电商服务开放平台-DDD设计实战三

DDD项目演进之道：服务开放平台微服务体系设计改造实战

--楼兰

一、服务开放平台后端微服务拆分。

上一节课将tlmall-open从一个MVC的微服务架构退化成了tlmall-openV2的本地架构。虽然项目结构变得清晰了一点，但是后端只能接入两个虚拟的服务，距离把电商项目的后端服务暴露给App的目的，似乎更遥远了。毕竟你不太可能真的要求电商项目为了接入tlmall-open，去开发一大堆的jar包。接下来，我们就将tlmll-openV2的后端服务接入流程，从单体架构升级成与电商项目兼容的微服务架构，真正把电商项目的后端服务接入进来。

首先设计接入方案

方案的核心依然是要通过App申请的ServiceCode路由到具体的后端服务中，所以重点是要设计一个后端服务与ServiceCode的对应关系。在Nacos服务发现机制中，服务名和分组名都可以作为对应的要素。例如可以直接用服务名来替代GsService中的serviceCode来进行服务路由。分组名如果要与电商的后端服务隔离开，就可以单独指定一个分组。这里简单一点，就直接用电商项目的DEFAULT_GROUP就可以了。

找到对应的后端服务后，接下来要调用后端服务。只要有一个机制能找到后端对应的业务方法就可以了。由于Feign是基于请求地址做的服务路由，所以，这里可以规定一个后端服务一个固定的请求路径来替代单机版本固定的doBusi方法。在这里，可以指定一个固定的请求路径/open/service即可。

> 方案的设计是很自由的，你可以自己想到很多的扩展方式。比如给服务名指定一个特殊一点的规范，与电商的后台服务区分开，或者将请求路径更换为/open/{serviceCode}，在一个后端服务中动态的支持多个服务。你可以尝试下在tlmall-openV2中自己进行这样的实现。

然后如何将设计方案落地

在DDD的指导下，应用中的各种变化都被有效的隔离开了。你会发现，几乎每一个功能都有专门的组件负责。而每个组件，都是通过依赖反转原则提供了接口扩展的。未来的业务变更，如果变动不大，可以把组件的功能实现稍微修改一下就可以了。如果变动很大，直接提供新的实现即可。

在tlmall-openV2中，最终的业务请求转发会交由AsyncBusiService组件来实现的。这次的改动其实算是比较大的，毕竟整个服务转发的机制都做了改变。所以，在tlmall-openV2的代码中，同时保留了基于Local的实现类以及基于Nacos的实现类。你只需要在GsServiceController中选择注入特定的实现类即可。

在实现NacosAsyncBusiServiceImpl时，可以通过discoveryClient直接去找Nacos上注册的服务实例，然后这里是用一个随机数来实现简单的负载均衡。找到一个具体的服务实例，发起POST请求即可。在示例当中，就完成了对open-mobiletag的微服务化。希望你能把这个调用过程想明白。

> 1、这个过程依然可以用DDD的四层架构思想继续进行优化抽象。
>
> 2、对比LocalAsyncBusiServiceImpl，你会发现dealMessageAsync的逻辑基本是一样的，私有的doBusi方法业务流程也基本是差不多的，只是寻找服务和调用服务的方法有点不一样。这也意味着，在这个具体的场景下，AsyncBusiService可以进一步的抽象细化。

当然要让NacosAsyncBusiServiceImpl能够生效，其他的一些框架接入的功能是需要你自己去补充的。包括：

1、引入nacos服务发现的依赖

```
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
        </dependency>
```

2、在application.yml中配置nacos服务地址

```
spring:
  cloud:
    nacos:
      discovery:
        server-addr: 192.168.65.206:8848,192.168.65.209:8848,192.168.65.210:8848
```

这往就完成了一个基于Nacos的服务发现机制，后端服务只需要按照要求往Nacos上注册服务即可。tlmall-openV2与后端服务完全解耦了。并且，由于电商项目的后端服务也是注册到这个Nacos上，所以，将电商的服务接入到tlmall-openV2就成了一件很简单的事情了。

留几个扩展的问题

对tlmall-open的改造，到这里就告一段落。但是课程结束了，并不意味着这个项目就结束了。我真正希望给你的，是一个活的项目，是一个不断演进，不断完善的项目。有些重复性比较高的工作，希望你能自己有时间去尝试完善。比如对mobiletag和mobilearea两个虚拟服务的四层架构改造，比如open-server中三个主要核心模块的微服务拆分。

另外，如果采用的是其他的RPC框架，比如Dubbo，要怎么实现？

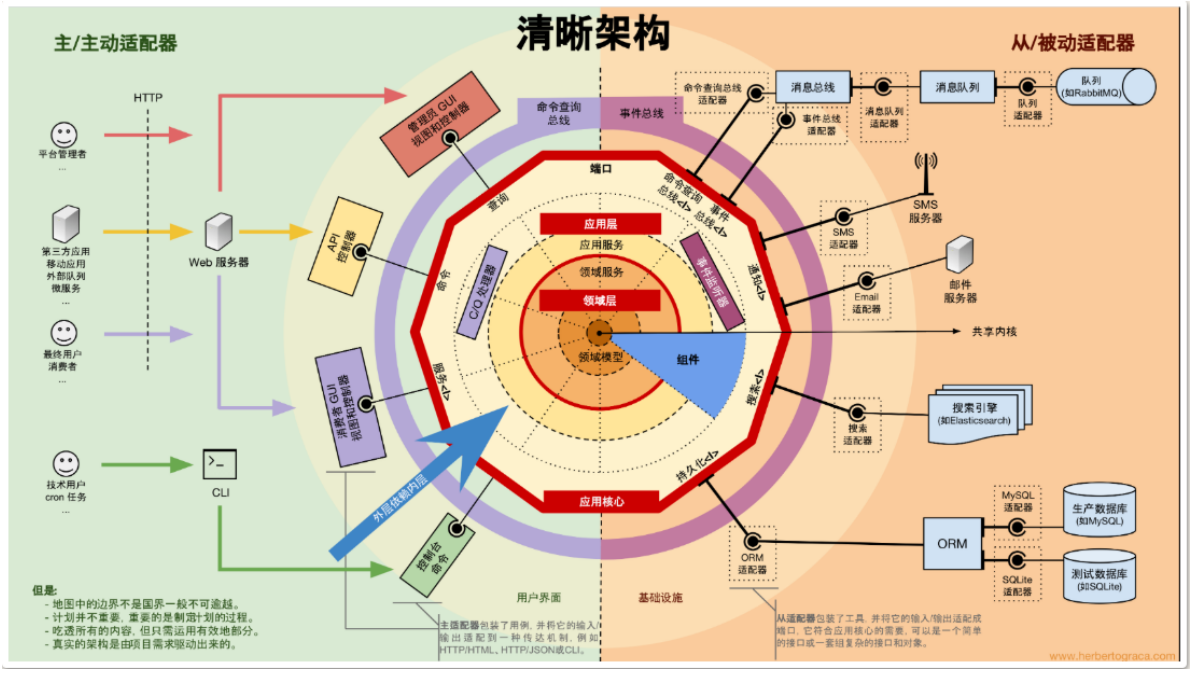
二、电商服务开放平台的战略演化路线分析

接下来我们按照DDD的理论路线，来继续讨论下tlmall-open如何向一个真正的互联网服务开放平台演进。后面提出的一些路线，希望你也能够自己思考下要如何进行完善。

1、构建更强大的领域模型

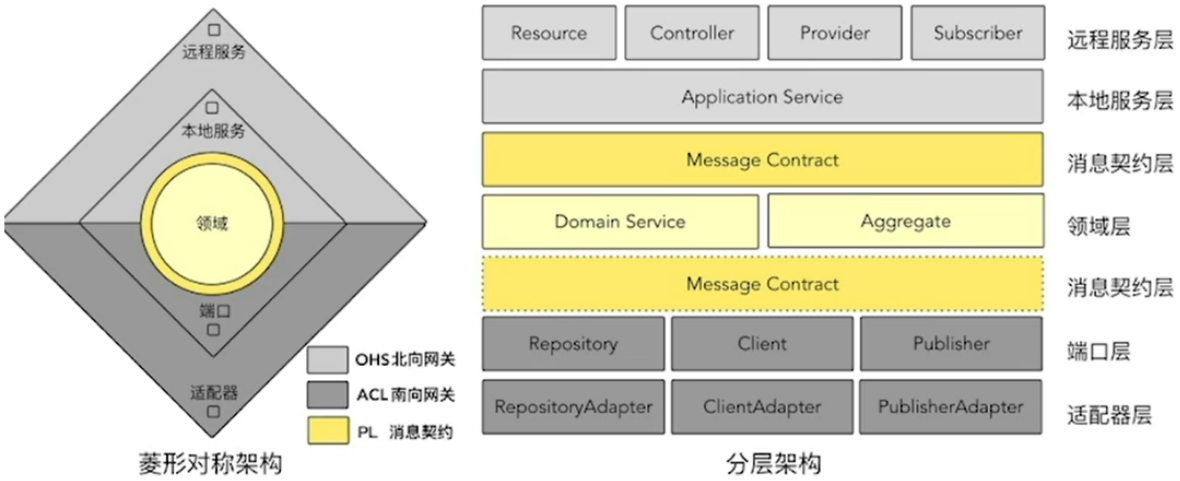
在之前的改造过程当中，对open-mobiletag服务做了诸多改革。你会发现，mobiletag这个虚拟服务变成了一个即支持单体接入标准，又支持微服务接入标准的复合型领域。那你有没有想过将这个特点放大化，能不能设想出一个可以支撑各种接入场景的通用型的领域？

这并不是突发奇想，而是早就有很多人想到过了。如果你喜欢没事的时候在网上逛逛技术贴，那么对于下面这个图应该不陌生。



这个图称为DDD的清晰架构。对于DDD的清晰架构，网上有很多不同的解读。我更倾向于是对DDD那些花里胡哨的洋葱架构、六边形架构、整洁架构等等架构概念的具象化分析。其核心，依然是要在复杂的业务场景下保持领域模型的稳定性。只不过，对业务场景的复杂性进行了具体的总结。将互联网中一些常见的领域接入场景进行总结并分类。

在DDD清晰架构中，对领域外部的业务场景进行了分类，大致分成了响应请求的主动适配器和响应结果的被动适配器。而这些适配器，都可以作为领域的应用层进行接入。这样，一个万能型的领域的雏形就很容易形成了。将应用层也按照需要适配的结构分成两类，一类是响应外部请求的北向网关，一类是响应结果的南向网关。整体就可以形成这样的领域结构：



> 对应之前的DDD实战，可以想到将adaptor层按照请求和响应进行分割，就能大致形成这样的雏形。至于其他的层，你可以自行想象一下。

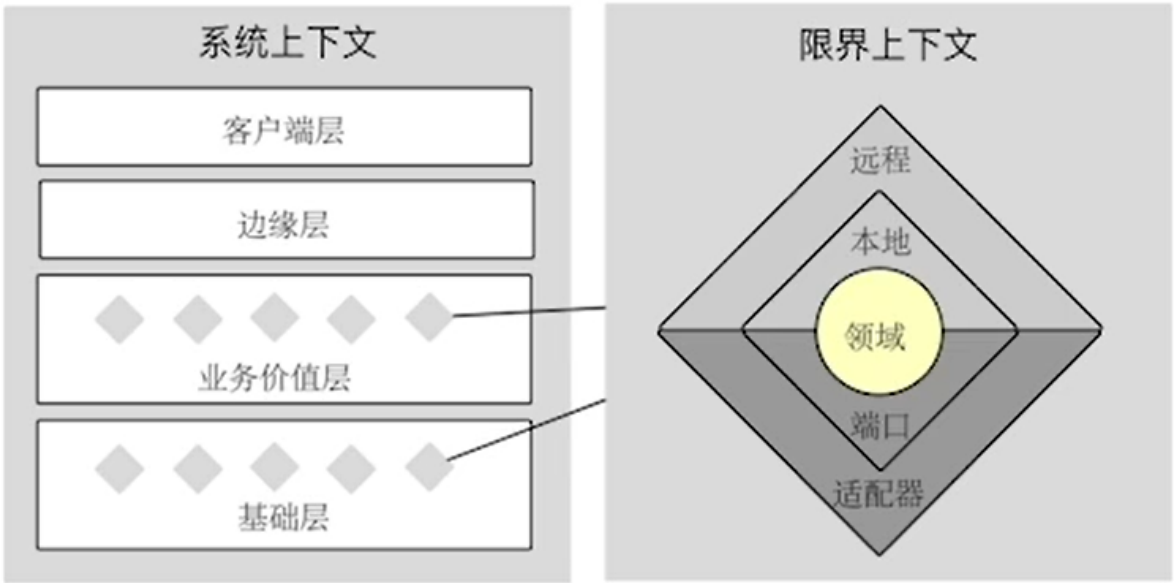
当然，其中还有很多的细节需要补充。在南向网关中，对接入的服务进行进一步分割，拆分成远程服务层响应各种跨进程的服务调用。远程服务层统一通过调用本地服务层中的进程内服务来调用，这样不管是RPC场景，还是单体架构的本地调用场景，北向网关都可以进行统一支持。南向网关也是类似的思路。

而在领域层与网关之间增加消息契约层，则是为了让领域层能够更加纯洁。在很多MVC项目的开发过程当中，会将传递数据的POJO，划分成DTO、VO、BO、DO等各种各样的Object，并且不断规定在每一个技术层面使用不同的Object，其目的也是为了尽量保证每一层的数据更纯洁，减少外部数据的影响。而消息契约层的作用跟这种方式的思想大致是相同的。在领域层与外部网关层之间，通过消息契约层，不光保证逻辑边界清晰，同时也保证数据边界也很清晰。这样这种领域模型就可以用来指导具体的应用开发了。

这种领域模型结构，其实你拿我们的实战项目，做一翻精益求精的改造，是不难形成这样的标准的。业界也已经有很多人在对这种领域模型结构进行研究。在《IDDD》中，这种领域模型就被称为是菱形结构。(还记得IDDD是哪本书吗?)

2、构建领域仓库，实现领域复用

接下来，你可以想象，如果有一个强大的企业，将企业内所有的应用都像tlmall-open一样完成了领域化改造，那么整个企业的所有业务能力就都成为了一个一个的领域。这些领域，即是一种业务能力，也是一个代码模块。将这些领域收集起来并集中管理，就可以形成一个领域仓库。而所有的应用，都变成了领域仓库里的领域的自由组合。



未来如果需要设计新的业务场景，也可以通过对领域仓库进行梳理，以搭积木的方式设计业务场景。在实施过程中，只要快速构建一个接入层即可。在技术层间，极大的减少了新业务的开发成本。

这种开发模式你可能很少参与，但是你绝对都能够听说得到。像阿里开发很多新的业务场景，比如无人超时、无人酒店等新业务时，整个软件方案并不需要重新构建，只需要做一个App的壳子，所有的后端能力都由现成的业务能力进行组合。这种方式，被称为中台战略。

3、理解DDD战略工具

是的，你能看到， DDD的未来与现在讨论不断的模糊的中台战略是有那么一丝丝的联系，其实这也是DDD非常火爆的一个原因。但是，这点似是而非的联系，并不能支撑DDD与中台的关联。实际上，将这几节课中介绍过的这些基础思想放大，在战略层面，DDD也形成了非常丰富的战略工具，能够用一种相对一致的思想，将整个企业的软件体系，从战略层面统一到战术层面。真正让每一个人都成为企业战略的一环，从而提升团队的凝聚力，全面提高企业的竞争力。

当然，企业战略层面的一些架构设计，在我们课上是接触不到的，所以，这里就不多做展开了。

总结

这一节课，我们最终完成了原本定制的小目标，将电商项目的后台服务面向互联网开放。中间颇多曲折，埋了很多坑，也填补了很多坑。有些坑是故意埋的，也有些坑确实是在开发过程中发现的，是非曲折，希望你能够形成自己的理解。

整个电商服务开放平台，希望带你理解的是软件进化的过程，而不是一个大团圆的完美结局。最后给出的tlmall-open其实也非常简陋，并不完美。但其实这不是关键，这就像你以后在工作中遇到的大部分企业真实项目一样，没有一个项目会是完美的。而通过DDD的实战演练，希望能够带你以发展的眼光看待项目。

DDD有很多具体的概念，工具，这些工具也跟DDD项目一样，是活的，是不断发展变化的。而按照DDD的思想来进行业务开发，实际上是希望将项目的架构设计能够和业务设计一样，分散到整个软件过程中，而不是在一开始就让架构师把架构全部做完了，后续只要填补CRUD就行了。在每次迭代的过程中，都可以对架构做一点点优化。也希望你能够掌握DDD的思想，在实现每一个功能，写每一行代码时，尝试让项目一点点的变好，你也能随着项目一起，慢慢的变强。