13 | 代码模型（上）：如何使用DDD设计微服务代码模型?

2019-11-13 欧创新 来自北京

《DDD实战课》

你好，我是欧创新。

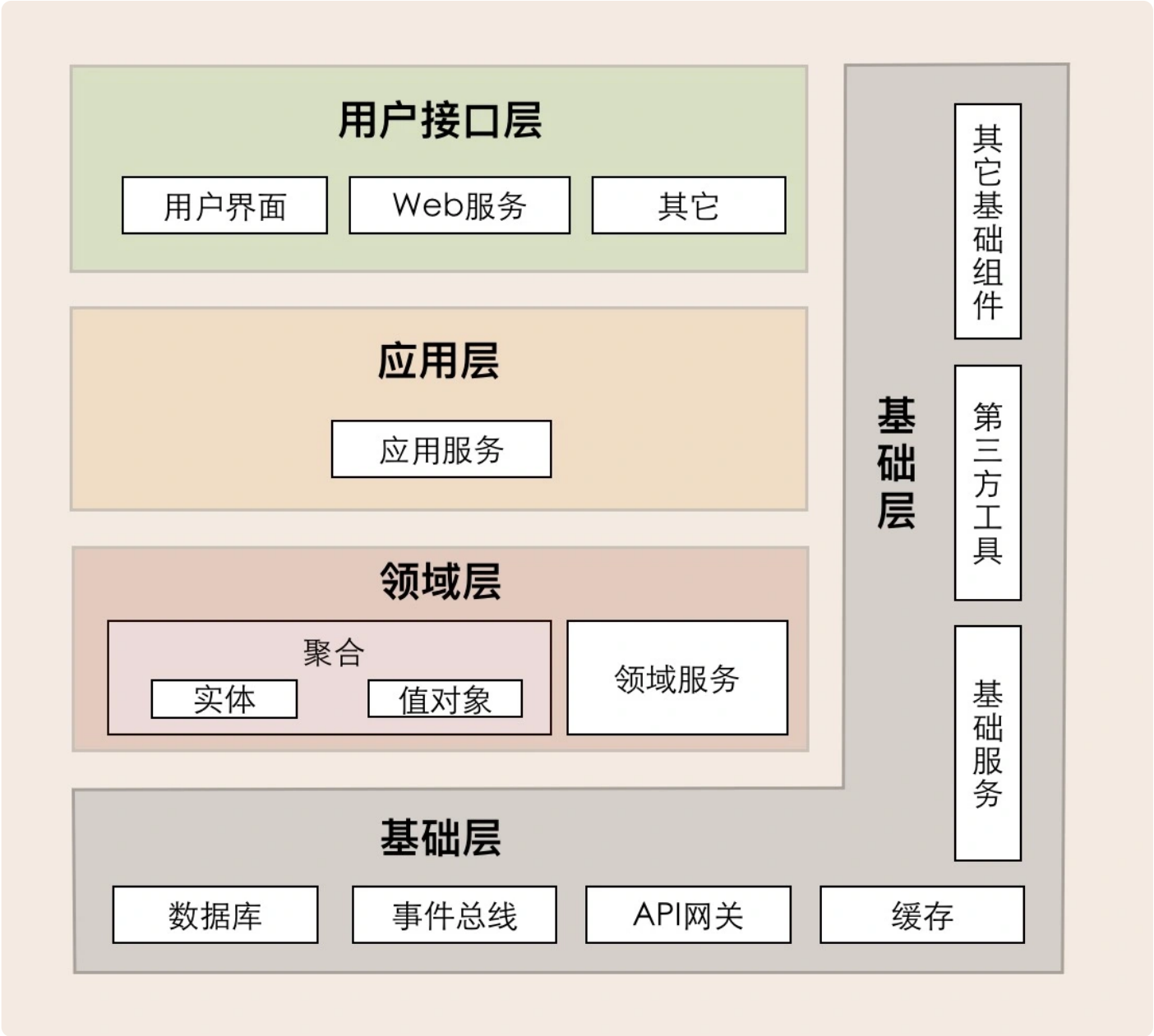
上一讲我们完成了领域模型的设计，接下来我们就要开始微服务的设计和落地了。那微服务落地时首先要确定的就是微服务的代码结构，也就是我今天要讲的微服务代码模型。

只有建立了标准的微服务代码模型和代码规范后，我们才可以将领域对象所对应的代码对象放在合适的软件包的目录结构中。标准的代码模型可以让项目团队成员更好地理解代码，根据代码规范实现团队协作；还可以让微服务各层的逻辑互不干扰、分工协作、各据其位、各司其职，避免不必要的代码混淆。另外，标准的代码模型还可以让你在微服务架构演进时，轻松完成代码重构。

那在 DDD 里，微服务的代码结构长什么样子呢？我们又是依据什么来建立微服务代码模型？这就是我们今天重点要解决的两个问题。

DDD 分层架构与微服务代码模型

我们参考 DDD 分层架构模型来设计微服务代码模型。没错！微服务代码模型就是依据 DDD 分层架构模型设计出来的。那为什么是 DDD 分层架构模型呢？



我们先简单回顾一下 [第 07 讲] 介绍过的 DDD 分层架构模型。它包括用户接口层、应用层、领域层和基础层，分层架构各层的职责边界非常清晰，又能有条不紊地分层协作。

> 用户接口层：面向前端提供服务适配，面向资源层提供资源适配。这一层聚集了接口适配相关的功能。

应用层职责：实现服务组合和编排，适应业务流程快速变化的需求。这一层聚集了应用服务和事件相关的功能。

领域层：实现领域的核心业务逻辑。这一层聚集了领域模型的聚合、聚合根、实体、值对象、领域服务和事件等领域对象，以及它们组合所形成的业务能力。

基础层：贯穿所有层，为各层提供基础资源服务。这一层聚集了各种底层资源相关的服务和能力。

业务逻辑从领域层、应用层到用户接口层逐层封装和协作，对外提供灵活的服务，既实现了各层的分工，又实现了各层的协作。因此，毋庸置疑，DDD 分层架构模型就是设计微服务代码模型的最佳依据。

微服务代码模型

现在，我们来看一下，按照 DDD 分层架构模型设计出来的微服务代码模型到底长什么样子呢？

其实，DDD 并没有给出标准的代码模型，不同的人可能会有不同理解。下面要说的这个微服务代码模型是我经过思考和实践后建立起来的，主要考虑的是微服务的边界、分层以及架构演进。

微服务一级目录结构

微服务一级目录是按照 DDD 分层架构的分层职责来定义的。从下面这张图中，我们可以看到，在代码模型里分别为用户接口层、应用层、领域层和基础层，建立了 interfaces、application、domain 和 infrastructure 四个一级代码目录。

这些目录的职能和代码形态是这样的。

Interfaces（用户接口层）：它主要存放用户接口层与前端交互、展现数据相关的代码。前端应用通过这一层的接口，向应用服务获取展现所需的数据。这一层主要用来处理用户发送的 Restful 请求，解析用户输入的配置文件，并将数据传递给 Application 层。数据的组装、数据传输格式以及 Facade 接口等代码都会放在这一层目录里。

Application（应用层）：它主要存放应用层服务组合和编排相关的代码。应用服务向下基于微服务内的领域服务或外部微服务的应用服务完成服务的编排和组合，向上为用户接口层提供各种应用数据展现支持服务。应用服务和事件等代码会放在这一层目录里。

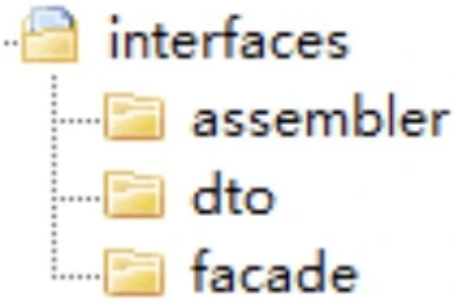
Domain（领域层）：它主要存放领域层核心业务逻辑相关的代码。领域层可以包含多个聚合代码包，它们共同实现领域模型的核心业务逻辑。聚合以及聚合内的实体、方法、领域服务和事件等代码会放在这一层目录里。

Infrastructure（基础层）：它主要存放基础资源服务相关的代码，为其它各层提供的通用技术能力、三方软件包、数据库服务、配置和基础资源服务的代码都会放在这一层目录里。

各层目录结构

1. 用户接口层

Interfaces 的代码目录结构有：assembler、dto 和 façade 三类。



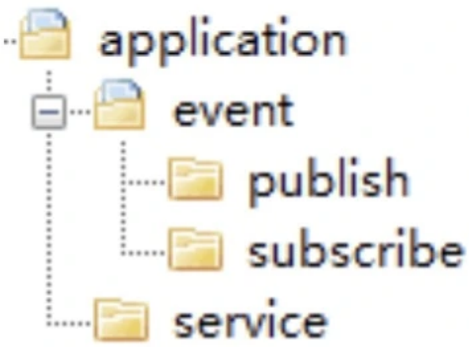
Assembler：实现 DTO 与领域对象之间的相互转换和数据交换。一般来说 Assembler 与 DTO 总是一同出现。

Dto：它是数据传输的载体，内部不存在任何业务逻辑，我们可以通过 DTO 把内部的领域对象与外界隔离。

Facade：提供较粗粒度的调用接口，将用户请求委派给一个或多个应用服务进行处理。

2. 应用层

Application 的代码目录结构有：event 和 service。



Event（事件）：这层目录主要存放事件相关的代码。它包括两个子目录：publish 和 subscribe。前者主要存放事件发布相关代码，后者主要存放事件订阅相关代码（事件处理相关的核心业务逻辑在领域层实现）。

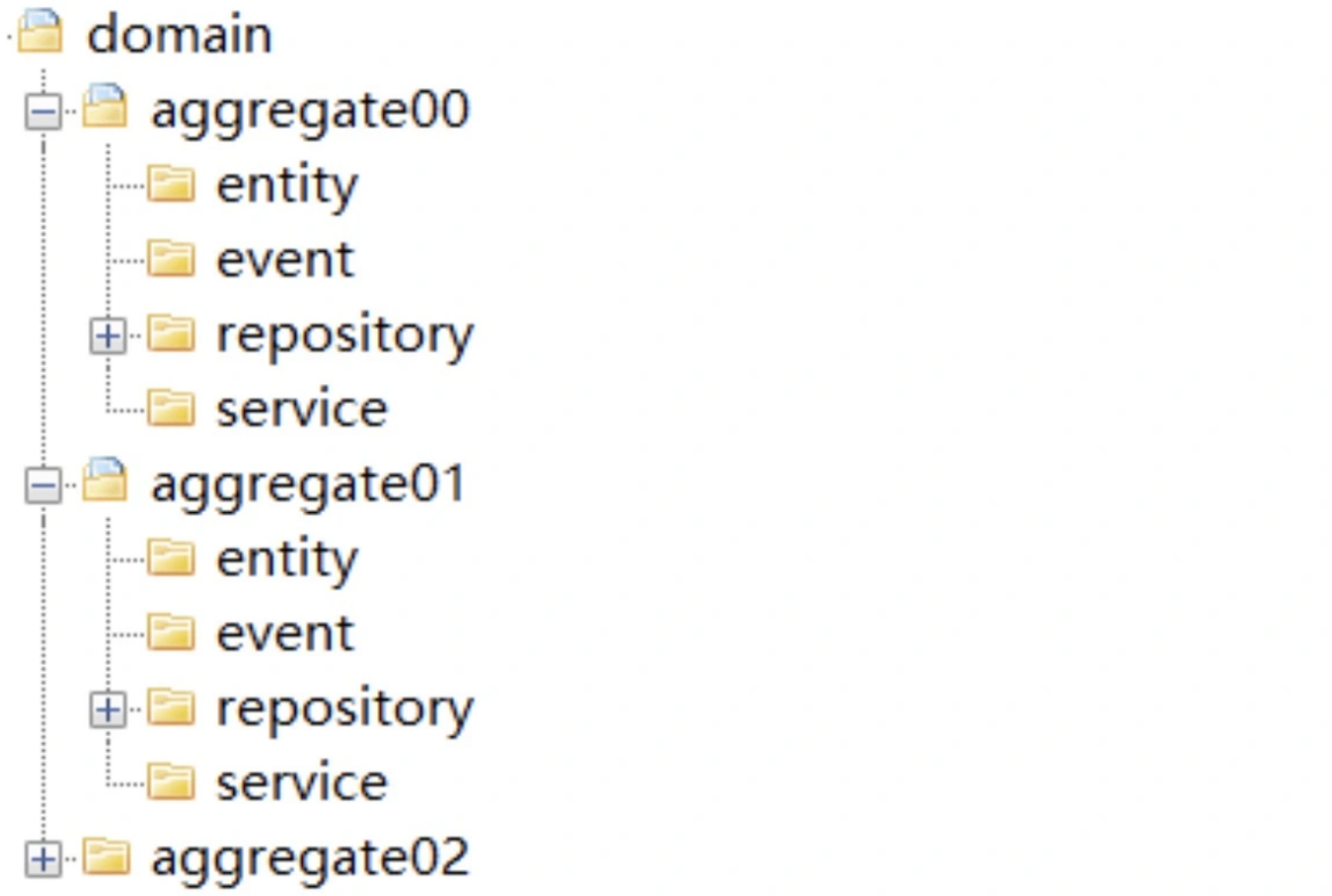
这里提示一下：虽然应用层和领域层都可以进行事件的发布和处理，但为了实现事件的统一管理，我建议你将微服务内所有事件的发布和订阅的处理都统一放到应用层，事件相关的核心业务逻辑实现放在领域层。通过应用层调用领域层服务，来实现完整的事件发布和订阅处理流程。

Service（应用服务）：这层的服务是应用服务。应用服务会对多个领域服务或外部应用服务进行封装、编排和组合，对外提供粗粒度的服务。应用服务主要实现服务组合和编排，是一段

独立的业务逻辑。你可以将所有应用服务放在一个应用服务类里，也可以把一个应用服务设计为一个应用服务类，以防应用服务类代码量过大。

3. 领域层

Domain 是由一个或多个聚合包构成，共同实现领域模型的核心业务逻辑。聚合内的代码模型是标准和统一的，包括：entity、event、repository 和 service 四个子目录。



而领域层聚合内部的代码目录结构是这样的。

Aggregate（聚合）： 它是聚合软件包的根目录，可以根据实际项目的聚合名称命名，比如权限聚合。在聚合内定义聚合根、实体和值对象以及领域服务之间的关系和边界。聚合内实现高内聚的业务逻辑，它的代码可以独立拆分为微服务。

以聚合为单位的代码放在一个包里的主要目的是为了业务内聚，而更大的目的是为了以后微服务之间聚合的重组。聚合之间清晰的代码边界，可以让你轻松地实现以聚合为单位的微服务重组，在微服务架构演进中有着很重要的作用。

Entity（实体）： 它存放聚合根、实体、值对象以及工厂模式（Factory）相关代码。实体类采用充血模型，同一实体相关的业务逻辑都在实体类代码中实现。跨实体的业务逻辑代码在领域服务中实现。

Event（事件）： 它存放事件实体以及与事件活动相关的业务逻辑代码。

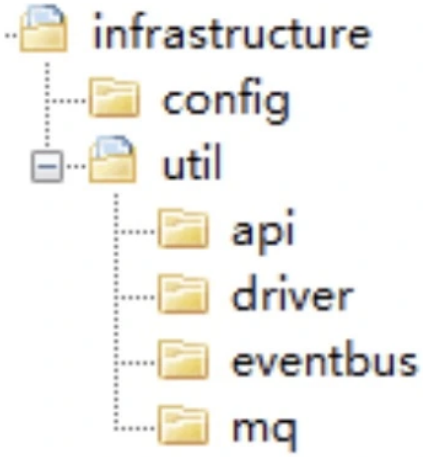
Service（领域服务）： 它存放领域服务代码。一个领域服务是多个实体组合出来的一段业务逻辑。你可以将聚合内所有领域服务都放在一个领域服务类中，你也可以把每一个领域服务设计为一个类。如果领域服务内的业务逻辑相对复杂，我建议你将一个领域服务设计为一个领域服务类，避免由于所有领域服务代码都放在一个领域服务类中，而出现代码臃肿的问题。领域服务封装多个实体或方法后向上层提供应用服务调用。

Repository（仓储）： 它存放所在聚合的查询或持久化领域对象的代码，通常包括仓储接口和仓储实现方法。为了方便聚合的拆分和组合，我们设定了一个原则：一个聚合对应一个仓储。

特别说明：按照 DDD 分层架构，仓储实现本应该属于基础层代码，但为了在微服务架构演进时，保证代码拆分和重组的便利性，我是把聚合仓储实现的代码放到了聚合包内。这样，如果需求或者设计发生变化导致聚合需要拆分或重组时，我们就可以将包括核心业务逻辑和仓储代码的聚合包整体迁移，轻松实现微服务架构演进。

4. 基础层

Infrastructure 的代码目录结构有：config 和 util 两个子目录。

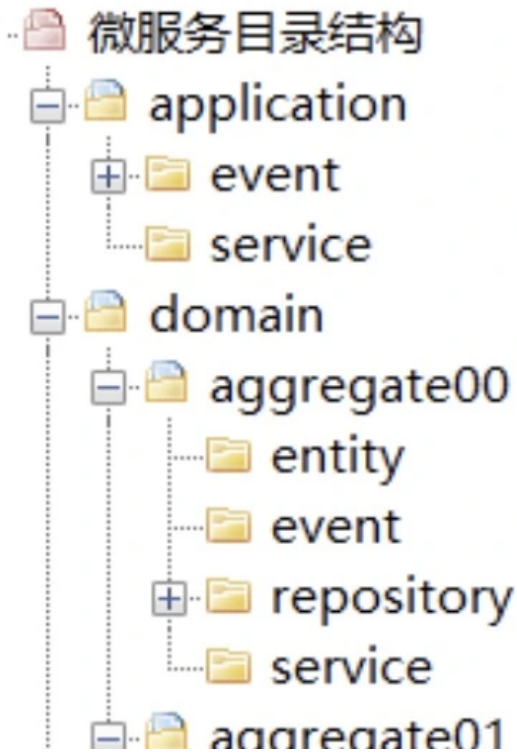


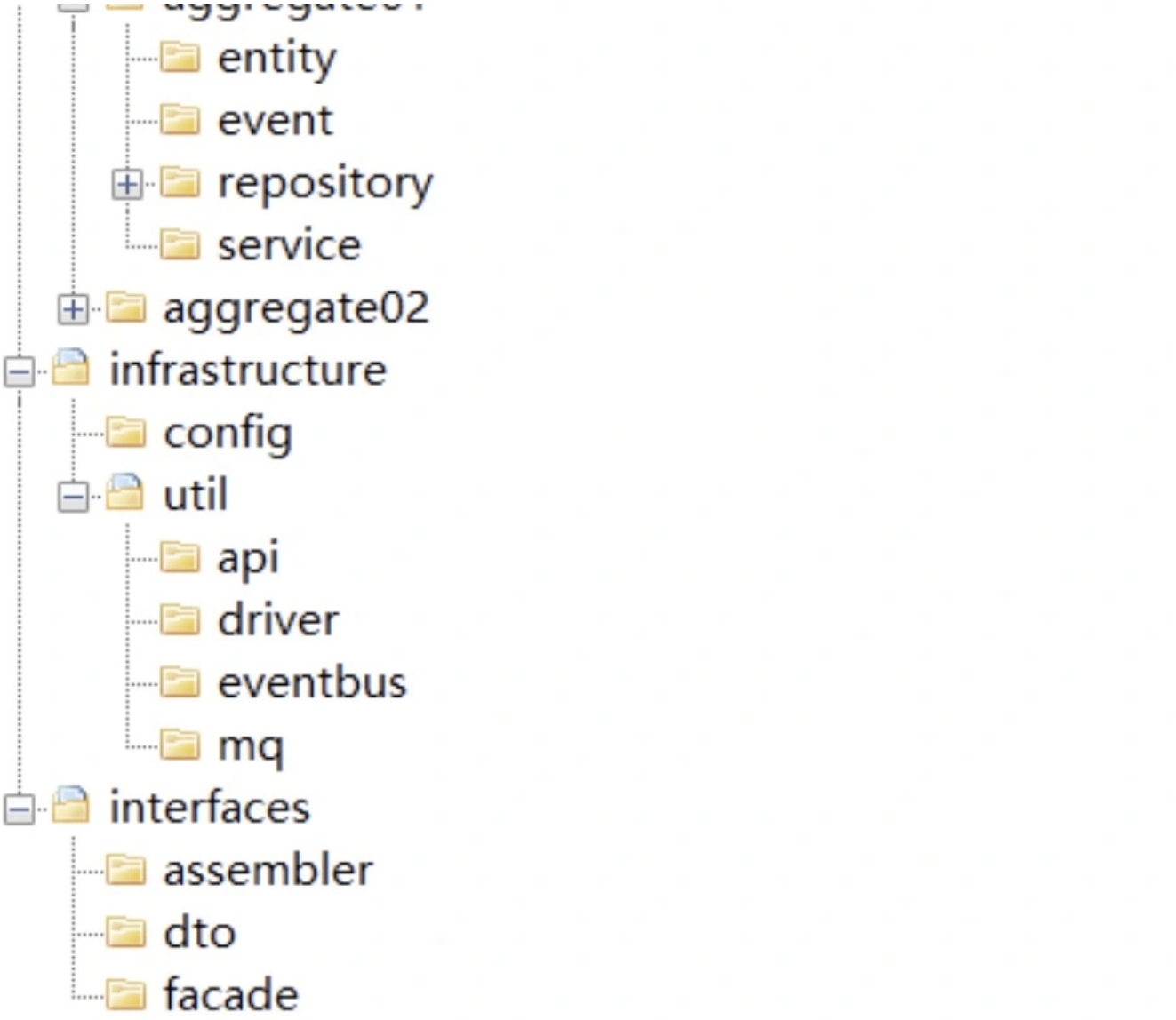
Config：主要存放配置相关代码。

Util：主要存放平台、开发框架、消息、数据库、缓存、文件、总线、网关、第三方类库、通用算法等基础代码，你可以为不同的资源类别建立不同的子目录。

代码模型总目录结构

在完成一级和二级代码模型设计后，你就可以看到下图这样的微服务代码模型的总目录结构了。





总结

今天我们根据 DDD 分层架构模型建立了标准的微服务代码模型，在代码模型里面，各代码对象各据其位、各司其职，共同协作完成微服务的业务逻辑。

那关于代码模型我还需要强调两点内容。

第一点：聚合之间的代码边界一定要清晰。聚合之间的服务调用和数据关联应该是尽可能的松耦合和低关联，聚合之间的服务调用应该通过上层的应用层组合实现调用，原则上不允许聚合之间直接调用领域服务。这种松耦合的代码关联，在以后业务发展和需求变更时，可以很方便地实现业务功能和聚合代码的重组，在微服务架构演进中将会起到非常重要的作用。

第二点：你一定要有代码分层的概念。写代码时一定要搞清楚代码的职责，将它放在职责对应的代码目录内。应用层代码主要完成服务组合和编排，以及聚合之间的协作，它是很薄的一层，不应该有核心领域逻辑代码。领域层是业务的核心，领域模型的核心逻辑代码一定要在领域层实现。如果将核心领域逻辑代码放到应用层，你的基于 DDD 分层架构模型的微服务慢慢就会演变成传统的三层架构模型了。

思考题

对比一下 DDD 分层架构和三层架构的代码结构的差异？

期待你的分享，我们一同交流！

精选留言 (95)

陈　争
2019-11-13

不知道我这样理解的对不对
比如执行一个创建用户的命令，
1.用户接口层：
1.1)Assembler->将CustomerDTO转换为CustomerEntity
1.2)Dto->接收请求传入的数据CustomerDTO
1.3)Facade->调用应用层创建用户方法
2.应用层
2.1)Event->发布用户创建事件给其它微服务
2.2)Service:
内部服务->创建用户
外部服务->创建日志
3. 领域层
3.1)Aggregate->进入用户聚合目录下(如：CustomerAggregate)
3.2)Entity->用户聚合跟
3.3)Event->创建用户事件

3.4)Service->具体的创建用户逻辑，比如用户是否重复校验，分配初始密码等
3.5)Repository->将用户信息保存到数据库

作者回复: 是的，就是这样的。理解的很到位。

共 14 条评论 >　　87

HuAng
2020-12-23

老师，这种代码分层的优势在哪里？不知道你还看不看留言。

作者回复: 分层架构大多是通过前端或后端适配，逐层控制外部变化向领域层传导，从而降低外部变化对领域模型的影响。比如在前端应用中可以消化掉页面逻辑和页面流程类需求；在用户接口层可以完成前端应用接口和数据适配，避免将接口和数据适配类需求传导到应用层；在应用层通过服务组合和编排，可以避免用例或服务组合类需求向领域层传导；在基础设施层通过依赖倒置设计，可以隔离技术组件变化对领域逻辑的影响。
这样由外向里逐层消化和隔离外部变化对领域模型的影响，从而可以最大限度保持领域模型原子性和长期稳定。
一个稳定的领域模型可以给我们带来非常多的好处。
首先，领域模型的构建过程是项目团队通用语言建立的过程，领域模型就是团队的通用语言，它会贯穿项目的所有过程。
其次，领域模型的业务逻辑大多是可复用的原子逻辑，不易受外部变化的影响。将这些核心逻辑沉淀到领域层，让核心逻辑更聚焦，核心代码更内聚。然后在应用层通过应用服务对领域层服务完成组合和编排，可以大量复用领域层的核心业务逻辑代码，从而提升代码复用率。
第三，领域模型的核心代码可交由专门的资深开发人员维护，从而提升代码质量，也能够保证应用关键核心业务逻辑的长期稳定运行。
第四，领域模型内聚合边界更加清晰，可方便微服务以聚合为单位的功能和代码的拆分和重组，让微服务具有更强的演进能力。

共 2 条评论 >　　22

xj_zh
2019-11-18

老师，求DDD的系统样例代码。

作者回复: 代码样例还没准备好，后面我找时间整理一下吧。

共 9 条评论 >　　14

FIGNT

2019-11-14

我们在设计领域模型时，遇到一些问题

1. 查询聚合的操作应该放在哪一层?
2. entity的实体和值对象太多需要分目录吗?
3. 针对实体的维护，需要通过聚合去维护吗？可以直接修改实体吗?
4. 一个聚合保存在一个库里，还是多个聚合都在一个库里？一个实体需要单独放一个库吗?如果一个实体被修改了。用到这个实体的聚合需要更新吗?
5. 聚合是设计成单个的还是批处理的？比如一棵树，业务上是以一片叶子为单位的，那么是以树为聚合还是以叶子为聚合?

作者回复: 1、个人感觉批量大数据量的查询用仓储有点勉强，你可以用传统的方式来做。如果不涉及到领域逻辑的话，可以放应用层。

2、一个微服务的聚合内部应该不会有太多的实体和值对象吧。在目录结构里面是一个聚合一个代码目录。当然如果实在太多，你是可以再分目录的。

3、聚合内的实体数据维护是通过聚合根通过仓储来统一维护的。

4、一个微服务一个库，微服务内的多个聚合可以共用一个库，但是尽量避免聚合之间的表关联，聚合之间的数据要做到松耦合。

5、不清楚你说的单个和批处理是什么意思？聚合是具有一个完整业务功能的单位，就看你业务的粒度大小。多个不同功能的聚合是可以构成一个比较大的业务模块。

13

Farewell、

2019-11-13

1.应用服务只能调用领域服务和实体的方法，能调用仓储接口的方法么?
按理说应该隔离，也就是说应用服务应该调用领域服务的方法，再让领域服务调用仓储接口的方法吧?

2.实体的转换只有从用户接口层到应用服务层一次是么？也就是说，到应用服务层之后，以及之后的仓储接口都是可以直接对领域实体进行操作的?

3.参考了Spring Data Jdbc项目，里边也采用了DDD的设计思路，但是发现会需要在实体中配置一些和底层存储相关的注解，这样会不会不能把领域层可仓储实现进行隔离？如果是这样的化，那么Spring Data Jdbc是不是没有严格遵守DDD的一些设计？而且它提供的领域事件的发布机制实现，是在对应的实体中产生的，例如在某一个实体中定义产生领域事件的源头，当对应的实体保存或更新时，就会发出这样一个领域事件。按照咱们文章中讲解的事件的发布是在应用层，那么如果要这样做的话，是不是就需要在应用层重新转发领域层实体内产生的领域事

件呢?
因为看到Spring Data这样比较广泛的项目实现和咱们文章的描述有一些我理解上的区别，所以比较困惑和疑问。

作者回复: 1、如果是应用服务直接调用文件或者缓存之类的，应用服务是可以之间调用仓储的。但如果中间有领域实体和数据库，则需通过领域服务，然后通过聚合根来调用仓储。
2、用户接口层大多是DTO，应用层和领域层大多是DO，基础层则是PO，在不同层之间是需要进行数据转换的。我有一节专门讲这个。
3、如果是这样的话，确实领域层与数据库层会有耦合。领域事件其实放领域层也是可以的，放应用层主要是为了统一管理。如果领域事件放在实体内部，查找和运维起来就不是太方便，而且这个实体还需要对领域事件的实体进行操作。目录结构的设计主要是从边界、分层和便利性考虑的。

共 3 条评论 > 9

小明
2020-11-03

欧阳老师，看回复收获很多，我也想请教一个问题，比如一个查询商品详情接口，包含查询商品信息、店铺信息、促销信息，可能跨多个域，那么结合DDD分层设计，应该怎么做呢，谢谢

作者回复: 在用DDD设计时，这种复杂的查询逻辑一般不放到领域模型的领域逻辑中，而是采用CQRS模式，也就是一种读写分离的设计模型，在CQRS模型中读业务模型和写业务模型（领域模型）是分离的。
第一种方案，如果读模型和写模型在同一个微服务中，这种复杂查询模型可以共享同一个数据库的不同聚合的数据库表，用传统的SQL查询就可以实现复杂的跨聚合查询。
第二种方案，你也可以将读模型独立为查询库和查询微服务，汇集多个源头数据后，重构多个微服务或者聚合的读数据模型，然后提供独立的查询服务。
具体选用哪种方案，需要结合你的技术和业务场景来设计。

共 3 条评论 > 6

剑八
2020-05-24

传统三层：用户接口层，业务层，基础层
领域DDD分层：用户接口层，应用层，领域层，基础层
两者差异，DDD分层将业务领域核心逻辑聚焦在领域层，而跨聚合的组合编排在应用层
这在传统三层中实现跨域的操作需要在用户接口层或者业务层加个组合方法，导致分层不清

共 1 条评论 > 6

刘小龙
2020-04-05

Repository，放在领域层，如果一个对象出现在领域，多个领域对其进行操作，会不会太多重复的操作数据库？【将包括核心业务逻辑和仓储代码的聚合包整体迁移，轻松实现微服务架构演进】这个是为了将各个领域的代码进行隔离，进行了竖向划分，达到目标。如果系统过大，将基础层划分出来，接口层划分出来，也就横向划分，是不是也行？

作者回复: 聚合在领域建模的时候是要坚持职责单一性原则的，一个聚合的功能在企业内应该是唯一的。而一个仓储只对应一个聚合，所以这个仓储应该不会共享给其它的领域模型，不会在基础层被其它的聚合复用的。


5

Tony
2021-01-27

实体自身的业务逻辑放在实体里面，会不会让实体对象很庞大，假如实体里面的业务逻辑设计仓储层的调用会不会有点奇怪了？求解

作者回复: 充血模型才是真正的面向对象设计方式。
在一个聚合中，除了聚合根和领域服务外，一般的实体不会调用仓储接口的。为了保证聚合内数据的一致性和符合聚合的业务规则，聚合新增或修改的数据通常都是通过聚合根或领域服务一次提交到持久层，不允许直接去修改某一个普通实体的持久化数据的。


4

甲小蛙
2020-07-30

感觉老师有点纸上谈兵的意思呢，只讲理论，连示例代码库都没有，感觉谁都可以照着书来一个这样的课程了，总是浅尝辄止，不免有点失望

作者回复: 加餐里面有一个完整的示例代码分析。


4

燕羽阳
2020-01-03

在《实现领域驱动设计》这本书中，Demo(https://github.com/VaughnVernon/IDDD_Sample

s)。
会倾向于：在application中调用repository, 领域实体和领域服务是不应当调用repository的，这样领域层会保持独立。在实际写代码的过程中，发现这样代码写比较麻烦。
老师能在详细对比讲讲，对repository和第三方接口依赖的情况，在哪一层处理么？

作者回复: 是的，我感觉对应复杂的聚合在领域服务中调用repository比较顺手。第三方接口我建议放在应用服务中处理，应用服务主要对服务进行组合和编排。

4

开心小毛
2020-05-09

接口层的facade是不是用来解析REST URL的: 调用URL对应的应用层服务，并传入URL变量作为调用参数。这样理解对么？谢谢老师。

作者回复: 是这样的。
在微服务面向不同前端应用时，同样的一段业务逻辑，可能由于渠道不同，而在前端展示的页面要素不同，因此要求后端微服务返回的数据结果会不同。比如在面向内部员工的PC端应用时，可能要求返回全部数据。而面向外部客户的移动端应用，可能只需要返回几个关键数据就可以了。
为了避免暴露微服务的核心业务逻辑，防止数据外泄，你不能将后端所有的数据，不加区分的传递给前端应用。你更不能仅仅因为前端应用不同的数据展示需求，而开发出多个不同的后端服务，面向前端应用提供不同的服务。
这时用户接口层的Facade服务和数据组装器Assembler就可以发挥作用了。Facade服务可以封装应用服务，数据组装器Assembler可以根据不同前端应用的数据需求，完成前端DTO和后端DO对象的组装和转换等操作。面向不同前端应用提供不同的Facade接口和DTO数据服务。这样，我们不需要调整任何后端服务，就可以面向不同的前端应用，提供灵活的接口定制和数据适配服务了。

3

何磊
2020-03-22

老师好，有两个个疑问：
1. 如果在一个聚合中，只有一个 Entity，它已经实现了所有的业务逻辑，那么是否在聚合中还需要领域服务？

如果Entity跟Service都有业务逻辑的实现，在应用层调用的时候能否直接调用Entity呢？还是说不管如何。Service都要封装一下Entity的业务逻辑，方便应用层调用？

2. 这里困扰我的第二个问题，如果我的A微服务，需要先从自己的存储中获取数据的id（比如商品id），然后调用B服务获取id具体的信息，再进行其它处理。那么我是否这个逻辑是，A服务提供了一个查询id的方法，以及获取数据后对数据处理的方法；然后再A服务的应用层先调用查询id的方法，然后调用B服务获取到数据后，再调用另外一个方法来处理吗？

作者回复: 第一个问题，我在分层架构时讲到了，分层架构包括严格分层架构和松散分层架构，在严格分层架构中只能上层调下层，在松散分层架构中上层可以调任意下层。如果你采用严格分层架构，那就需要封装成领域服务，如果采用松散分层架构，应用层可以直接调实体方法。我一般建议采用严格分层架构。
第二问题，是这样的。这就是应用服务的职责，负责微服务内不同聚合领域服务和微服务之间的应用服务的组合和编排。

共 3 条评论 >　　3

Jerry银银
2019-12-30

我的理解是各个不同的微服务应该是共享一套infrastructure的吧？如果是的话，代码结构是不是不应该这样？

作者回复: 这里的基础层应该与基础层的技术组件有些差异。比如数据库、api网关之类的基础组件是单独部署而且可能是很多微服务共享的。这里的基础层更多的是偏向于应用与这些基础组件的交互相关的内容，比如组件驱动以及适配相关的代码等内容。

共 2 条评论 >　　3

杨杰
2019-11-18

关于微服务的用户接口层和应用层有点儿疑问。在整个微服务架构里面一般微服务上层还有BFF层、聚合服务层，一般BFF层或聚合服务层用来协调多个微服务或者做数据转换。那么对于某个具体的微服务是否还需要用户接口层和应用层的区分呢？如果DTO是在用户接口层，那么这些数据如何传入到应用服务层呢？

作者回复: 你可以这样定位。微服务内的应用层主要处理自己的逻辑编排，bff主要处理微服务之间的逻辑。

 　　3

Doctor Lau

2020-11-11

个人理解repository的实现应该放到infra，才能起到隔离持久层的作用，但这样按目前的工程结构会出现循环依赖，除非将领域模型提出到单独工程，请问这个问题怎么考虑的？

作者回复: 基础资源层可以面向所有层提供服务。放在领域层主要是考虑一个聚合对应一个仓储，方便以后以聚合为维度的代码拆分。如果在同一个微服务工程里面，个人感觉放在领域层与放在基础层的依赖关系不会有多大的改变吧。不清楚您说的循环依赖是什么样的场景？能否举一个案例？感谢！

2

z

2020-05-19

业务相关的校验是放到应用层还是领域层呢？放在应用层的话应用层有了业务逻辑。放在领域层，面向web的和面向微服务间的调用校验规则不同又不够灵活难以实现。

作者回复: 一般来说是这样的，前端页面逻辑的数据校验在前端应用完成。应用服务中进行安全认证、权限校验、事务控制、领域事件发布或订阅等。业务规则的校验一般在领域层完成。

2

苗

2020-04-14

老师，请教一下大家，业务数据网关如果实现？一个order-service提供了一个queryOrder的接口，输入起始日期查询对应的订单列表，其有2个消费者：C端网站应用服务 和 报表应用服务，C端网站应用服务 只需要知道订单的基本信息如下单时间、商品名称、金额就可以了，而报表应用服务是给管理者看的，需要的订单数据很全，除了C端网站应用服务需要的之外，还需要看平台与商家的结算金额。另外看到一篇类似的微服务分层方面的文章和课程中相比多了一个网关层，http://tbwork.org/2018/10/25/layed-dev-arch/#4%E9%A2%86%E5%9F%9F%E5%88%92%E5%88%86%E5%92%8C%E5%BE%AE%E6%9C%8D%E5%8A%A1%E5%8C%96 请老师看看，然后讲解下这个网关层和api gateway是一样的吗？老师有微信交流群吗？

作者回复: 大概看了一下他的文章，可能理解上有些差异，这篇文章中的用户界面层应该是前端应用的概念，而网关应该就是API网关。
你说的这个情况其实在DDD用户接口层很好解决，应用服务可以拿到订单相关的所有数据，然后用户接口层在组织Facade服务的时候，可以调用应用服务获取DO数据，然后根据外部不同接口的数据要求，将DO数据进行重新组装成不同的DTO，比如面向C端DTO是包含了下单时间、商品名称、金额。

而面向报表DTO则包含了更全的订单数据。这样一个应用服务可以经过Facade封装成两个面向不同前端并提供不同数据的用户接口层的服务了，这个服务可以发布在API网关中，供前端应用调用。

2

码弓手
2020-03-27

文章看完，还是又一些关于调用过程的疑问，Application层使用的是Domain层中聚合的entity对象对象呢还是repository对象，repository不是更应该放在基础层吗，repository理解起来还有一些困惑，希望大佬以一个下单为例子解说一下每个层的调用链路，谢谢！

作者回复: 你可以看一下第十八章，里面有一个服务调用关系图，在加餐里面也有代码示例，希望能帮你加深理解。

2

Jerry银银
2019-12-30

请教一下老师：客户端开发中，存在着UI层，那UI层的代码也放在哪儿呢？

作者回复: 微服务架构下前后端分离，前端会有自己的工程的。

共 3 条评论 >

2