	领域服务	方法：查询个人客户信息	*.person.domain.person.service	PersonService	GetPersonInfoDomService
			个人客户仓储接口	仓储接口		*.person.domain.person.repository.facade	PersonRepositoryInterface	PersonRepositoryInterface
			个人客户仓储实现	仓储实现		*.person.domain.person.repository.persistence	PersonRepositoryImpl	PersonRepositoryImpl

下面我对表格的各栏做一个简要的说明。

层：定义领域对象位于分层架构中的哪一层，比如：接口层、应用层、领域层以及基础层等。

领域对象：领域模型中领域对象的具体名称。

领域类型：根据 DDD 知识体系定义的领域对象的类型，包括：限界上下文、聚合、聚合根、实体、值对象、领域事件、应用服务、领域服务和仓储服务等领域类型。

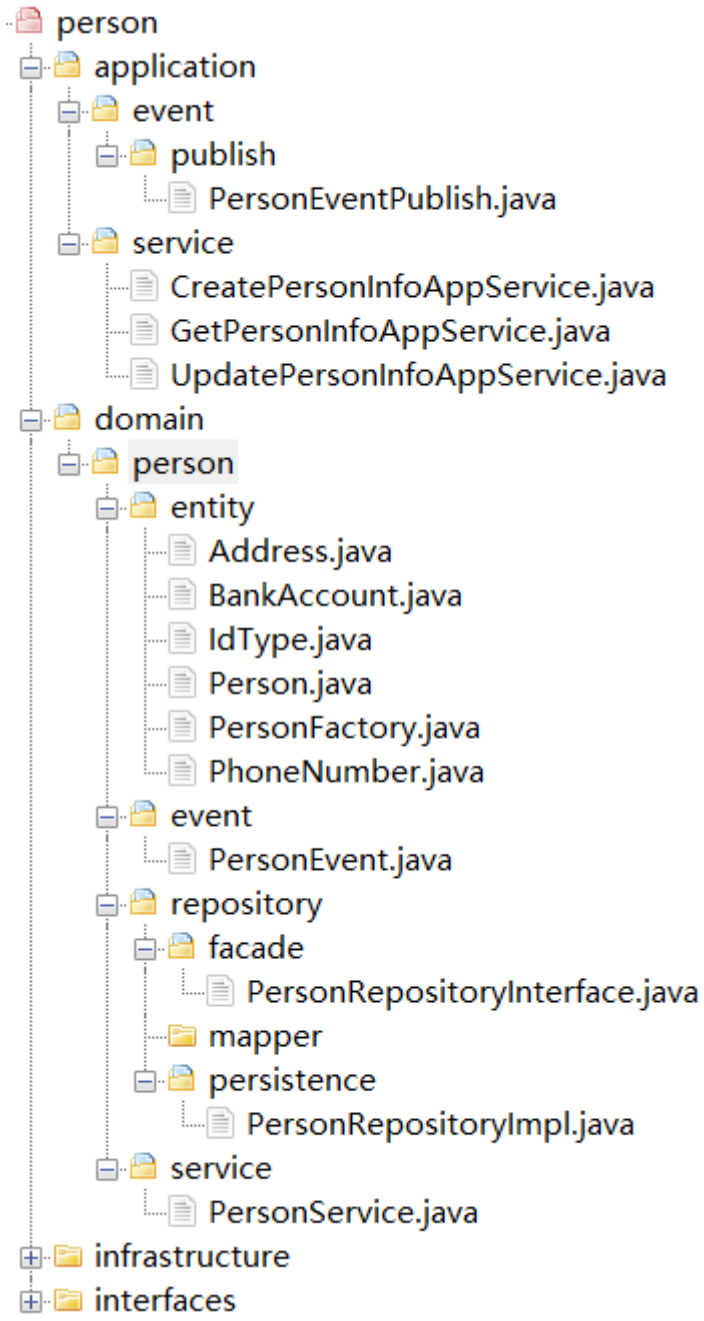
依赖的领域对象：根据业务对象依赖或分层调用的依赖关系，建立的领域对象的依赖关系，比如：服务调用依赖、关联对象聚合等。

包名：代码模型中的包名，对应领域对象所在的软件包。

类名：代码模型中的类名，对应领域对象的类名。

方法名：代码模型中的方法名，对应领域对象实现或操作的方法名。

在建立这种映射关系后，我们就可以得到如下图的微服务代码结构了。



非典型领域模型

有些业务场景可能并不能如你所愿，你可能无法设计出典型的领域模型。这类业务中有多个实体，实体之间相互独立，是松耦合的关系，这些实体主要参与分析或者计算，你找不出聚合根，但就业务本身来说它们是高内聚的。而它们所组合的业务与其它聚合是在一个限界上下文内，你也不大可能将它单独设计为一个微服务。

这种业务场景其实很常见。比如，在个人客户领域模型内有客户归并的聚合，它扫描所有客户，按照身份证号码、电话号码等是否重复的业务规则，判断是否是重复的客户，然后对重复的客户进行归并。这种业务场景你就找不到聚合根。

那对于这类非典型模型，我们怎么办？

我们还是可以借鉴聚合的思想，仍然用聚合来定义这部分功能，并采用与典型领域模型同样的分析方法，建立实体的属性和方法，对方法和服务进行封装和分层设计，设计仓储，建立领域对象之间的依赖关系。唯一可惜的就是我们依然找不到聚合根，不过也没关系，除了聚合根管理功能外，我们还可以用 DDD 的其它设计方法。

总结

今天我们学习了从领域模型到微服务的设计过程，这个过程在微服务设计过程中非常的关键。你需要从微服务系统的角度，对领域模型做深入、细致的分析，为领域对象分层，找出各个领域对象的依赖关系，建立领域对象与微服务代码对象的映射关系，从而保证领域模型与代码模型的一致性，最终完成微服务的设计。

在建立这种业务模型与微服务系统架构的关系后，整个项目团队就可以在统一的通用语言下工作，即使不熟悉业务的开发人员，或者不熟悉代码的业务人员，也可以很快就定位到代码位置。

思考题

分析一下基于 DDD 领域模型的微服务设计方式，和你公司现在所进行的微服务设计，或者和你了解到的微服务设计，有什么不同？

期待你的分享，我们一同交流！

精选留言 (70)

伊来温

2020-07-02

我的回复怎么不见了，再发一下。请教下老师，关于领域代码的分层和编排上一直以来我有个一个疑问。假设我有两个领域聚合跟，用户(User)和企业(Corp)，对应的领域服务是UserDomi

anService和CorpDomainService，那我假如需要这个一个接口listCoprUser来获取企业下面的用户列表， 这个接口该放在哪一层做编排呢。1. 如果放在CorpDomainService里面，则会造成对User实体的引用，造成耦合。2.难道上升到app层做编排么？但listCoprUser又像是一个领域服务。3.又或者做成一个新的领域服务CorpUserDomainService吗？那是不是CorpUserDomainService在代码结构上只有一个领域服务，而没有repository, domain层级了呢?

作者回复: 从你的场景来看，企业跟用户是一对多的关系吧。你可以这样设计，在企业聚合中将用户的相关信息设计为一个包含若干用户属性的用户值对象，然后企业聚合根引用用户值对象，用户值对象的数据来源于用户聚合。
用户聚合包含了全量的用户数据，而在企业聚合中用户值对象只是简单必须的用户数据，通过这种数据冗余的方式，在企业聚合中，就可以通过企业领域服务中一次获取企业和它相关的用户数据清单。

共 13 条评论 >　　22

Jack.Chen
2019-12-30

希望把完整样例代码放出来

作者回复: 这两天会加一篇完整的代码详解。

共 6 条评论 >　　16

冯磊
2020-06-29

感觉application这一层完全可以去掉，intrerface直接调domain service就可以的。作者能解释一下application这一层为什么必须存在吗?

作者回复: 应用层连接用户接口层和领域层，它是很薄的一层，主要职能是协调领域层多个聚合完成服务的组合和编排。
应用层之下是领域层，领域层是由多个业务职责单一的聚合构成，实现核心的领域逻辑。应用层负责协调领域层多个聚合的领域服务或领域对象，面向用例和业务流程完成服务的组合和编排。所以理论上应用层不应该实现领域模型的领域逻辑。这也是应用层为什么会很薄的原因。
应用层之上是用户接口层，在应用层完成领域层服务组合和编排后，应用服务被用户接口层Facade服务封装，完成接口和数据适配后，以粗粒度的服务通过API网关面向前端应用发布。
此外，应用层也是微服务之间服务调用的通道，微服务在应用层可以调用其他微服务的应用服务，完成微服务之间的服务组合和编排。
在应用层主要有应用服务、事件订阅和发布等相关代码逻辑。

其中，应用服务主要负责服务的组合、编排和转发，处理业务用例的执行顺序以及结果的拼装。在应用服务中还可以进行安全认证、权限校验、事务控制、领域事件发布或订阅等。

suke

2020-06-27

老师 请问依赖倒置是如何体现的？还有所谓的充血模式，最好能有具体的代码说明，不然还是觉得很空洞

作者回复: 依赖倒置的代码在加餐里面会有详细说明。

一、依赖倒置（DIP）设计：是指面向接口编程，而不是面向实现编程。这样可以避免业务逻辑与实现逻辑的耦合，在实现逻辑出现变化时，降低对业务逻辑的影响。

为了解耦领域逻辑和数据处理逻辑，我们在领域层和基础层之间增加了薄薄的一层，这一层就是仓储。

仓储模式包含仓储接口和仓储实现，仓储接口面向领域层提供基础层数据处理相关的访问接口，仓储实现完成仓储接口对应的数据持久化相关的逻辑处理。一个聚合会有一个仓储，统一由仓储来完成聚合数据的持久化。

领域层业务逻辑面向仓储接口编程，当聚合内的实体数据需要持久化时，只需将领域对象DO对象转换成PO持久化对象，然后传递给仓储接口，通过仓储实现完成DO数据的持久化工作。这样领域层就可以更好的聚焦于聚合的领域逻辑，而不必关心实体数据在基础层到底是如何实现持久化的了。

仓储接口的实现逻辑非常简单，只需要在仓储接口类中，定义仓储实现的基本接口和参数就可以了。

仓储接口代码如下：

```
public interface PersonRepository {
  void insert(PersonPO personPO);
  void update(PersonPO personPO);
  PersonPO findById(String personId);
  PersonPO findLeaderByPersonId(String personId);
}
```

仓储实现会根据仓储接口的数据处理逻辑要求，调用DAO完成数据查询或数据持久化，如基于聚合根ID的查询，聚合中新增或修改等领域对象数据的持久化操作。假如数据库需要技术升级，我们只需调整仓储实现的数据处理逻辑，适配新的数据库就可以了，这种调整不会影响领域逻辑。

仓储实现代码如下：

```
public class PersonRepositoryImpl implements PersonRepository {
      @Autowired
    PersonDao personDao;
    @Override
    public void insert(PersonPO personPO) {
```

```
        personDao.save(personPO);
    }
    @Override
    public void update(PersonPO personPO) {
        personDao.save(personPO);
    }
    @Override
    public PersonPO findById(String personId) {
        return personDao.findById(personId).orElseThrow(() -> new RuntimeException("未找到用
户"));
    }
    @Override
    public PersonPO findLeaderByPersonId(String personId) {
        return personDao.findLeaderByPersonId(personId);
    }
}
```

在领域服务中，可以调用仓储接口完成数据持久化操作。由于领域服务只与仓储接口发生调用关系，数据的持久化逻辑在仓储实现中完成。因此在更换数据库时，只要仓储接口不变，领域服务的逻辑就可以一直保持不变。

领域服务如下：

```
public class PersonDomainService {
    @Autowired
    PersonRepository personRepository;
    public void update(Person person) {
        personRepository.update(personFactory.createPersonPO(person));
    }
}
```

这样就保持了领域层领域逻辑的稳定，实现了领域层与基础层的解耦和依赖倒置。

二、充血模型与贫血模型的关键差异：

在充血模型中，业务逻辑都在领域实体对象中实现，实体本身不仅包含了属性，还包含了它的业务行为。DDD领域模型中实体是一个具有业务行为和逻辑的对象。

而在贫血模型中领域对象大多只有setter和getter方法，业务逻辑统一放在业务逻辑层实现，而不是在领域对象中实现。

1，对于实体采用充血模型，包含自己的属性及行为，例如保持、更新、删除等行为方法，需要持久化，依赖基础层数据库操作，是在实体直接引入，例如mybatis的mapper?
2，对于相对简单的实体操作增删改查这种，需要暴露到接口层；那要一层一层向上封装，实体》领域服务》应用服务》接口服务；这样是不是又显得代码很多余；一个简单的增加修改方法接口，需要很多冗余代码，上层也没有其他逻辑，封装一下调用下层，写一个接口，要写很多层次调用，是否会很臃肿啰嗦，是不是就可以直接接口层封装就省去一些层呢?
3，在服务编排上有没有一些框架什么的？还是都是通过if else的手写?

作者回复: 1、实体的这些数据库映射是通过mapper来实现的。
2、松散分层架构是可以跨层调用了，实现起来很容易。但是在复杂的情况下，服务不太容易管理，比如，你可能不知道你的方法到底被谁组合和封装了，一旦出现方法变更，你不容易一次找出所有受影响方。而逐层封装的话，你只需要逐层通知到上层就可以了。
3、微服务内的服务编排相对简单，就是业务逻辑的执行顺序而已，个人感觉不需要引入什么工具。

共 7 条评论 >

5

Jxin
2019-11-15

1.同求代码案例。（基于一个非ddd微服务的demo，分支形式实现微服务内部代码规范，跨服务间代码重组）

2.代码案例这个成本很大，但还是厚颜无耻的提了。毕竟缺少代码这个实体，这个专栏感觉就少点东西。毕竟讲得再抽象精准，可能也没有展示code来得直接明了。

3.我们需要从实悟虚，从虚就实。如果理论能结合code案例，这个专栏的学习成本和实用性将会有质得飞跃。

作者回复: 谢谢你的建议。后面准备准备，可能需要点时间。

5

吴海洋
2019-11-15

写得不错，通俗易懂。文章结构也符合我的阅读习惯。👍

4

日月星辰
2020-06-02

同一个微服务里不同领域之间的调用可以在应用层直接调用吗?

作者回复: 同一个微服务不同聚合之间为了解耦，不建议聚合之间直接调用。你可以将聚合之间的调用提升到应用层，通过应用服务来实现跨聚合的组合和调用。

4

Geek_deb968

2020-03-28

大部分业务场景其实都是查询的比较多，关于领域模型我现在看到的和理解到的都是实体简单业务操作，我十分希望能看到关于查询在DDD上代码是怎么实现的，比如门店是一个聚合根，门店菜系设计为值对象，那么我根据菜系查询门店是不是在领域模型上很难操作了，感觉领域模型都是在实体也就是满足确定了唯一标识的情况下，才能发挥作用，动态的查询在DDD上是需要怎样实现呢

作者回复: 复杂的查询一般都不走领域模型，一般这种查询你可以用原来的查询设计方式，或者采用读写分离方式。在DDD领域模型中主要是基于聚合根的id的查询。

共 3 条评论 >

3

Peter Yu

2020-11-26

老师，aDomainService何以调用bDomainService的方法吗。比如之前有个同学提问：Corp和User属于两个领域，但是Corp中有个查询user的服务，你建议他将此方法放在domainService层，那同步user的数据时，corpDomainService岂不是得调用userDomainService了?

作者回复: 聚合之间的领域服务是不建议相互调用的，这样聚合之间会产生耦合，不利于未来领域模型演进和聚合的拆分。聚合之间有两种协作模式，一种是领域事件驱动的模式，可以实现聚合之间数据的传输，另外一种是在应用层通过应用服务来组合和协调不同聚合的领域服务，完成跨聚合的调用和操作。

3

峰

2020-03-21

如何识别出聚合根?

作者回复: 一个参考是聚合中的关键实体，另外可以根据引用关系来判断，在所有具有引用关系的实体或值对象中，处于根位置的就是聚合根。

3

Jesen
2019-11-29

老师，如果把仓储Respository的实现放到基础设施层，其仓储接口定义在领域层里面，那么在领域层里面该怎么持久化呢，可以通过在应用层中将仓储注入到领域服务里面来实现吗?

作者回复: 这个只是代码存放目录的考虑。具体的持久化是在仓储实现的服务里面。为了方便聚合的重新组合，我在代码目录结构里面将仓储的接口和实现都放在领域层的聚合目录下，如果微服务架构演进，你可以直接将聚合相关的代码一起拿走。

2

徐李
2021-12-20

这样的一个微服务设计过程，在几十万，几百万行的代码系统中，不是要维护很大很大一个映射关系吗？基本上是每一个类，每一个操作，现在的接口都是要映射成DDD对应的实体，事件模型等对应名词。

1

Peter Yu
2021-07-19

老师，我在ddd实际中遇到一个非常困惑的问题。我有个服务是实现代码质量分析的。现在分析数据在表里都有了，主要有一张DailyMeasure表，记录date（日期），author（开发人员），metric（度量单位：如提交数commits、有效代码行书validCodeLines、sonar问题数issues等），value（度量的值）。【相应的领域对象也是这些字段】。

现在我有一个报表功能，主要是分页展示每个用户指定日期时段内的度量数据AuthorMeasures（authorId、duplicatonLines、validCodeLines、validCommentLines、issues、commits）。这个对象没有任何业务，只要聚合DailyMeasure表的数据返回给前端即可。

按传统写法只要写个sql封装成AuthorMeasuresDto就回传给前端可以了。但按照ddd思想，还得要建do、po，然后在各个层转化一遍，多了许多刻板流程（repositoy->po->factory->do->domainSerive->applicationService->assembler->dto）。我想问的是，对于没有业务逻辑

的查询，也得严格遵循ddd的代码范式吗，感觉非常冗余和臃肿。

望解惑！

共 4 条评论 >

1

Even He

2020-11-03

老师您好。想请教一个问题。
例子中，遵循了严格的分层方式，即不允许跨层调用。如appService需要调用entity的方法需要通过domainService。（虽然我觉得appService调entity也不算跨层，理由是application层和domain层中间没有其他的层。）
那interface中的assembler，它的职责是将dto转化成domain object。这个处理如果实在interface种执行，是不是变成了跨层调用？
期待您的回复。谢谢。

作者回复: appService是在应用层，entity的方法是在领域层，虽然它们都在同一个微服务，中间还有领域服务，因此属于从应用层到领域层的跨层调用。如果应用层直接访问entity的话，部分领域逻辑就会落到应用服务中，如果应用服务组合了多个聚合的实体，就容易使得聚合之间产生耦合，甚至直接对不同聚合实体对象的访问，会破坏领域层聚合内部的业务规则控制逻辑。记住，应用服务主要是完成服务组合和编排，以及协同的工作，基本不参与聚合内部的领域逻辑。
而interface中的assembler职责主要是完成对象之间的转换操作，这部分内容不涉及服务调用和业务逻辑处理。

1

一两

2020-10-26

老师，聚合或实体可以直接调用仓储接口吗？为什么网上有的说法是仓储接口应该在应用层调用，感觉应用层调用仓储接口的话，难免会把业务逻辑拆的四分五裂把

作者回复: 是的，仓储一般都是跟聚合是一对一的。应用层服务主要是做服务组合和编排，一般不会直接和数据打交道。但是也有特殊的情况，比如在应用层完成查询逻辑，或者跨聚合的复杂查询，而这些逻辑一般不会有领域模型的概念。

1

okjesse

2020-02-05

【如果一个业务动作或行为跨多个实体，我们就需要设计领域服务】，请问这个实体指的是一个聚合根下面的多个实体吗。 我看例子代码也是一个聚合根会有一个领域服务，会存在一个领域服务跨多个聚合根吗。

作者回复: 领域服务只在一个聚合内，一般不建议跨聚合的领域服务调用，这样会增加聚合之间的耦合度，不利于架构演进时微服务的再次拆分。
领域服务只是针对一个聚合内的实体。

1

蚂蚁内推+v
2020-01-29

想问下实体的增删改查是由实体来做还是领域服务来做？谁来调用仓储

作者回复: 实体自身的方法来完成增删改，复杂查询可以交由应用服务来做。仓储接口可以在应用服务或者领域服务中调用，也可以是聚合根的方法里。

1

山巅最小的费马质数颗...
2020-01-03

老师，如果存储是在实体类里面调用repository，那么我们要通过spring注入这些repository，但是实体理论上是每次new出来的新对象,那么我们就不能直接用@component啥的了，因为默认是单例的，这样我们岂不是每次创建实体都得自己调用applicationcotext.getbean来得到prototype的实体，或者把这个封装成工厂，所有创建实体都走工厂?

作者回复: 复杂聚合用工厂来实现吧。

1

ZIxuAN
2019-12-25

老师，领域层调用仓储添加实体，是在实体里面调用还是在领域服务中调用。实体中会涉及持久化操作吗

作者回复: 通过工厂来完成所有关联实体的初始化。一般都在领域服务里面来调工厂和仓储完成持久化。

1