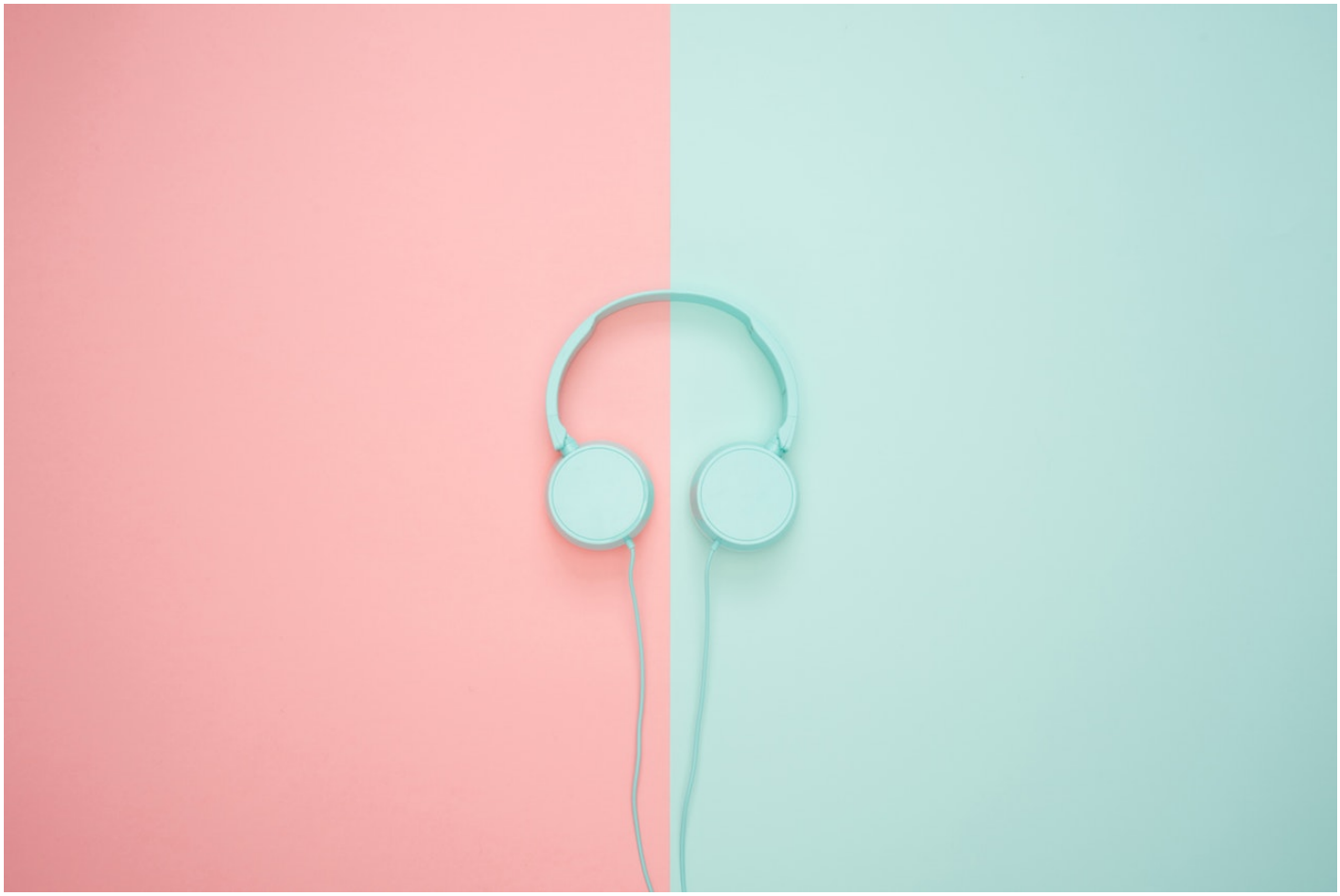
23讲实战一（上）：针对业务系统的开发，如何做需求分析和设计



对于一个工程师来说，如果要追求长远发展，你就不能一直只把自己放在执行者的角色，不能只是一个代码实现者，你还要有独立负责一个系统的能力，能端到端（end to end）开发一个完整的系统。这其中的工作就包括：前期的需求沟通分析、中期的代码设计实现、后期的系统上线维护等。

前面我们还提到过，大部分工程师都是做业务开发的。很多工程师都觉得，做业务开发没啥技术含量，没有成长，就是简单的CRUD，翻译业务逻辑，根本用不上专栏中讲的设计原则、思想、模式。

所以，针对这两个普遍的现象，今天，我通过一个积分兑换系统的开发实战，一方面给你展示一个业务系统从需求分析到上线维护的整个开发套路，让你能举一反三地应用到所有其他系统的开发中，另一方面也给你展示在看似没有技术含量的业务开发中，实际上都蕴含了哪些设计原则、思想、模式。

话不多说，让我们正式开始今天的学习吧！

需求分析

积分是一种常见的营销手段，很多产品都会通过它来促进消费、增加用户粘性，比如淘宝积分、信用卡积分、商场消费积分等等。假设你是一家类似淘宝这样的电商平台的工程师，平台暂时还没有积分系统。Leader希望由你来负责开发这样一个系统，你会如何来做呢?

你可能会说，只要产品经理给我产品设计文档（PRD）、线框图，我照着实现就可以了。我觉得，这种想法有点狭隘。我认为，技术人员应该更多地参与到产品设计中。在Google工作的时候，我很明显能感受到，Google工程师跟其他公司工程师有一个很大区别，那就是大部分人都具备产品思维，并不是完全的“技术控”。所以，Google很多产品的初期设计都是工程师来完成的，在产品发展壮大到一定程度的时候，才会引入产品经理的角色。

那你可能要问了，作为技术人，我该怎么做产品设计呢？首先，一定不要自己一个人闷头想。一方面，这样做很难想全面。另一方面，从零开始设计也比较浪费时间。所以，我们要学会“借鉴”。爱因斯坦说过，“创造的一大秘诀是要懂得如何隐藏你的来源”。你看大师都含蓄地表达了“借鉴”的重要性，我们也没有必要因为“借鉴”而感到不好意思了。

我们可以找几个类似的产品，比如淘宝，看看它们是如何设计积分系统的，然后借鉴到我们的产品中。你可以自己亲自用用淘宝，看看积分是怎么使用的，也可以直接百度一下“淘宝积分规则”。基于这两个输入，我们基本上就大致能摸清楚积分系统该如何设计了。除此之外，我们还要充分了解自己公司的产品，将借鉴来的东西糅合在我们自己的产品中，并做适当的微创新。

笼统地来讲，积分系统无外乎就两个大的功能点，一个是赚取积分，另一个是消费积分。赚取积分功能包括积分赚取渠道，比如下订单、每日签到、评论等；还包括积分兑换规则，比如订单金额与积分的兑换比例，每日签到赠送多少积分等。消费积分功能包括积分消费渠道，比如抵扣订单金额、兑换优惠券、积分换购、参与活动扣积分等；还包括积分兑换规则，比如多少积分可以换算成抵扣订单的多少金额，一张优惠券需要多少积分来兑换等等。

我刚刚给出的只是非常笼统、粗糙的功能需求。在实际情况中，肯定还有一些业务细节需要考虑，比如积分的有效期问题。对于这些业务细节，还是那句话，闷头拍脑袋想是想不全面的。以防遗漏，我们还是要有方法可寻。那除了刚刚讲的“借鉴”的思路之外，我还喜欢通过产品的**线框图**、**用户用例**（user case ）或者叫用户故事（user story）来细化业务流程，挖掘一些比较细节的、不容易想到的功能点。

线框图对你来说应该不陌生，我就不赘述了，我这里重点说一下用户用例。用户用例有点儿类似我们后面要讲的单元测试用例。它侧重情景化，其实就是模拟用户如何使用我们的产品，描述用户在一个特定的应用场景里的一个完整的业务操作流程。所以，它包含更多的细节，且更加容易被人理解。比如，有关积分有效期的用户用例，我们可以进行如下的设计：

- 用户在获取积分的时候，会告知积分的有效期；
- 用户在使用积分的时候，会优先使用快过期的积分；
- 用户在查询积分明细的时候，会显示积分的有效期和状态（是否过期）；
- 用户在查询总可用积分的时候，会排除掉过期的积分。

通过上面讲的方法，我们就可以将功能需求大致弄清楚了。积分系统的需求实际上并不复杂，我总结罗列了一下，如下所示。

1.积分赚取和兑换规则

积分的赚取渠道包括：下订单、每日签到、评论等。

积分兑换规则可以是比较通用的。比如，签到送10积分。再比如，按照订单总金额的10%兑换成积分，也就是100块钱的订单可以积累10积分。除此之外，积分兑换规则也可以是比较细化的。比如，不同的店铺、不同的商品，可以设置不同的积分兑换比例。

对于积分的有效期，我们可以根据不同渠道，设置不同的有效期。积分到期之后会作废；在消费积分的时候，优先使用快到期的积分。

2.积分消费和兑换规则

积分的消费渠道包括：抵扣订单金额、兑换优惠券、积分换购、参与活动扣积分等。

我们可以根据不同的消费渠道，设置不同的积分兑换规则。比如，积分换算成消费抵扣金额的比例是10%，也就是10积分可以抵扣1块钱；100积分可以兑换15块钱的优惠券等。

3.积分及其明细查询

查询用户的总积分，以及赚取积分和消费积分的历史记录。

系统设计

面向对象设计聚焦在代码层面（主要是针对类），那系统设计就是聚焦在架构层面（主要是针对模块），两者有很多相似之处。很多设计原则和思想不仅仅可以应用到代码设计中，还能用到架构设计中。还记得面向对象设计的四个步骤吗？实际上，我们也可以借鉴那个过程来做系统设计。

1.合理地将功能划分到不同模块

前面讲到面向对象设计的时候，我们提到，面向对象设计的本质就是把合适的代码放到合适的类中。合理地划分代码可以实现代码的高内聚、低耦合，类与类之间的交互简单清晰，代码整体结构一目了然，那代码的质量就不会差到哪里去。类比面向对象设计，系统设计实际上就是将合适的功能放到合适的模块中。合理地划分模块也可以做到模块层面的高内聚、低耦合，架构整洁清晰。

对于前面罗列的所有功能点，我们有下面三种模块划分方法。

第一种划分方式是：积分赚取渠道及兑换规则、消费渠道及兑换规则的管理和维护（增删改查），不划分到积分系统中，而是放到更上层的营销系统中。这样积分系统就会变得非常简单，只需要负责增加积分、减少积分、查询积分、查询积分明细等这几个工作。

我举个例子解释一下。比如，用户通过下订单赚取积分。订单系统通过异步发送消息或者同步调用接口的方式，告知营销系统订单交易成功。营销系统根据拿到的订单信息，查询订单对应的积分兑换规则（兑换比例、有效期等），计算得到订单可兑换的积分数量，然后调用积分系统的接口给用户增加积分。

第二种划分方式是：积分赚取渠道及兑换规则、消费渠道及兑换规则的管理和维护，分散在各个相关业务系统中，比如订单系统、评论系统、签到系统、换购商城、优惠券系统等。还是刚刚那个下订单赚取积分的例子，在这种情况下，用户下订单成功之后，订单系统根据商品对应的积分兑换比例，计算所能兑换的积分数量，然后直接调用积分系统给用户增加积分。

第三种划分方式是：所有的功能都划分到积分系统中，包括积分赚取渠道及兑换规则、消费渠道及兑换规则的管理和维护。还是同样的例子，用户下订单成功之后，订单系统直接告知积分系统订单交易成功，积分系统根据订单信息查询积分兑换规则，给用户增加积分。

怎么判断哪种模块划分合理呢？实际上，我们可以反过来通过看它是否符合高内聚、低耦合特性来判断。如果一个功能的修改或添加，经常要跨团队、跨项目、跨系统才能完成，那说明模块划分的不够合理，职责不够清晰，耦合过于严重。

除此之外，为了避免业务知识的耦合，让下层系统更加通用，一般来讲，我们不希望下层系统（也就是被调用的系统）包含太多上层系统（也就是调用系统）的业务信息，但是，可以接受上层系统包含下层系统的业务信息。比如，订单系统、优惠券系统、换购商城等作为调用积分系统的上层系统，可以包含一些积分相关的业务信息。但是，反过来，积分系统中最好不要包含太多跟订单、优惠券、换购等相关的信息。

所以，综合考虑，我们更倾向于第一种和第二种模块划分方式。但是，不管选择这两种中的哪一种，积分系统所负责的工作是一样的，只包含积分的增、减、查询，以及积分明细的记录和查询。

2.设计模块与模块之间的交互关系

在面向对象设计中，类设计好之后，我们需要设计类之间的交互关系。类比到系统设计，系统职责划分好之后，接下来就是设计系统之间的交互，也就是确定有哪些系统跟积分系统之间有交互以及如何进行交互。

比较常见的系统之间的交互方式有两种，一种是同步接口调用，另一种是利用消息中间件异步调用。第一种方式简单直接，第二种方式的解耦效果更好。

比如，用户下订单成功之后，订单系统推送一条消息到消息中间件，营销系统订阅订单成功消息，触发执行相应的积分兑换逻辑。这样订单系统就跟营销系统完全解耦，订单系统不需要知道任何跟积分相关的逻辑，而营销系统也不需要直接跟订单系统

交互。

除此之外，上下层系统之间的调用倾向于通过同步接口，同层之间的调用倾向于异步消息调用。比如，营销系统和积分系统是上下层关系，它们之间就比较推荐使用同步接口调用。

3.设计模块的接口、数据库、业务模型

刚刚讲了模块的功能划分，模块之间的交互的设计，现在，我们再来看，模块本身如何来设计。实际上，业务系统本身的设计无外乎有这样三方面的工作要做：接口设计、数据库设计和业务模型设计。这部分的具体内容我们放到下一下节课中跟实现一块进行讲解。

重点回顾

今天的内容到此就讲完了。我们来一块总结回顾一下，你需要掌握的重点内容。

技术人也要有一些产品思维。对于产品设计、需求分析，我们要学会“借鉴”，一定不要自己闷头想。一方面这样做很难想全面，另一方面从零开始设计也比较浪费时间。除此之外，我们还可以通过线框图和用户用例来细化业务流程，挖掘一些比较细节的、不容易想到的功能点。

面向对象设计聚焦在代码层面（主要是针对类），那系统设计就是聚焦在架构层面（主要是针对模块），两者有很多相似之处。很多设计原则和思想不仅仅可以应用到代码设计中，还能用到架构设计中。实际上，我们可以借鉴面向对象设计的步骤，来做系统设计。

面向对象设计的本质就是把合适的代码放到合适的类中。合理地划分代码可以实现代码的高内聚、低耦合，类与类之间的交互简单清晰，代码整体结构一目了然。类比面向对象设计，系统设计实际上就是将合适的功能放到合适的模块中。合理地划分模块也可以做到模块层面的高内聚、低耦合，架构整洁清晰。在面向对象设计中，类设计好之后，我们需要设计类之间的交互关系。类比到系统设计，系统职责划分好之后，接下来就是设计系统之间的交互了。

课堂讨论

对公司业务及已有系统的熟悉程度，有时候甚至会超过个人的技术能力，更能决定一个人在公司内部的发展前途。但是，当我们出去面试的时候，面试官大部分情况下更加关注你的技术能力，而非特定的业务细节，特别是你做的业务并不是太复杂，或者跟要面试岗位无关的时候。

这两者听起来比较矛盾。作为一名技术人，为了谋求更好的发展，你觉得是应该多花点时间研究业务呢，还是要多花点心思在技术上呢?

欢迎在留言区写下你的答案，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

精选留言

沉淀的梦想

个人觉得，如果对当前工作不满意，想要找更好的工作的话，应该多花点时间在技术上，但是如果对当前工作很满意，想要继续在当前岗位上发展的话，还是应该多花时间在业务上

2019-12-25 01:11

李小四

设计模式_23

这个问题太有感触了，不熟悉业务当然做不好系统，但业务理解的深度与技术能力并没有直接的对应关系。

我们先讨论一下另外一个问题：

技术人员创造的价值是什么呢？我们的技术最终体现某一个商业的业务上，解决用户的问题，从而创造价值。我们去面试，去找新的工作机会，目的是更好地实现自己的价值，目的并不只是换个涨点工资的工作。比如王争老师，现在也是在教育的这个具体的业务上闪闪发光。

这里引用一下吴军老师的“五级工程师模型”

> 第五级：能独立解决问题，完成工程工作；
> 第四级：能指导和带领其他人一同完成更有影响力的工作；
> 第三级：能独立设计和实现产品，并且在市场上获得成功；
> 第二级：能设计和实现别人不能做出的产品，也就是说他的作用很难取代；
> 第一级：开创一个产业

可以看到，在更高级别的工程师中，对于产品和商业的理解变成了一种要求，因为他是技术与现实生活的桥梁。
2019-12-25 18:13

黄林晴
首先我非常赞成，技术人也要有产品的思维，但是悲催的是，对我这种喜欢和产品经理讨论需求的技术人其实说多了怕是觉得找借口……

课堂讨论:
一般面试的时候都是注重技术，但是进去公司后都是以业务为核心，我一直觉得程序猿甚至公司的竞争核心不止于技术上，如果你看好公司前途想脚踏实地干一番，那么久好好熟悉公司业务系统，如果觉得想找更好的出路就多学技术争取很好的出路✔
2019-12-25 00:32

下雨天
相辅相成，缺一不可！如果说业务是场景，技术是招式，没有场景的招式是无意义的，没有招式的场景是苍白的！不同场景下选用有效招式才是最终目的！
2019-12-25 08:03

阿玛铭
不懂业务（需求）是做不好业务系统的，不懂技术是做不好软件系统的。重技术作为底层通用模块，在跳槽的时候可以发挥作用，让自己的选择面更广，但也是有代价的，意思是如果跳到不同行业，以前的业务积累就没有意义了。工作性质决定，我目前和以后业务、技术时间精力投入占比分别是20、80。终身学习是人无法避免的趋势，为了增加个人综合能力和生存韧性，还会投资一些非IT相关的通用软技能。
2019-12-25 08:04

Chen
技术能力是敲门砖，是比业务能力更下层的东西，扎实的技术能力能应用于各种业务场景。我认为应该优先提升技术能力，当然当你有技术话语权的时候业务能力能够让你如虎添翼！
2019-12-25 07:53

业余爱好者
永远不能忘了我是谁。我是一个程序员，技术是我吃饭的家伙，是我赖以生存的东西。无论何时我都选技术，除非我转行。
2019-12-25 07:30

debug
两手都要抓，两手都要硬
2019-12-25 15:34

守拙
课堂讨论Answer：

钻研业务与发展技术的取舍（或精力分配）取决于其风险与收益。

如果公司基本面良好，业务蒸蒸日上，我个人受到领导的认可，同事的尊敬，则倾向于将更多的精力钻研业务，提升个人在公司的不可替代性。

如果公司基本面恶化或前途未卜（比如大多创业型公司），领导本领低微或不重视我的意见，和同事相处一般，则应将大量精力分配在技术精研上，提升个人在就业市场上的竞争力。

最后，钻研业务和提升技术并不是一枚硬币的两面，如果一些事情上同时满足两种需求，就再好不过。

2019-12-26 16:28

Jxin

课后讨论

1.业务是专业资源（复用受限），技术是通用资源（复用不受限），所以业务知识的沉淀对于个人是一个风险比较高的投资，所以大部分人本能的就会讨厌这种受制的局势，进而就不喜欢在业务上投入太多的精力。

2.其实我觉得很遗憾，大部分程序员会规划自己的技术栈却不关心干什么活（做什么业务）。当然，业务知识不好考核，技术知识好考核，所以技术知识在面试吃香也是必然。不过可以遇见的情景是：招了个技术牛逼的人，但短时间他产生不了什么价值（需要了解业务），等到他吃透业务（半年甚至一年），他的技术水平已经又上了一个台阶，面临新一轮人员调动的开始。结果就是，一个技术很牛逼的人，在我这个项目组干活期间，不显山不露水，资深的水平干着高级的事，我还知道这合情合理，等他开始发挥资深水平价值时，人走了。

3.产品技术其实是一体两面，如果有产品思维，那么业务领域的选择和技术栈的选择其实是绑在一起的。走在自己想干的业务领域，自己的技术栈也跟着这个领域的需求去发展，如此以来每份工作的业务积累都能为下份公司带来大量点子和突破口。但这样就会出现，高级的水平干架构师的事，但一般没公司为你带来的“架构师级价值”买单。目前的情况是欢喜的接受你带来的改变，但并不会为你加薪（因为你的技术水平在市场只值这么多，公司给你的钱会尽量平衡在市场线，而非你真正的价值）。这种病态得不到改善，程序员不重视业务就得不到变革。所幸当下，跑马圈地已过，抢人已经不再疯狂，招聘方的招聘目标也在慢慢从技术牛逼到实际价值转变。

2019-12-25 13:15

明翼

我曾经做过几年的项目经理，对于业务已经非常熟悉了，后面公司转型，以前的业务知识完全清零了。所以我认为如果作为乙方的程序员技术能力更重要，决定了你的职业选择广度，业务也要懂，只是不要把大精力花在上面。如果你是甲方的程序员，而且不会跳槽国企的那种，想成为行业专家，那业务更重要，技术很多可以速成或招聘，但是业务熟悉的人想找不好找，除非在同类竞争公司去挖。

2019-12-26 20:01

刘大明

我觉得技术是服务于业务的。毕竟纯粹的技术驱动型公司没有那么多。课堂讨论题目是到底是业务多花点时间还是技术多花点时间。我觉得应该是相辅相成的，比方说现在公司要做积分系统，那么你要知道这个系统怎么设计，用哪些技术。如文中提到的消息队列，那么就要研究目前有消息队列，各个消息队列的优缺点，以及适用场景和规模等。结合公司业务来最终取舍用哪个消息队列。

2019-12-26 19:30

尘封

业务细节和技术个人能力不冲突

理解业务细节可以能好的看懂代码

个人能力越高对业务细节可以写出更好的代码

2019-12-26 15:56

WL

请问一下老师为什么上下层系统之间的调用倾向于通过同步接口，同层之间的调用倾向于异步消息调用？是因为上层系统依赖下层系统的执行结果而执行后续逻辑所以用同步调用，同层系统中没有这种依赖所以用异步调用吗？请问这么理解对吗？

2019-12-26 09:49

大饶Raysir

技术是基础，也是程序员市场的硬通货，这条路走下去可以选择架构师、技术总监；业务能在公司或团队内持续发展、转型产品项目经理、晋升团队leader，也是一条康庄大道，看性格咯~

2019-12-25 08:42

Geek_1e684b

可否多解释上下层系统之间跟同层之间的差异

举例

- 买家付款完成订单后，要开发票是哪一种呢?

-- 完成订单后一定要开发票，所以是上下

-- 完成订单跟发票属于不同业务性质，所以是同层
不知道哪种解释才对，谢谢
2020-01-04 15:45

Rain
我怎么感觉老师提的话题有点跑偏？还是好好想一想如何做好设计模式比较合理对吗？另外，老师本课带的思路也有点偏差我觉得，毕竟我们要设计的是一个积分模块而不是一个淘宝系统。架构和设计都是有演变过程的，个人认为任何撇开并发与用户数而不谈的需求是耍流氓。此处也应该循序渐进的讨论设计方而不是如此的过度设计，谢谢。
2019-12-27 09:04

作者回复
哪里过度设计了，麻烦指出来呢
2019-12-30 08:44

辣么大
积分系统设计第一种方案和争哥第12节实战的虚拟钱包设计思路很像呀！
2019-12-25 15:40

南山
个人更倾向于技术，但是并不表示不需要花心思在业务上，技术就是用来解决业务需求的，业务的复杂度也能反过来推动个人技术的成长，再总结，提炼出自己的套路，也就是经常挂在嘴边的常用架构设计方案。
做得多，技术储备丰富，应对新的业务也就更有思路和合理的技术方案
2019-12-25 12:22

Geek_862694
30岁之前多研究技术，30岁以后如果技术上没有什么特别突出的地方就多花点时间在业务上
2019-12-25 11:10