04 | 领域驱动设计（上）：如何设计金融软件顶层架构？

2020-12-28 任杰

分布式金融架构课 进入课程 >

讲述：任杰
时长 22:59 大小 21.05M

你好，我是任杰。这节课我想和你聊一聊如何设计顶层金融软件架构。

通过前面三节课的学习，你应该感觉到金融业务比较复杂，对应的金融软件也很复杂。那想让业务和架构设计良好配合，就是一个非常有挑战性的事情了。这也是软件工程要解决的问题。

软件工程的教材里一般不会写项目怎么成功，而是记录项目怎么失败。金融行业对信息化要求很高，那在实际解决问题时，金融行业自然也会经历很多失败的软件项目。

幸运的是，错的次数多了也渐渐总结出一些比较好的方法论，本世纪初出现的**领域驱动设计**就是其中一个。一些大型的金融公司已经用这个方法，从战略上重新设计了顶层框架和运作机制，也取得了不错的效果。

想更好地掌握这种方法，你就要搞清楚它的适用范围以及整体思路，这样才能在后续实践中更好地使用。话不多说，接下来我会从三个角度入手，带你理解领域驱动设计理论中的核心知识点。

领域驱动设计侧重点

顾名思义，领域驱动设计（**Domain Driven Design**，简称 **DDD**）指的是针对特定领域进行定制化的设计。

从**空间**上来讲，它看到的是整个行业或者整个领域。从**时间**上来讲，它看到的是软件从发生、发展到消亡的整个生命周期。从**角色**上来讲，它看到的是业务、产品、开发和运维等所有参与人员的合作。

因此领域驱动设计是战略上的思路，而不是战术上的实操。这里你要抓住一个重点，**领域驱动设计解决的是复杂问题。**

虽然领域驱动设计是从宏观角度解决问题，但是它非常务实，主要体现在以下两个方面，一方面是**注重投资回报比**（Return On Investment，ROI），另一方面是**做长期优化**。

注重投资回报比

我们想做好领域驱动设计，需要多角度思考问题，还需要对各方面因素做一些平衡，所以这个设计本身是一个非常耗时耗力的过程。

如果你投入了大量的人力物力和时间，却没有收到相应的成果，就有点得不偿失了。这种情况下就不推荐使用领域驱动设计了。

具体来说，适合使用领域驱动设计的系统需要具备这些特点：

1. 系统组件足够多。

2. 业务逻辑足够复杂。

3. 软件生命周期长。

总之一句话，简单问题不要用领域驱动设计。金融软件恰好满足了上面三个要求，因此它也是一个领域驱动设计的标志性应用场景。

长期优化

你也许听过一些国外的真实故事，讲的是金融系统完成开发之后，毫无问题地运行了几十年。当它突然出了问题时，还需要把早已经退休的开发人员反聘回来解决。

这些故事告诉我们，金融系统可能会存在很久，可能所有最初的相关人员都已经去做其他项目了，这个系统还需要有人继续维护。

这说明软件的生命周期可能超过个人的项目周期。如果维护的人换了，就有可能出现经验知识的传承问题。领域驱动设计尝试解决这些长期沟通问题，降低整个软件生命周期中的沟通成本，这样就能降低项目失败的可能性。

了解了这两个重点之后，我们还要搞清楚具体应该怎么做。

我们在解决问题的时候一般顺序是先搞定人，再做事。所以接下来我们先看看怎么设计人员组织架构，然后再考虑怎么设计系统组织架构。

人员组织架构

我们都知道，软件开发过程中涉及到三类人，第一类是业务方；第二类就是系统分析人员，包括产品经理和架构师；还有一类是系统开发人员。

相应地，软件开发流程有 4 个步骤，估计你也不陌生：

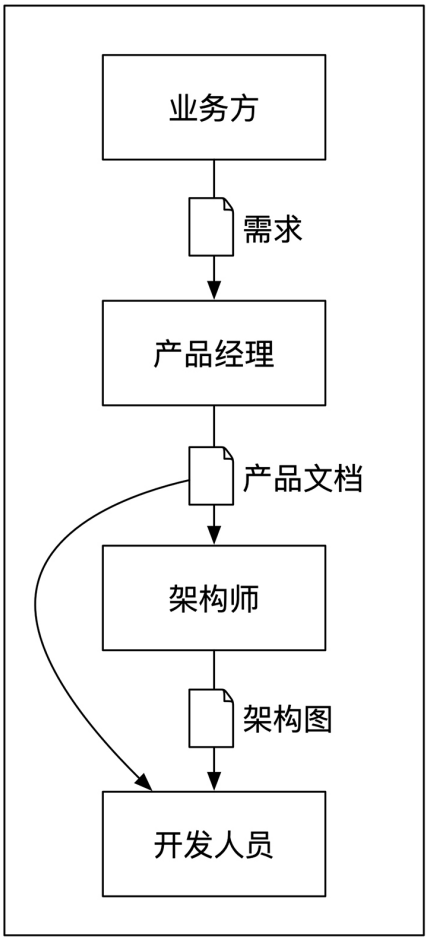
1. 业务方提出需求。

2. 产品经理分析需求。

3. 架构师根据产品经理的分析做出合理的架构设计。

4. 开发人员按照产品文档和架构设计文档来进行开发。

开发流程示意图如下：



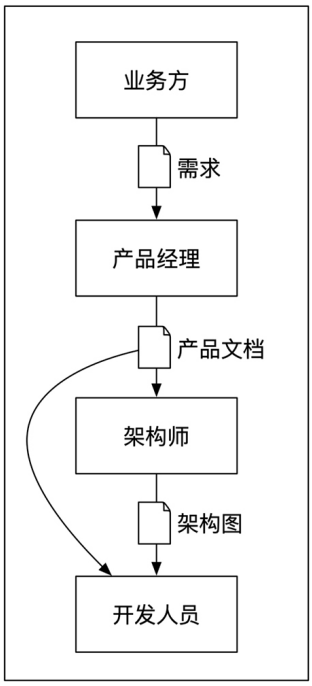
常见沟通方式

这个流程看上去很好，但是在实际操作中会有一些问题。比如当开发人员发现文档细节不够全面，甚至内容有误怎么办呢？这时候，沟通只能原路返回，等产品或者业务方再次确定清楚后，再从上往下传递消息。

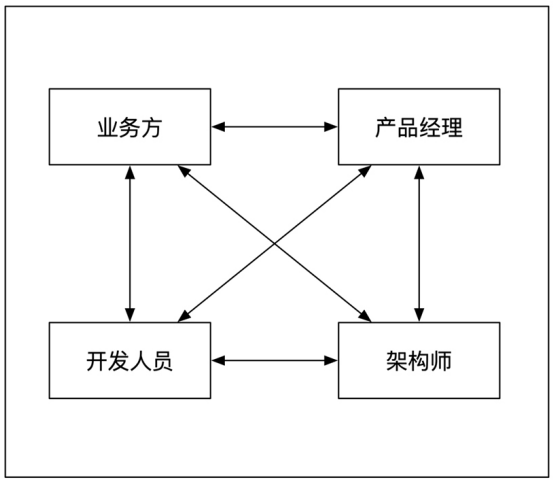
我们尝试从宏观角度分析这个沟通问题。软件开发过程中不同角色之间的沟通，其实是一个内容翻译的过程。由于每一步沟通都会造成信息损失，沟通的流程越长，信息损失越大。

那怎么解决沟通中的信息丢失问题呢？领域驱动模型提了一个根本的解决方案。既然沟通的流程长了会丢失信息，那么最优化的沟通方式不就是只有一个层级的沟通吗？

所以领域驱动设计取消了常见的垂直沟通方式，变成小组沟通方式。**每次沟通都是所有相关人员参与。**下图列出了这两种不同的沟通方式：



常见沟通方式



推荐沟通方式

表面上这只是沟通方式上的一点小小的改变，但其实是一个**文化的转变**，时间长了会带来一些根本性的变化。

第一个变化是**提高了决策的速度。**在小组沟通的模式下，问题能直接从发现的人手上交给解决问题的人，不再需要层层分析，层层审批。

第二个变化是**打破了部门壁垒。**我们在开篇提到了领域驱动设计需要有一定的业务复杂度。复杂业务会有自己独有的专业术语。

金融行业充满了生僻的术语，比如说中文的有融资融券、内保外贷、永续债券等等，英文的有 DCF、long/short、cap/floor、Snowball、Butterfly 等等。不知道这些术语你都了解多少。开发人员想要正确地进行系统建模，前提就是先对这些术语有正确的了解。

那怎样才能了解呢？答案很简单，就是**直接参与。**开发人员需要直接和业务方沟通，甚至直接参与业务。同样，业务方也很难理解开发人员的术语，也需要通过直接沟通来了解一些基本的行业逻辑和假设。

这个沟通过程中，会沉淀下来对专业术语的共同理解，这就是**领域语言**（Domain Language）。有了领域语言之后，各方的沟通再也不是鸡同鸭讲，而是永远在一个频道上。

第三个变化是**弱化了产品经理的角色。**产品经理负责收集业务问题，分析后将需求交给开发人员。那问题来了。谁对业务最了解呢？在金融行业，开发人员才是最了解金融业务的人。这一点可能会超出你的想象，但是事实如此。

金融行业讲究细节，每一个犄角旮旯的情况都要考虑清楚。开发人员在写程序的过程中，需要对金融业务的正常和异常情况进行极为完整的分析，所以不得不深入了解金融业务。这也是为什么在一些大型的金融公司里很少有产品经理这个岗位。

总结一下：**领域驱动设计建议软件的所有参与方之间能以小组的形式进行直接沟通。开发要懂业务，业务方和产品也要懂技术。沟通的结果是形成一个大家都能认同和理解的领域语言。**

系统组织架构

人的问题解决了，我们再来看看怎么解决事情。当你接手了一个复杂问题，第一步不是想怎么去解决它，而是应该仔细分析它为什么复杂，复杂在什么地方。

经验告诉我们，一个复杂的软件系统也不是所有地方都复杂，各个组件的复杂度和重要程度都不一样。所以我们可以像庖丁解牛一样，对系统组件进行分解，然后把相同的归类，统一对待。

领域驱动设计提供了一个分解问题的思路。领域驱动设计将所有业务领域划分为三大类型：**核心领域、通用领域以及支持型领域。**不同领域有不同的处理态度。那我们来逐一讲解一下每个类型的划分标准。

核心领域

金融系统软件有一个特点是所有组件看起来都很重要，就连最基本的短信通知功能都不能出问题。这就是我们之前提过的，只要是和钱相关的都是大事情。

但是你明显能感觉到，组件之间的重要性还是有微妙的区别。比如说，同样是对外发消息的网关组件，券商对接交易所的网关明显比短信通知网关更重要。那你的这种“感觉”究竟是从哪里来的呢？

一个软件组件是否重要，这取决于它所属的业务是不是核心业务。而业务是不是核心又取决于竞争对手。从整个金融行业的角度来看，一家金融公司如果能存活下来，一定需要有比其它类似金融公司做得更好的地方，也就是它的竞争优势。

举个例子。上世纪 70 年代股票交易量大增。当时一台大型计算机的价格简直就是天价。一些有远见的金融公司花了巨额资金，买了这些计算机来处理股票交易记录。

很快，没这么做的人就因为无法处理大量交易而淡出市场。他们退出不是因为业务方向错误，而是因为在没有计算机辅助的情况下，服务质量下降。这时候的金融行业的核心竞争力就是电算化，股票交易处理就是核心业务领域，交易清结算组件就是核心组件。

到了 21 世纪初，衍生品定价模型逐渐成熟了，谁计算金融风险的速度越快，结果越准确，谁的盈利就会越高。

在 08 年经济危机的时候，有的金融公司计算风险的速度快，就能提前逃离市场。如果你计算得慢，就得被迫吞下有毒资产。这时候的金融行业的竞争优势就是风险的计算和对冲，风险计算就是核心业务领域，风险计算组件就是核心组件。

所以，你可以这样理解，**一个业务是否是核心业务，取决于它是否能给公司带来行业内的竞争优势。**

我结合自己的经验，给你梳理了核心领域的三个要点，分别是资源分配、审时度势以及宏观视角。

资源分配

核心竞争力是买不来的。所以核心领域所对应的软件组件一般建议自研。同时我建议你把公司最好的人手放在核心领域的开发项目组中，这就是好钢用在刀刃上。

另外，由于核心竞争力需要不断升级，你还要做好长期维护的准备。我们开头说过，领域驱动设计的一个侧重点是投资回报比。核心领域是重要的赢利点，所以它需要有足够的时间和资源投资来保证未来的竞争优势。

审时度势

金融行业的玩家都是聪明人，所以一家公司的核心竞争力是瞒不住的。稍微过段时间就会有人学会你的赚钱之道，这时候你们之间的竞争就变成了公平竞争，你就再也没有核心竞争力了。

还是举之前金融市场的例子。在上世纪 70 年代，计算机处理订单是一家金融公司的核心竞争力。但是很快人们发现只要用了电脑就可以有类似的服务质量。金融公司最不缺的就是钱，所以它们很快也都对业务做了更新换代，用电脑替代人工操作。

这时候金融行业的核心竞争力又变成了怎么用计算机的速度来赚钱。

过了 40 年，到了 08 年金融危机的时候，有的金融公司靠着超前的风险计算能力躲过了危机。很快怎么复制这种计算方法不再神秘，这时核心竞争力就变成了谁能最快地上线风控模型，以及谁的计算成本最低。

所以在金融行业，我们想判断核心领域，就需要合理判断此时此刻公司所处的行业位置，在可预见的未来对手会如何应对，以及自己应该如何针对性地升级业务和系统。

宏观视角

每家公司的核心竞争力都不相同，所以一家公司的核心领域可能对其它公司来说并不是核心。比如，尽管同属于金融行业，券商的会计系统不是最核心的领域，但是对于一家提供会计服务的公司来说，会计系统是它最核心的领域。

所以核心组件不能根据其内容来一刀切，而是要根据它属于哪一个更大的环境，再来做灵活判断。

那怎么宏观地判断一个核心领域呢？这就要看它能不能给你带来**超额利润**。超额利润是一个经济学的名词，表示超出一般行业水平的利润。比如对券商来说，能不能发短信通知给

用户其实对盈利水平的影响不大。但是如果能提高用户下单的速度，那就完全不一样了。

通用领域

顾名思义，通用领域是可以在不同行业通用的领域。比如我们前面提到的短信平台，不只是金融公司可以使用，物流公司也可以使用。类似的还有安全、日志、存储等等。

既然是通用领域，那么市场上一定会有多个类似的产品，而且也会存在第三方服务提供商。领域驱动设计强调投资回报比，**所以当市场上存在多个类似的产品的话，我们要尽量采购相关服务或者产品，而不是自己研发。**

道理很简单。同一件事情如果做的次数越多，质量就会越好，成本也会更低。所以相比自己做，通用领域选择第三方服务提供商的性价比更高。

当然了，有些情况下我们会发现通用领域也不是完全通用的，在一些细小的地方不能完全满足要求。如果你找不到完全符合自己需求的产品，就争取只做二次开发。如果二次开发也不行，一定需要投入研发力量，那么尽量投入一些非核心开发力量，比如外包团队或者非资深开发人员。

同时你也不要对软件质量做过多的要求。这时候要**本着能用就行的心态**，研发系统的时候也按**随时能替换**的方式去设计。

支持型领域

支撑型领域是那些用来辅助核心领域正常运行的领域。支持型领域并非核心竞争力，但是缺了之后也无法正常开展业务。比如说会计系统、市场数据系统等等，一般属于支持型领域。

支持型领域和通用领域很容易搞混。支持型领域一般并不会跨行业通用，顶多只在金融行业内通用。

比如对一家做聊天论坛的公司来说，一个股票数据分析系统并没有多大帮助。因为支持型领域通用程度不大，它们的共性就会更小，因此更难找到可以替代的解决方案。

如果你确定了某个系统是支持型领域，那么你可以按照这 3 个步骤来考虑：

1. 和通用领域类似，首先考虑购买第三方软件。

2.**如果市场上没有合适的产品，考虑能否人工处理。**比如一些简单的会计科目处理或者市场数据处理，用办公软件就能达到很好的效果。金融公司不是互联网公司，只要能赚钱就行，不需要为了面子而投入稀缺的软件开发力量。

3. 如果我们一定需要自研，要和对待通用领域的态度一样，能用就行，保持**和核心领域低耦合**，随时准备替换。

领域分析举例

我们讲完了怎么做领域分析之后，你一定想知道怎么分析一个实际的例子。

假设你有一天成为了有钱人，需要有人帮你打点一下财产。这时候金融公司会给你提供**资管服务**，替你理财。理财产品有很多种，其中资产数量最大、收益最高、风险也最高的一类是衍生品。在这里我们看看衍生品管理系统应该怎么分析。

理财产品的管理分两步。先要购买合适的金融产品，也叫交易前，或者**投前**。买好之后就需要管理，在合适的时候买进卖出。这一步也叫交易后，或者**投后**。我们先来看看投前的过程涉及到的系统应该怎么分析。

首先，你需要在系统中记录有兴趣购买的衍生品。因为衍生品是一种有完整生命周期的金融合同，所以系统需要有一个金融合同的**生命周期管理系统**。

其次，你在挑选金融产品的时候需要知道这个产品的价格，如果价格低就买。这就是定价和报价环节。这个环节开始的时候，你需要对合同价格有自己的判断，这就要用到**定价系统**。

接着就到了合同签订和打印的步骤。金融合同的签订其实是一个交易过程，就需要有**交易系统**。

衍生品交易涉及到的金额特别巨大，因此很多人都不太信任电子合同。合同签订之后还需要打印存档，这就需要有**打印系统**。

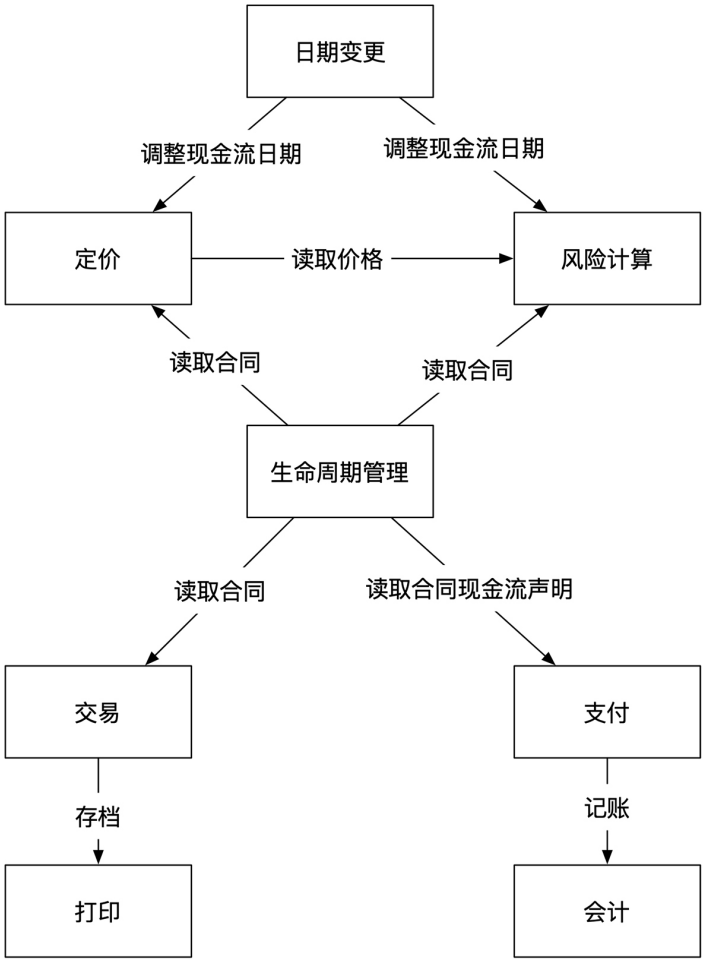
投前的步骤在合同确认之后就结束了。接下来就需要你自己管理合同了。这时一般要注意这几个事项。

第一，金融合同签订以后，买卖双方需要履行合同义务，也就是按照合同声明的金额进行资金往来，因此需要有**支付系统**和**会计系统**。

第二，资金转账通常不能在节假日进行，所以需要**日期系统**来通知系统自动调整日期。

第三，你需要随时知道自己的资产有多少，风险有多大，这里需要有**风险计算系统**。

那么，一个简单的衍生品管理系统就成型了，它需要下面这些不同的领域系统：

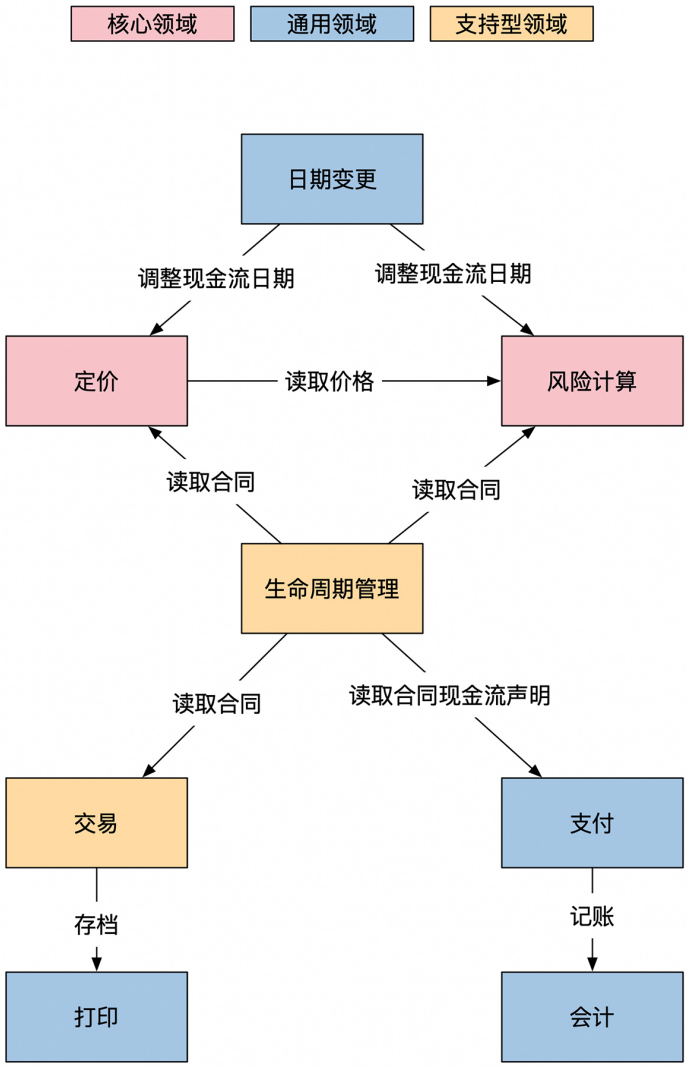


到这里，我们完成了衍生品管理系统的初步设计。接下来我们来做个练习，看看如何将每个部分划分到正确的领域。

我们先来看看哪几个是核心领域。你做资产管理是为了获得合理的收益，所以核心业务应该和金融产品的收益相关。这里我们发现有两个系统组件比较相关，分别是定价和风险计算系统，所以这两个属于核心领域。

至于打印、支付、会计、日期变更等都是常见功能。除了金融行业，很多其他行业也有，所以这些都是通用功能，属于通用领域。

但是相对而言，交易、报价和生命周期管理没那么通用，但是也不属于核心竞争力，因此这些都属于支持型领域。下面这幅图展示了我们划分下来的结果：



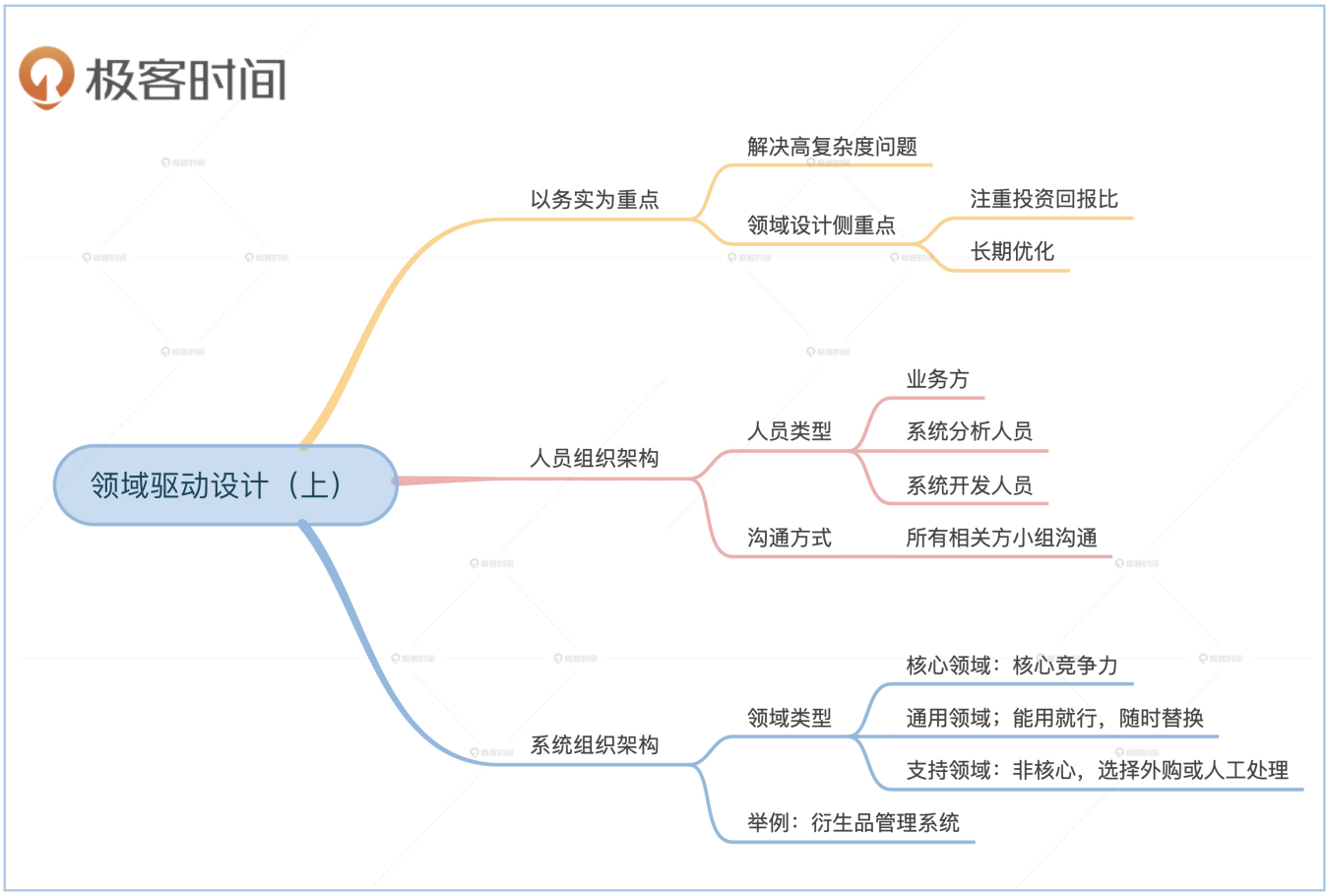
小结

这节课我们介绍如何通过领域驱动设计来对金融系统做顶层设计。设计的两个核心原则是**追求投资回报比，以及长期优化。**

领域驱动设计首先解决了人的问题。软件系统研发是多个部门通力合作的结果。合作的过程需要有效的沟通，因此建议直接沟通，而不是采用传统的层级式沟通方式。沟通时不同部门的人需要尽量了解对方的术语。当大家有了共同的术语后还要沉淀下来，形成领域语言。

理顺了人之后，我们就可以解决事情了。领域驱动设计将复杂的系统组件分为三大领域。核心领域与金融公司的核心竞争力有直接关系。通用领域可以跨不同行业。支持型领域负责支持核心领域，跨行业的可能性较低。

最后我们结合前面所学的内容，通过分析衍生品管理这个业务，练习了如何分析必须的系统组件，以及如何划分这些组件的领域。



思考题

两个不同领域之间传输的数据类型也是有讲究的。一般来说有两种选项：

1. 传输的是领域模型。
2. 传输的是简单数据类型，比如 POJO（Plain Old Java Object），Json 等。

如果你是一家大金融公司的 CTO，需要进行公司级别的领域驱动设计。这时除了需要考虑软件本身的设计外，还需要结合公司的人员变更现状（10% 的平均年员工流动率）和公司未来 10 年的软件发展计划（金融业务变更非常频繁）。

现在你需要拿出一份指导意见，那么你会选择传输哪种数据类型呢？

欢迎留言和我分享你的想法。如果这节课让你有所收获，也欢迎分享给你的朋友，一起交流进步。

提建议

上一篇 03 | 产品大观：不同金融业务都有哪些技术实现要点？

下一篇 05 | 领域驱动设计（下）：如何设计统一的金融业务模型？

精选留言 (7)

写留言

Jerry Wu
2020-12-31

我说下自己的一些经历吧~

在人员组织架构上，领域驱动设计要求太高，不适合大部分公司。金融行业可以用领域驱动设计，一个重要的原因是，这些企业能招到足够的人才。

...

展开

6

楼下小黑哥
2020-12-29

领域模型将会包含自身领域的业务逻辑，，不应该暴露给其他领域，否则就会形成不同领

域之间的系统耦合。
所以应该选择 2，对外只传输简单数据类型，业务逻辑还只是在自身领域内实现

1

tt
2020-12-28

我也觉得是选项2，因为不同的领域属于不同的业务。如果直接传输领域模型:

一个会带上很多目的领域不需要的数据;
第二个划分不同领域的目的是为了解耦，如果直接传输领域模型相当于违背了这个初衷。

展开

> 编辑回复: 给你的学习热情和思考点赞～这里小小剧透一下，第一模块的参考答案下一期会公布的。

1

webmin
2021-01-14

传输数据可以采用模型语言（IDL）定义好模型后，根据模型通过生成器生成不同的编程语言和RPC协议的Server和Client的stub。

展开

Geek_b17612
2021-01-12

选项2，不同领域之间通过简单对象进行交互。既然区分了领域，那就意味着两个不同领域的内容不应该强依赖。他们内部的处理逻辑也不应该暴露出来，那么也就没有必要暴露出领域模型。

Geek_74107b
2020-12-28

选json或者protobuf这种通用的 业务无关的 语言无关的数据类型

江湖小虾
2020-12-28

选项2，因为领域模型会根据业务逻辑经常变化，容易造成不同系统之间的耦合，使用不带

有业务逻辑的简单数据类型，不带有业务逻辑，适合接口的版本的管理

展开