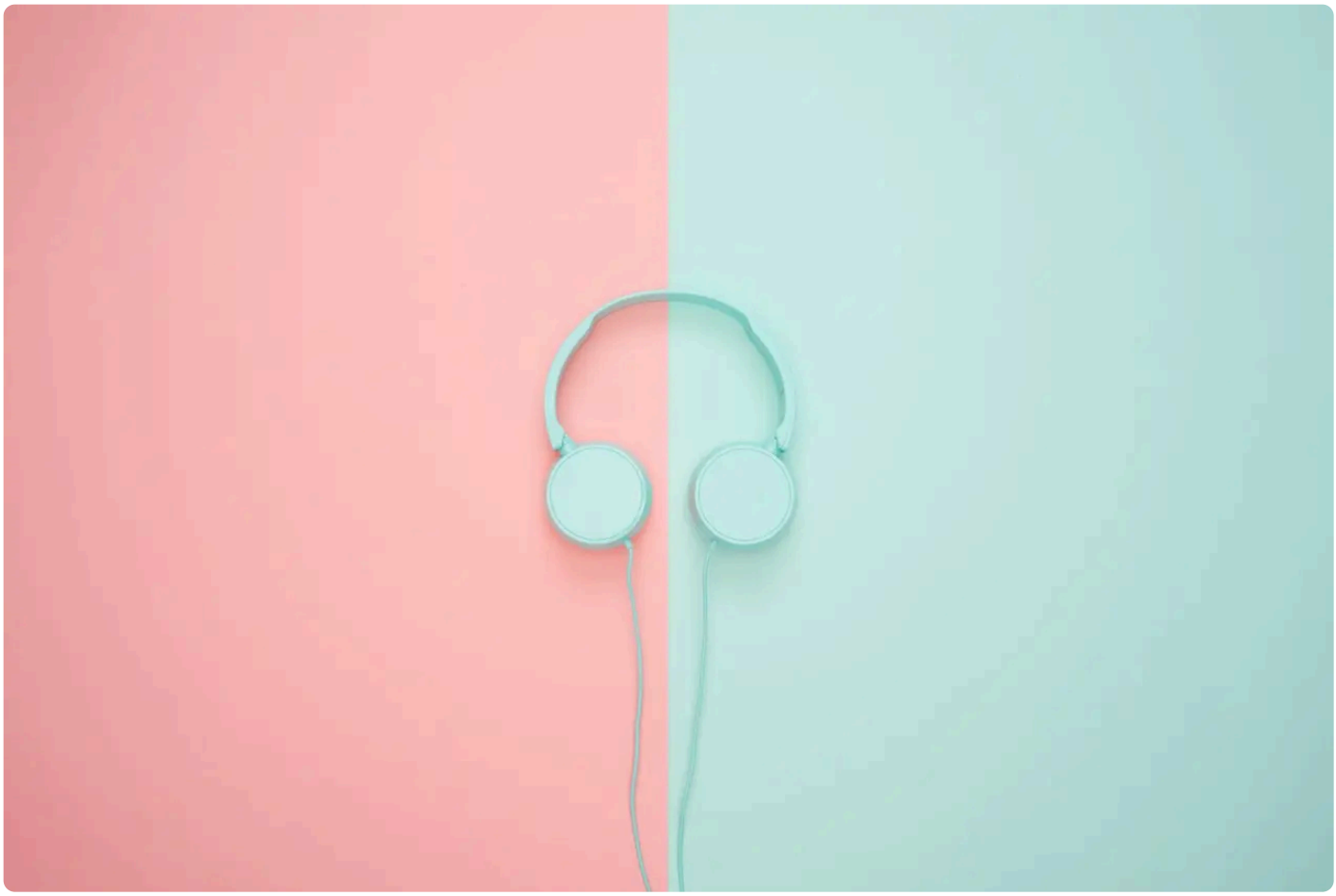
23 | 实战一（上）：针对业务系统的开发，如何做需求分析和设计？

王争 · 设计模式之美



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。

AI智能总结

本文介绍了业务系统的需求分析和设计，强调了技术人员在业务系统开发中的重要作用，以及需求分析和系统设计的关键性。作者通过一个积分兑换系统的开发实战，展示了从需求分析到上线维护的整个开发套路，并强调了在业务开发中蕴含的设计原则、思想、模式。在需求分析阶段，建议技术人员更多地参与产品设计，并提出了借鉴其他产品、细化业务流程的方法。在系统设计阶段，强调了面向对象设计的四个步骤可以借鉴到系统设计中。此外，文章还介绍了模块划分、模块间交互关系设计以及模块的接口、数据库、业务模型设计。总的来说，本文强调了技术人员在业务系统开发中的重要作用，以及需求分析和系统设计的关键性。

全部留言 (213)

最新　精选

未来小娃
2020-04-20

聊聊技术与业务

最近看了一些关于程序员价值方面的文章，很多人包括我自己有个疑问，面试时只看技术，进公司后做业务，两者矛盾么？这个问题的本质是哪个更有价值。如果你不喜欢做业务那就往中间件方向发展，做出类似高性能高可用的中间件不失为值得骄傲的事情。那么业务呢，要清楚的是公司的现金流都是来源于业务，业务产生价值。那么两者的关系是公司依赖业务，业务开发需要技术，也就是技术是更底层更通用的能力，所以自然也就更好评价，业务各不相同复杂度和场景天差地别不可复用，自然就没有比较好的评价标准。技术能力的高低决定了业务质量，而业务质量的高低反过来影响你的技术，业务复杂度上升了，之前的技术可能就不满足了，这也是技术不断更新的源动力，无法适应业务的技术是没有价值的，那么到底是什么造成了程序员价值的高低呢？我认为有以下几点：

第一，专业能力。虽然技术不断更新，但是底层的基础技术却没变，我们应该花更多时间掌握好底层技术，不变应万变。基本功扎实了才能走得更远
第二，落地能力。有了基本功后我们就具备完成一件开发任务的能力，但是怎么把事情做好却是需要不断提升的，结果导向，积极主动，有想法能落地才能体现你的价值。记住，当你开始积极主动思考一些事情后，你已经领先了大部分人
第三，终身学习能力。公司要持续发展，业务就会不断迭代发展，要让自己的技术能够匹配上业务的发展就需要不断学习，扎扎实实在自己想要专研的领域做精，你的价值也就体现出来了。
总结下：技术是业务的支撑，业务反过来影响技术，业务发展就需要提升技术。

作者回复: 👍

共 8 条评论>

368

Dawn
2020-11-17

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，依据是什么?

作者回复: 主要目的是解耦。上下层没有互相调用，只有上层调用下层，调用关系简单。同层之间互相调用，错综复杂，使用中间件异步消息解耦，让调用关系简单。

共 3 条评论>

25

姜玮
2020-11-14

课堂练习的思考，技术和业务好比生产力和生产关系的相互作用。大体上来讲，一定是相辅相成，互相作用的。当一个技术人员没有办法完成业务需求时候，那么，技术就显得更重要一些; 而当技术人员发现业务的指标都做完了，想要把业务做得更好，那么就要去深入理解业务，才能在通用技术之上构建出更贴合业务的优秀系统。

生产力，决定了整个社会价值形态的下限，而生产关系的优劣，可以影响在当前生产力局面下，社会价值形态的上限。

因此，技术可以解决温饱问题，业务可以带领走上上小康道路。

作者回复: 嗯嗯，说的好

9

蛀牙
2020-08-26

"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。"

老师或者哪位大神可否就这句话展开讲讲，为什么异层同步，同层异步？

作者回复: 同层之间一方面会有互相调用的情况，另一方面从业务上来讲有平行关系，如果设计成调用，可能存在互相调用，调用关系交叉复杂，使用消息中间件，实际上就能将网状调用关系，解耦为星状调用关系，调用关系更加简洁清晰。而且，使用消息中间件，更能体现同层之间的平行关系。

9

自然
2020-06-25

模块的划分我理解第三种也是合理的，因为积分的所有配置和规则维护应该都有积分系统完成，符合单一职责，订单、评价系统只需要发事件消息出来就可以了，他们不需要关心积分的业务逻辑，积分系统收到消息后去处理积分内部的业务逻辑。
这块不太清楚，老师怎么看？

作者回复: 实际上，哪种方式都可以，判断标准是：

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。

5

Rain
2019-12-27

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。

作者回复: 哪里过度设计了，麻烦指出来呢

共 5 条评论>

1

dxy_123456
2020-11-25

在一家国企工作，领导是个注重员工培养的人，每天都会花半个小时学习UX、产品经理相关的课程。期初都是不感兴趣的，觉得和自己无关，但今天在老师的讲课中看到一些UX相关的知识，还是非常惊喜。我觉得多学习会发现很多相关领域的知识都是可以串起来的，技术人员当然重点还是多钻研技术，但既然在公司也做业务相关的工作，为何不再花一点时间把它做得更好呢。

作者回复: 嗯嗯 ❖❖❖❖❖❖

Jevan Wu
2020-11-15

"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。" 请问，这点是基于什么样的考虑，我理解应该需要同步的才用接口，其他都用异步的方式就好了吧?

作者回复: 可以看看我对其他人的回复

Single
2020-08-02

为什么同层推荐消息，上下层推荐同步接口调用吗?

作者回复: 同层之间可能存在交叉调用，用消息中间件，不会出现很混乱。上下层不会有交叉调用，上层调用下次，用接口就可以了

小川
2020-06-14

老师好，请教您一个问题，针对这个案例，我有一个困扰，数据库是否要维护 用户总积分这个字段，还有就是积分过期这种是如何实现方式比较好呢?

感谢老师的回答!

> 作者回复: 可以维护，也可以不维护。维护的好处是不用每次都计算，但需要处理数据一致性的问题（每条积分记录和总积分之间）。如果积分表数据量不大，不维护也可以，如果积分表数据量很大，查询总积分很慢，就要维护。