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

DDD实战课 进入课程 >

讲述：欧创新
时长 15:13　大小 10.46M

你好，我是欧创新。

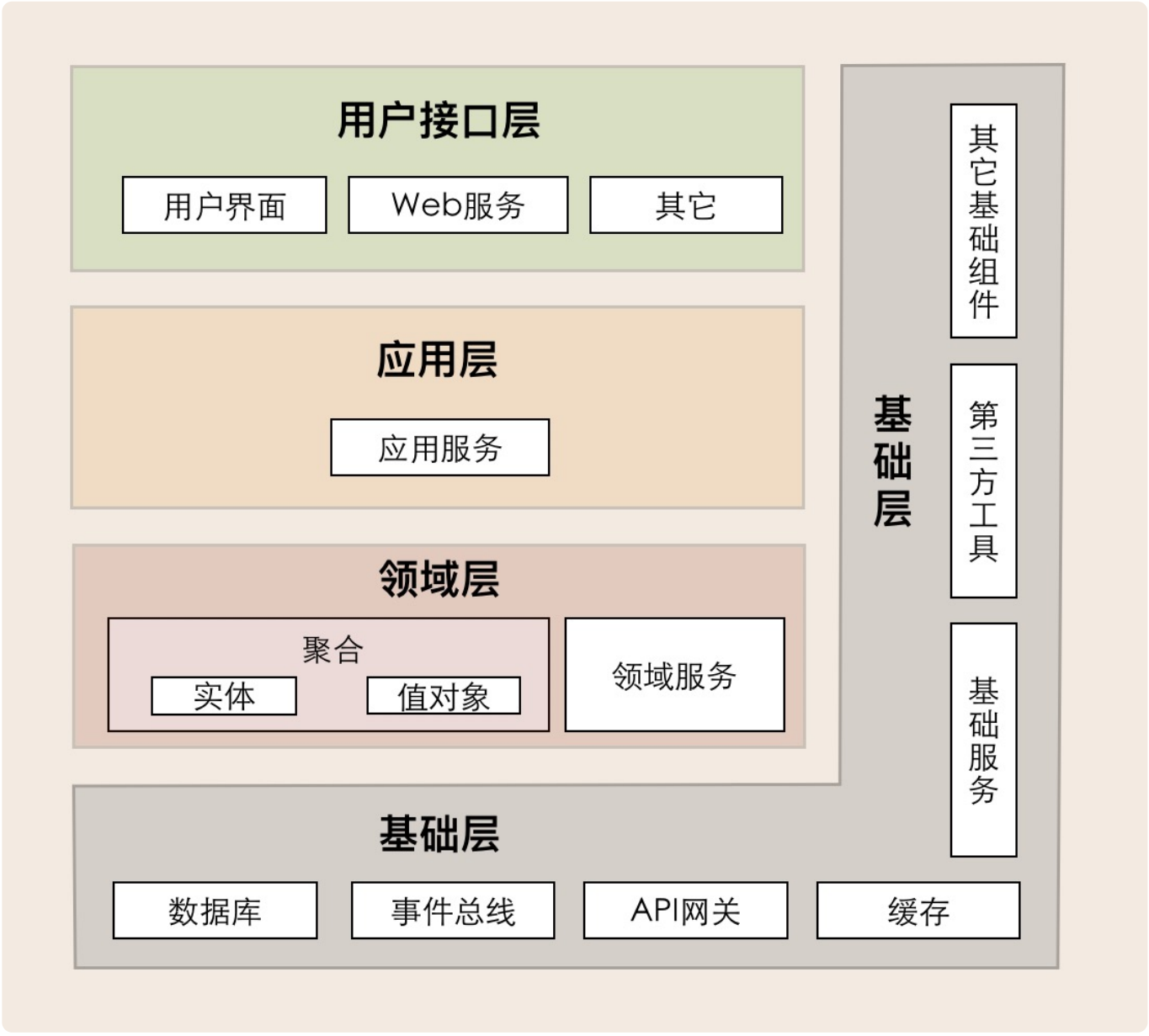
上一讲我们完成了领域模型的设计，接下来我们就要开始微服务的设计和落地了。那微服务落地时首先要确定的就是微服务的代码结构，也就是我今天要讲的微服务代码模型。

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。

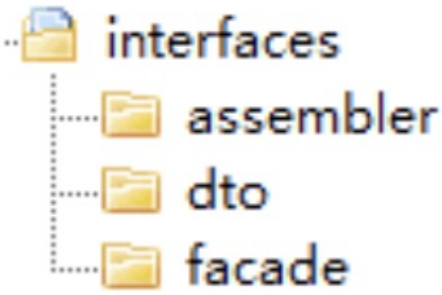
Domain（领域层）：它主要存放领域层核心业务逻辑相关的代码。领域层可以包含多个聚合代码包，它们共同实现领域模型的核心业务逻辑。聚合以及聚合内的实体、方法、领域服务和事件等代码会放在这一层目录里。

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。



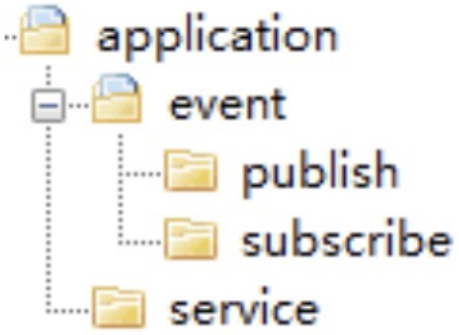
Assembler：实现 DTO 与领域对象之间的相互转换和数据交换。一般来说 Assembler 与 DTO 总是一同出现。

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。



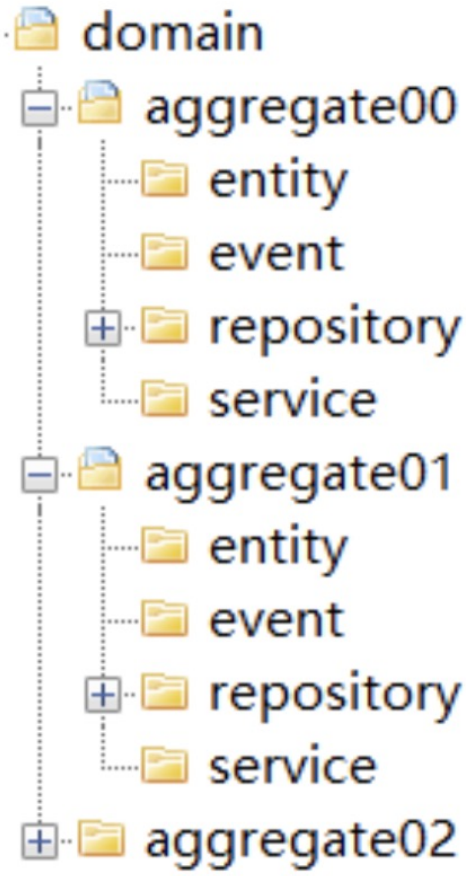
Event（事件）：这层目录主要存放事件相关的代码。它包括两个子目录：publish 和 subscribe。前者主要存放事件发布相关代码，后者主要存放事件订阅相关代码（事件处理相关的核心业务逻辑在领域层实现）。

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。

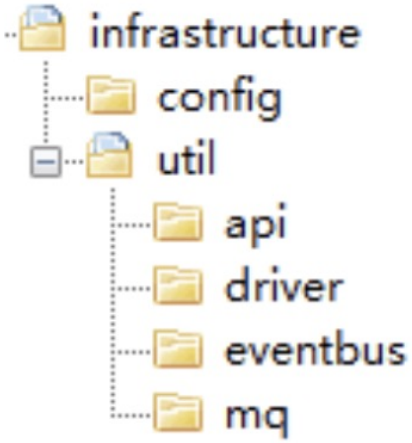
Service（领域服务）：它存放领域服务代码。一个领域服务是多个实体组合出来的一段业务逻辑。你可以将聚合内所有领域服务都放在一个领域服务类中，你也可以把每一个领域服务设计为一个类。如果领域服务内的业务逻辑相对复杂，我建议你将一个领域服务设计为一个领域服务类，避免由于所有领域服务代码都放在一个领域服务类中，而出现代码臃肿的问题。领域服务封装多个实体或方法后向上层提供应用服务调用。

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。

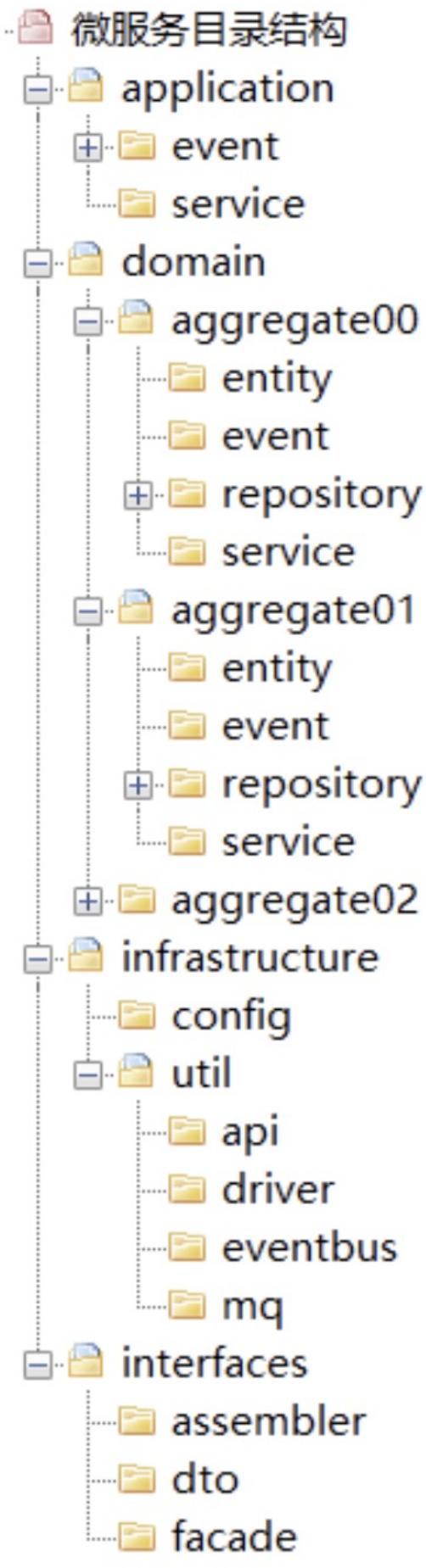


Config：主要存放配置相关代码。

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！

上一篇　12 | 领域建模：如何用事件风暴构建领域模型?

下一篇　14 | 代码模型（下）：如何保证领域模型与代码模型的一致性?

精选留言 (31)

写留言

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...

展开

作者回复: 是的，就是这样的。理解的很到位。

2　5

xj_zh
2019-11-18

老师，求DDD的系统样例代码。

展开

作者回复: 代码样例还没准备好，后面我找时间整理一下吧。

1

Todd BD

2019-11-14

如果我有多个聚合， 比如聚合根A和聚合根B， 从业务的角度讲，可以接受AB间数据的最终一致性，但从数据展示的角度考虑， A和B是有强关联性的，也就是说在页面上，他们总是一起在页面的某部分出现， 那么在应用层是否要在query 接口中把这两个聚合根封装成一个新的对象再返回?
还是我想的太多了， application层应该以增 删 改这种业务诉求为导向设计， 而query...

展开

作者回复: 你可以分别调两个聚合的领域服务，然后将两个聚合根的DO对象转换为一个DTO，就可以给前端提供包含两个聚合数据的数据服务了。

2 1

如是

2019-11-13

还有这个依赖倒置，一开始感觉很清晰，现在感觉越看越懵，老师能给再解答下吗?

作者回复: 我在你的上个问题回复了，不知道解答你的问题了没有。不知道你的疑问在哪里?如果没解答清楚，麻烦告诉一下你的疑惑在哪里哈。

1 1

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，...

展开

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。

1

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？

展开

陈云
2019-11-17

有一个细节，想问下是如何处理的，接口层接收到的DTO对象,里面的字段跨多个DO，你没办法将一个DTO 完全转成一个DO，这里可能一个DTO的一些字段压根DO里就没有，这个时候 如果 从接口层 传到应用层，再传到 领域层？再封装一个VO对象？然后这个VO对象 是属于领域层？

展开

作者回复: 不同的对象在不同的层转换。用户接口层DTO和DO转换，应用层主要是DO，调外部微服务的服务的时候应用层有dto和do的转换。领域层与基础层之间，在基础层有DO和PO的转换。

小美
2019-11-15

今天的文章中写到应用层编排领域服务和外部服务。领域服务可以直接调用外部服务吗？比如一项业务是先根据实体组装调用参数，然后调用外部服务，再根据结果更新前面实体的状态，那这项业务是由领域服务整体实现？还是由应用层来编排，调实体获取参数，调外部服务，调实体更新状态？

展开

作者回复: 对外还是尽量靠应用服务来实现。

领域服务也不是不能做，要考虑耦合和对核心逻辑的影响，综合考虑成本吧。


美美

2019-11-14

请问下接口层，应用层，领域层，基础层是一个部署单元还是4个部署单元，老师可以截一个真实项目的maven项目结构吗?

展开

作者回复: 是在一个部署单元，它们合在一起就是一个微服务。


Todd BD

2019-11-14

只有应用层的方法才允许暴露给接口层吗？ 比如聚合的service已经能很好的表达业务的诉求，而且并不需要在应用层进行编排，是否在接口层可以直接调用聚合的service？ 还是需要在应用层代理一次供接口层调用?

作者回复: 松散分层架构是可以的。严格分层架构只能调用它的紧邻的下层。


如是

2019-11-14

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...

展开

作者回复: 😁

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...

展开

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。

Harvey
2019-11-14

spring-mvc的Controller和Intercepter应该放在哪个目录下呢?

作者回复: 这些是基础层的组件吧。

ZIxuAN
2019-11-13

每次通过聚合执行一些操作之前都要先查询聚合，如果这个聚合要从很多表中查数据组装的话性能就很低，请问这里是不是要针对整个聚合根加缓存呢?

展开

作者回复: 批量的查询我不太建议走聚合根，不走聚合根，用传统方法也行。

4

墨名次
2019-11-13

“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。” ...这句话好矛盾啊，不应该是一个领域服务设计多个领域服务类，才能避免所有领域服务代码都放在一个领域服务类里面吗？

展开

作者回复: 你可以理解为一个领域服务就是一个方法，一个方法设计为一个类，而不是将所有的方法放在一个类里，这样当领域服务业务逻辑很复杂的时候，一个类的代码量就不会太大了。

瓜瓜
2019-11-13

通用语言，我感觉我们在学习这个课程的时候，就需要通用语言，很过名词和概念到现在已经开始有点混淆了。好像需要重头再看一遍了。
今天我们讲的这个是一个微服务内的代码结构是吧？

作者回复: 概念确实有点多哈。今天讲的就是微服务内的代码结构。

瓜瓜
2019-11-13

差异我想是因人而异的，如果能把业务理解透彻，业务模型划分清楚，代码职责能很明确。
这种差异我想不会太大。DDD这种设计思想，我想就是一种工具化的方法，帮助我们能实现以上目标的一种途径。

展开

作者回复: 是这样的。

何沛
2019-11-13

三层架构在微服务系统是不合时宜的。因为应用层和领域层耦合在一起，会导致你的领域

层被频繁的需求变更，变得越来越不可复用。
我的服务改造前经常会遇到提供给服务甲和服务乙因为一个参数不兼容，或者编排服务的顺序不能兼容，就得重新提供一个新的服务出来。
...
展开

作者回复: 是的。
微服务架构下要考虑新的设计方法。

Jxin
2019-11-13

回答问题：
1.传统三层和面向过程编程没有什么区别。而ddd的分层给予了领域模型行为，脱离了贫血模型的处境，更贴合面向对象设计的理念。
2.充血领域模型在代码复用上会比传统三层更简洁。力度更小，语义更强，依赖更清晰。
...
展开

作者回复: 1、我这样设计的主要出发点就是为了聚合之间的解耦，避免以后微服务架构演进时，聚合之间耦合度过高而带来解耦的成本。而应用层是跨多聚合的，相对来说对聚合之间的协调就比较容易了。
2、放基础层其实也是没有问题的，但是要保证不同聚合之间的代码隔离和业务逻辑的解耦。将来如果以聚合为单位进行微服务之间代码的重组，实现微服务的架构演进，很容易实现代码和业务逻辑剥离。
设计的时候只要聚合之间逻辑是解耦的，代码是隔离的，并且按照分层原则把代码放在合适的层，把握好这个原则，可以按照自己熟悉的方式去做，不必受限于这个目录结构，能实现微服务架构的轻松演进就可以了。

atom992
2019-11-13

要是能再有一个调用依赖关系的图就好了。
展开

作者回复: 后面有一节专门讲服务依赖和数据转换。