图灵电商网站需求分析和架构介绍

课程暂定章节安排

目前安排有 19 个章节：

图灵电商网站需求分析和架构介绍（Mark）

管理后台快速实现跨库数据管理（楼兰）

前端架构、电商支持服务（楼兰）

电商项目微服务架构拆分实战（Fox）

电商项目 CI\CD 自动化部署功能实现与优化（楼兰）

电商项目分布式 ID 微服务实战（Mark）

电商项目微服务网关整合授权中心实现单点登录（Fox）

订单系统的设计与海量数据处理实战（Mark）

电商订单支付流程设计与实现（楼兰）

下单链路分布式事务 Seata&MQ 可靠消息实战（Fox）

电商项目高并发缓存实践（Mark）

大型网站的数据同步、备份与恢复（Mark）

高并发秒杀系统的设计与实现（Mark）

秒杀链路兜底方案之限流&熔断降级实战（Fox）

大型网站高并发的读、写实践（Mark）

海量数据存储、查询和最佳实践（Mark）

电商项目商品搜索 ElasticSearch 实战（Fox）

项目总结和架构师技能分析（Mark）

服务开放平台 DDD 设计与落地（楼兰）

注意：这个不是课时，完全可能存在着一个章节跨数个课时的情况。

电商概述

电商业务与我们的生活息息相关，大家应该对电商多少也有一些了解。我们先做个角色扮演，假设你现在是一个创业公司的 CTO，代入他的视角，设计一个最小化的电商系统，并以此理清电商系统的架构，让大家对电商系统的业务逻辑、系统架构、核心业务流程有一个基本的认知。这之后的学习就不用再解释什么是电商的业务和系统了，直接讲解具体的技术问题即可。

有个老板有天和你说：“我有一个改变世界的想法，万事俱备，前台小妹都配了。只差一个程序员“，于是邀请你加盟。新公司很快就成立了，你也成了新公司的 CTO。关于改变世界，目前唯一能确定的是，首先要做一个电商系统。具体要做成什么样，目前还不清楚。你需要与老板讨论业务需求。

你:“咱们要做的业务模式是 C2C、B2C 还是 B2B 呢?”老板:“什么 B?什么 C?我不懂你说的那些技术名词。”

你:“这么说吧,你要做一个某宝网,还是某东网,还是某 848 网呢?”老板:“不都是一样的吗?它们之间有什么区别?你赶紧做一个出来我看看不就知道了?!”于是你只好先和老板做科普，电商其实分为很多种类型，包括有：

按运营方向:

传统电商 淘宝天猫、京东、唯品会、聚美优品、当当...

社交电商 拼多多、京东、国美、苏宁

网红电商 抖音、快手

内容电商 头条三农领域

上面这么多电商系统肯定有其共同点，比如以下电商术语。

常见电商术语

PV： Page view，即网站被浏览的总次数

UV： Unique Vister 的缩写，独立访客

CR： ConversionRate 的缩写，是指访问某一网站访客中，转化的访客占全访客的比例（订单转化率=有效订单数/访客数）

SPU： Standard Product Unit （标准化产品单元），SPU 是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。

SKU： Stock keeping unit(库存量单位) SKU 即库存进出计量的单位（买家购买、商家进货、供应商备货、工厂生产都是依据 SKU 进行的），在服装、鞋类商品中使用最多最普遍。

例如纺织品中一个 SKU 通常表示：规格、颜色、款式。SKU 是物理上不可分割的最小存货单元。

但是电商系统类型的不同，肯定这几种还是有所不同的，比如商品的 SPU，类似淘宝这样的平台联营类型的 SPU 数量可以达到百万甚至千万级别，而小米商城之类的自营类型的 SPU 往往在几万左右。

我们的本期的项目则是类似于小米商城，属于自营类型的电商系统。

事实上，即使是一个最小化的电商系统,也依然是非常复杂的。故事发展到这里，作为程序员的你是不是有一种似曾相识的感觉?现实就是,需求永远不明确,永远在变化，唯一不变的只有变化。优秀的程序员适应变化,并且拥抱变化。在需求还不太明确的情况下，比较可行的方案就是,首先搭建不太会发生变化的核心系统，然后尽量简单地实现一个最小化的系统，后续再逐步迭代和完善。

设计电商系统的核心流程

接下来,我们一起设计一个电商的核心系统。

遵照软件工程的一般规律,我们先从需求阶段开始。那么，需求分析应该如何做呢?理想情况下,系统分析师或产品经理应该负责完成需求分析的任务。但是,现实中绝大多数情况下,你得到的所谓的“需求”,很有可能就是一两句话。需求分析的工作最终往往是由开发者完成的。

很多项目交付以后,仍需要不断地进行修改和变更,用户不满意,开发者也很痛苦,造成这个问题的根本原因其实就是缺失了需求分析的步骤。所以,为了后续工作能够顺利开展,每位开发者都应该掌握一些用于需求分析的方法。那么,开发者进行需求分析时应该做些什么呢？我们本次课不介绍那些做需求分析的方法和理论，更多有关需求分析的知识请参考附录《需求分析指引》。

其实任何产品都是给人用的，所以需求分析最重要、最关键的一个点：不要一上来就设计功能,而是先明确下面这两个问题的答案。

1）这个系统(或者功能)是给哪些人用的?

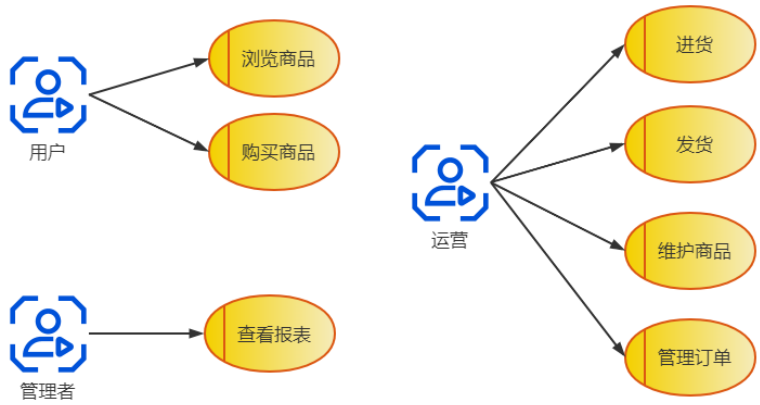
2）这些人使用这个系统是为了解决什么问题?

这两个问题的答案，我们称之为业务需求。那么，对于我们将要设计的电商系统，其业务需求又是什么呢?如果大家平时用过电商，熟悉电商的业务,那么回答这两个问题应该很容易。

第一个问题,电商系统是给哪些人用的?首先是买东西的人，即“用户”;其次是卖东西的人，即“运营”；还有一个非常重要的角色就是出钱的人，即“管理者”(请记住，在设计任何一个系统的时候，出钱的人的意见都是重要的，甚至可以说是最重要的)。综上所述,电商系统是面向用户、运营和管理者开发的。

第二个问题,用户、运营和管理者使用电商系统分别想要解决什么问题?这个也很容易回答，用户为了买东西，运营为了卖东西,管理者需要通过系统了解自己所得的收益。

这两个问题的答案,或者说业务需求可以用下图表述。



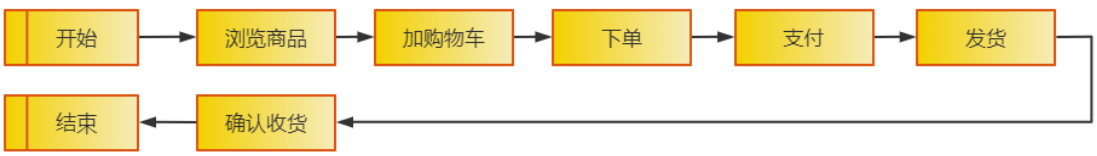
上面的图被称为用例图(Use Case)，是我们进行需求分析的时候所要画的第一张图。用例图可用于回答业务需求中的两个关键问题,即这个系统给谁用?他们用这个系统是为了解决什么问题?

但是实际来说业务需求与我们要设计的系统关系不大。为什么这么说呢?因为我们将上的用例，放在传统的商业企业(比如,一个小杂货铺、一个线下实体商场或商店,或者一个做电视购物的公司)中也是适用的,所以,做业务需求的主要目的是理清楚业务场景是怎样的。

下面就来分析电商系统的业务流程。很显然,电商系统最主要的业务流程,一定是购物流程。购物流程很简单，所有电商的购物流程几乎都是如此，具体流程如下图所示。

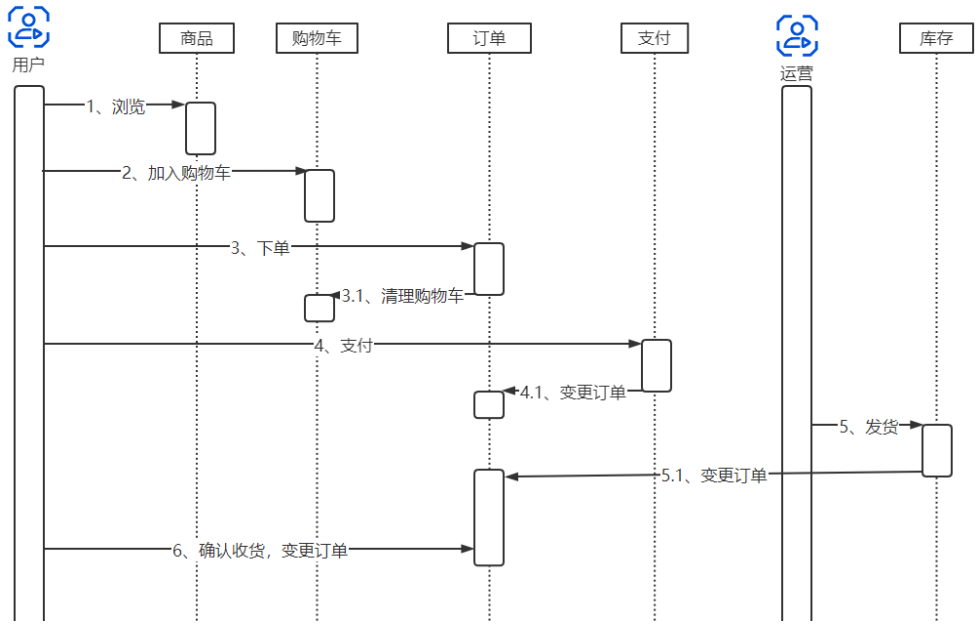
下面就来分析一下这个流程。流程从用户选购商品开始,用户首先在 App 中浏览商品,找到心仪的商品之后,把商品添加到购物车,选完商品之后,打开购物车,提交订单。

下单结算之后,用户就可以支付了。支付成功后,运营人员会为已经支付的订单发货,为用户邮寄相应的商品。最后,用户收到商品并确认收货。至此,一个完整的购物流程就结束了。



根据流程来划分模块

接下来,我们再进一步细化电商购物的业务流程,看一下电商系统是如何实现该流程的。下图所示的是细化之后的电商系统购物流程时序图(Sequence Diagram)。



1）用户浏览商品,这个步骤需要通过一个商品模块来展示商品详情页,用户可以从中获取所浏览商品的详细介绍和价格等信息。

2）然后,用户把选好的商品加入购物车,这个步骤需要使用一个购物车模块来维护用户购物车中的商品。

3）接下来是用户下单，这个步骤需要基于一个订单模块来创建新订单。订单创建好了之后,系统需要把订单中的商品从购物车中删减掉。

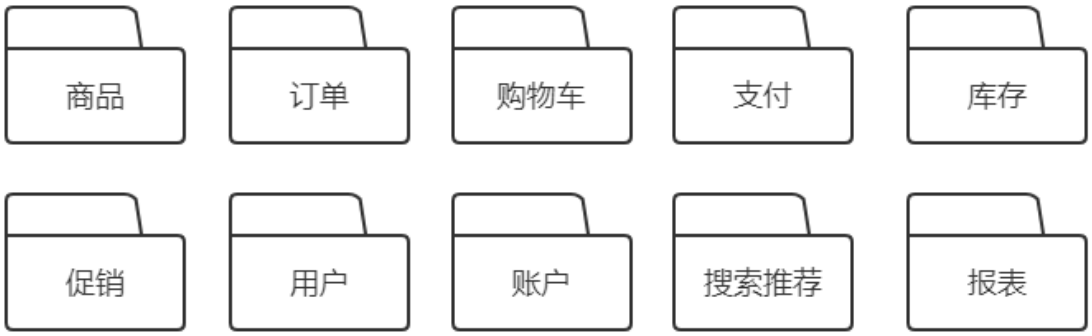
4）订单创建完成后,系统需要引导用户付款,即发起支付流程,可通过一个支付模块来实现支付功能,用户成功完成支付之后,系统需要把订单的状态变更为“已支付”。

5）成功支付之后,运营人员就可以发货了,发货之后,系统需要扣减对应商品的库存数量,这个步骤需要基于一个库存模块来实现库存数量的变更,同时系统还需要把订单状态变更为“已发货”。

6）最后,用户收到商品,在系统中确认收货,系统需要把订单状态变更为“已收货”,流程结束。

这个流程涉及 5 大功能模块,即商品、购物车、订单、支付和库存。这 5 大模块就是一个电商系统中的核心功能模块。

当然,仅有这 5 个模块是不够的,因为我们只分析了“购物”这个最主要的流程，并没有完全涵盖业务需求中的全部用例,比如,运营人员进货、管理者查看报表等还没有覆盖到。相比购物流程,剩下的几个用例和流程都相对简单一些,大家可以采用同样的方法来分析其他的功能模块。一般情况下的电商系统的功能模块划分如下图所示。



整个系统按照功能,可以划分为 10 个模块,除了购物流程中涉及的商品、订单、购物车、支付和库存这 5 个模块之外,还有了促销、用户、账户、搜索推荐和报表这 5 个模块，这些都是构建一个可实际运营电商系统必不可少的功能模块。每个模块需要实现的功能说明如下。

- 商品:维护和展示商品的相关信息。
- 订单:维护订单信息和订单状态,计算订单金额。
- 购物车:维护用户购物车中商品的信息。
- 支付:负责与系统内外部的支付渠道对接,实现支付功能。
- 库存:维护商品的库存信息。
- 促销:制定促销规则,计算促销优惠信息。
- 用户:维护系统的用户信息,注意,用户模块是一个业务模块,一般不负责用户的登录和认证,这是两个完全不同的功能。
- 账户:账户模块负责维护用户的账户信息，例如用户的积分、等级、余额。
- 搜索推荐:提供商品搜索功能,并负责各种商品列表页和促销页的组织和展示,简单地说就是，搜索推荐决定用户优先看到哪些商品。
- 报表:实现数据统计和分析功能，生成报表,为管理者进行经营分析和决策提供数据信息。

在实际的电商网站中，促销模块是电商系统中最复杂的一个模块。各种优惠券、满减、返现等促销规则,每一条都非常复杂，再加上这些规则往往还要叠加计算,有时甚至会复杂到连制定促销规则的人都算不清楚。在实际的运营中，所有电商公司几乎都发送过因为促销规则制定失误,导致商品实际售价远低于成本价,使公司受到一定程度的损失。尽管如此,五花八门的促销活动依然是提升销量最有效的手段,因此需要充分利用。

作为电商系统的设计者，我们需要把促销规则的变化和复杂性控制在促销模块内部，不能因为一个促销模块而导致整个电商系统都变得非常复杂,否则设计和实现将会很难。

所以一般来说,可以把促销模块与其他模块的接口设计得相对简单和固定,这样系统的其他模块就不会因为新的促销规则改变而随之进行改变。在创建订单时,订单模块需要把商品和价格信息传给促销模块,促销模块返回一个可以使用的促销列表，用户选择对应的促销和优惠,订单模块把商品、价格、促销优惠等信息,再次传给促销模块,促销模块再返回促销之后的价格。在最终生成的订单中,系统只需要记录订单使用了哪几种促销规则,以及最终的促销价格就可以了。这样,无论促销模块如何变化,订单和其他模块的业务逻辑都不需要随之改变。

至此,我们就完成了一个电商系统的概要设计,大家对电商系统应该也有了一个初步的了解。

总结电商系统设计核心要点

首先,电商系统面向的角色是:用户、运营人员和管理者。这三个角色对电商系统的需求是:用户通过系统来购物,运营人员负责商品的销售,管理者关注系统中的经营数据。电商系统最核心的流程是用户购物的流程,购物流程从用户浏览选购商品开始,加购、下单、支付、运营人员发货、用户确认收货,至此电商系统的购物流程结束。

细化这个流程之后,我们可以分析出支撑这个流程的核心功能模块:商品、订单、购物车、支付和库存。除此之外,一个完整的电商系统还包括促销、用户、账户、搜索推荐和报表这些必备的功能模块。

作为一名开发者,在做需求分析的时候,需要把握的一个要点是:不要一上来就设计功能,而是要先理清业务需求。

如果系统业务是复杂而多变的.那么请尽量识别出这部分复杂业务的边界,将复杂业务控制在一个模块内部,从而避免将这种复杂度扩散到整个系统中去。

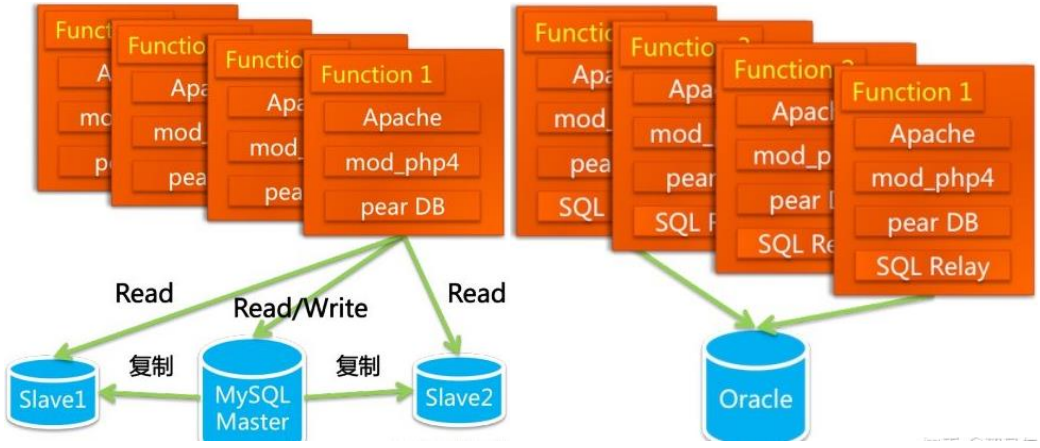
完成了概要设计之后,接下来就进入技术选型阶段了。作为公司的 CTO，请思考：这个电商系统的技术选型应该是什么样的？使用哪种编程语言和技术栈?

要注意，技术选型本身并没有好坏之分,更多的是选择“合适”的技术。什么是“合适”的技术？分布式？微服务？云原生？是不是最新的技术就是最适合我们的技术？不能这么看，从淘宝网的发展来看，就知道这样的选择并不正确。

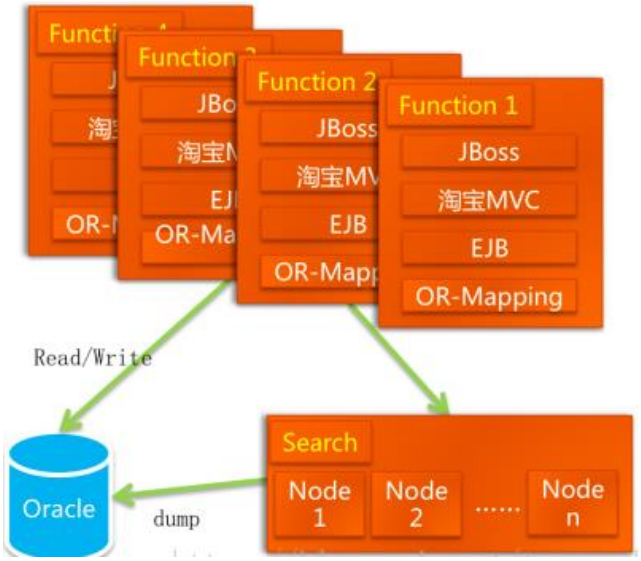
从《淘宝技术这十年》这本书我们就可以知道，阿里的技术发展也是一步步发展到今天这个地步的：

第一个阶段：LAMP（Linux+Apache+Mysql+PHP）。LAMP 是淘宝 03 年从美国买来（2000 美金）的，特点是无需编译，发布快速，所用的技术都是开源的、免费的。买回来之后做了一些本地化的修改。并且把一个数据库进行所有读写操作拆分成一个主库，两个从库，并且读写分离。

第二阶段：Oracle 阶段。随着业务量变大，mysql 已经撑不住了（不同于现在的 mysql），最简单的方案就是换成 oracle。



第三阶段：淘宝请的 sun 公司的人进行改造，思路是把业务分块，一个一个模块渐进式替换。整个结构用到的是 java MVC 模式。



第四阶段：IOE 阶段。随着业务的增加，IOE（IBM 小型机、Oracle 数据库、EMC 存储）阶段主要思路尽可能利用钱来解决问题。

第五阶段：稳中求变。DBRoute 框架统一处理了数据的排序、合并、分页等问题。Spring 替换掉 EJB。加入缓存、CDN（内容分发网络）。

第六阶段：创造技术阶段。随着淘宝业务量的猛增，当时现有技术已经不能满足需求，只能创造技术。TFS 文件系统的成功研发解决了淘宝的图片存储问题。缓存 Tair 的开发使淘宝的图片功能得到充分的发挥。AJAX、prototype 的技术大大简化前端的交互。ESI 技术解决了页面端静态页面的缓存。

第七阶段：分布式阶段。随着业务量到达一个瓶颈，原来架构已经无法满足需求，所以对系统模块进行逐步拆分和服务化改造。其实真的没有说的这么简单，整个架构变化都是问题出现、问题解决、不断探索、不断改造、不断调整的过程。

拆分之后，好处自然不必多说，但系统的交互关系变得异常复杂，越往底层的系统调用量和访问量越大，这就需要底层系统具备超大规模的容量和高可用性。又发展了 HSF（实时调用中间件）和 Notify（异步消息通知中间件），单点登录系统、TDDL 分布式数据层等等我们现在熟知的各种阿里开源的中间件。

第八阶段：大数据+云计算+服务化。

对于我们来说，不管是工作中还是面试中，而对于技术栈的选择，需要考虑如下方面，一方面是团队的人员配置，应该尽量选择大家熟悉的技术，另一方面是考察所选择技术的生态是否足够完善，其次就是业务可能需要应对的用户量、数据量和并发数等等关键指标。

面试题：你的项目为什么要从单体变化到微服务

对于我们大多数同学的工作经历来说，一般开始接触比较多的是单体 war 包应用，大致对应着上面的第 2、3 阶段，而在后续的业务变化中在进行架构调整，这个时候我们一般会选择使用微服务架构。当然随着技术的发展和开源运动的兴起，可以直接选用成熟的微服务架构框架，比如 SpringCloud 或者 SpringCloudAlibaba。

在进行架构升级前，我们要知道，为什么要从单体架构升级到微服务架构，单体架构和微服务架构的优缺点分别是什么？

单体服务的优点

应用的开发很简单，IDE 和其他开发工具只需要构建这一个单独的应用程序。

易于对应用程序进行大规模的更改:可以更改代码和数据库模式，然后构建和部署测试相对简。

直观:开发者只需要写几个端到端的测试,启动应用程序，调用 RESTAPI，然后使用 Selenium 这样的工具测试用户界面。

部署简单明了:开发者唯一需要做的,就是把 WAR 文件复制到安装了 Tomcat 的服务器上。

横向扩展不费吹灰之力:运行多个实例，由一个负载均衡器进行调度。

单体服务的缺点

单体架构存在着巨大的局限性。随着业务的复杂和用户的增多，小巧的、简单的、由一个小团队开发维护的应用程序就会演变成一个由大团队开发的巨无霸单体应用程序。此时应用就变成了单体地狱。开发变得缓慢和痛苦。

过度的复杂性

大型单体应用程序的首要问题就是它的过度复杂性，系统本身过于庞大和复杂导致任何一个开发者都很难理解它的全部。因此，修复软件中的问题和正确地实现新功能就变得困难且耗时。各种交付截止时间都可能被错过。

这种极度的复杂性正在形成一个恶性循环:由于代码库太难于理解，因此开发人员在更改时更容易出错，每一次更改都会让代码库变得更复杂、更难懂。就演变成为了我们常说的“代码屎山”。

开发速度缓慢

巨大的项目从编辑到构建、运行再到测试这个周期花费的时间越来越长,这严重地影响了团队的工作效率。

部署周期长,易出问题

巨大的项目从代码完成到运行在生产环境是一个漫长且费力的过程。一个问题是，众多开发人员都向同一个代码库提交代码更改，这常常使得这个代码库的构建结果处于无法交付的状态。采用功能分支来解决这个问题时，带来的是漫长且痛苦的合并过程。紧接着，一旦团队完成一个冲刺任务，随后迎接他们的将是一个漫长的测试和代码稳定周期。

把更改推向生产环境的另一个挑战是运行测试需要很长时间。因为代码库如此复杂,以至于一个更改可能引起的影响是未知的，为了避免牵一发而动全身的后果，即使是一个微小的更改，开发人员也必须在持续集成服务器上运行所有的测试套件。系统的某些部分甚至还需要手工测试。如果测试失败，诊断和修复也需要更多的时间。因此，完成这样的测试往往需要数天甚至更长时间。

难以扩展和可靠性不佳

在很多的时候，应用的不同功能和模块对资源的需求是相互冲突的。例如，有些模块需要将数据保存在一个大型的内存数据库中，理想情况下运行这个应用的服务器应该有较大容量的内存；另外，有些模块存在大量的计算又需要比较快的 CPU，这需要项目部署在具有多个高性能 CPU 和大内存的服务器之上。因为这些模块都是在一个应用程序内，因此在选用服务器时必须满足所有模块的需要。

应用程序缺乏故障隔离，因为所有模块都在同一个进程中运行。每隔一段时间，在一个模块中的代码错误，例如内存泄漏，将会导致应用程序的所有实例都崩溃。

单体应用的适用场景

公司规模较小，开发团队人数较少、产品上线周期短、产品在快速迭代期，核心功能尚未稳定时；或者用户规模和用户群体较少时。

微服务架构

今天,针对大型复杂应用的开发,越来越多的共识趋向于考虑使用微服务架构。但微服务到底是什么？针对微服务架构有多种定义。有些仅仅是在字面意义上做了定义:服务应该是微小的不超过 100 行代码,等等。另外有些定义要求服务的开发周期必须被限制在两周之内。

曾在 Netflix 工作的著名架构师 Adrian Cockcroft 把微服务架构定义为面向服务的架构，它们由松耦合和具有边界上下文的元素组成。由这个定义可以看到，微服务其实和 DDD 是天生一对。

微服务是模块化的

模块化是开发大型、复杂应用程序的基础。现代互联网应用程序为了让不同的人开发和理解，大型应用需要拆分为模块。在单体应用中，模块通常由一组编程语言所提供的结构（例如 Java 的包),或者 Java JAR 文件这样的构建制品来定义。但是，即使这样，随着时间的推移和反复的开发迭代,单体应用依然会变成我们前面所说的单体地狱。

微服务架构使用服务作为模块化的单元。服务的 API 为它自身构筑了一个不可逾越的边界,你无法越过 API 去访问服务内部的类，这与采用 Java 包的单体应用完全不同。因此模块化的服务更容易随着时间推移而不断演化。微服务架构也带来其他的好处,例如服务可以独立进行部署和扩展。

每个服务都拥有自己的数据库

微服务架构的一个关键特性是每一个服务之间都是松耦合的，它们仅通过 API 进行通信。实现这种松耦合的方式之一，是每个服务都拥有自己的私有数据库。对于我们的项目来说，订单服务拥有一个包括 oms_order 等表的数据库，用户服务拥有一个包含 ums_member 等表的数据库。在开发阶段就可以修改自己服务的数据库模式，而不必同其他服务的开发者协调。在运行时，服务实现了相互之间的独立。服务不会因为其他的服务锁住了数据库而进入堵塞的状态。

微服务架构的好处

使大型的复杂应用程序可以持续交付和持续部署。

每个服务都相对较小并容易维护。

服务可以独立部署。

服务可以独立扩展。

微服务架构可以实现团队的自主和松散耦合。

更容易实验和采纳新的技术。

更好的容错性，比如更好的故障隔离。

微服务架构的弊端

服务的拆分和定义是一项挑战

采用微服务架构首当其冲的问题，就是根本没有一个具体的、良好定义的算法可以完成服务的拆分工作。与软件开发一样，服务的拆分和定义更像是一门艺术。更糟糕的是，如果对系统的服务拆分出现了偏差，你很有可能会构建出一个分布式的单体应用:一个包含了一大堆互相之间紧耦合的服务，却又必须部署在一起的所谓分布式系统。这将会把单体架构和微服务架构两者的弊端集于一身。

分布式系统带来的各种复杂性,使开发、测试和部署变得更困难。

使用微服务架构的另一个问题是开发人员必须处理创建分布式系统的额外复杂性。服务必须使用进程间通信机制。这比简单的方法调用更复杂。此外，必须设计服务来处理局部故障,并处理远程服务不可用或出现高延迟的各种情况。

实现跨多个服务的用例需要使用不熟悉的技术。每个服务都有自己的数据库,这使得实现跨服务的事务和查询成为一项挑战。如第 4 章所述，基于微服务的应用程序必须使用所谓的 Saga 来维护服务之间的数据一致性。第 7 章将解释基于微服务的应用程序无法使用简单查询从多个服务中检索数据。相反，它必须使用 API 组合或 CQRS 视图实现查询。

IDE 等开发工具都是为单体应用设计的，它们并不具备开发分布式应用所需要的特定功能支持。编写包含多项服务在内的自动化测试也是很令人头疼的工作。这些都是跟微服务架构直接相关的问题。因此,团队中的开发人员必须具备先进的软件开发和交付技能才能成功使用微服务。

微服务架构还引入了显著的运维复杂性。必须在生产环境中管理更多活动组件:不同类型服务的多个实例。要成功部署微服务，需要高度自动化的基础设施,比如自动化部署等等。

当部署跨越多个服务的功能时需要谨慎地协调

使用微服务架构的另外一项挑战在于当部署跨越多个服务的功能时需要谨慎地协调更多开发团队。必须制定一个发布计划，把服务按照依赖关系进行排序，这就是我们常说的服务编排。这跟单体架构下批量部署多个组件的方式截然不同。

五期商城项目的架构

五期项目是在以前的项目上发展而来，在业务上基本保持不动，但做了更细致的微服务拆分，大体上秉承了前面所说的微服务设计原则，同时在具体的业务模块实现上，根据实际资源情况和教学、授课要求做了一定程度的取舍和合并，具体来说，相对一般的电商系统，我们没有单独实现账户、库存、支付服务，报表往往需要和 BI（Business Intelligence）报表框架集成，所以也未实现。因为说到底，我们学习本项目的目的是为了学习技术而不是为了学习电商业务。

项目结构

代码结构

前台

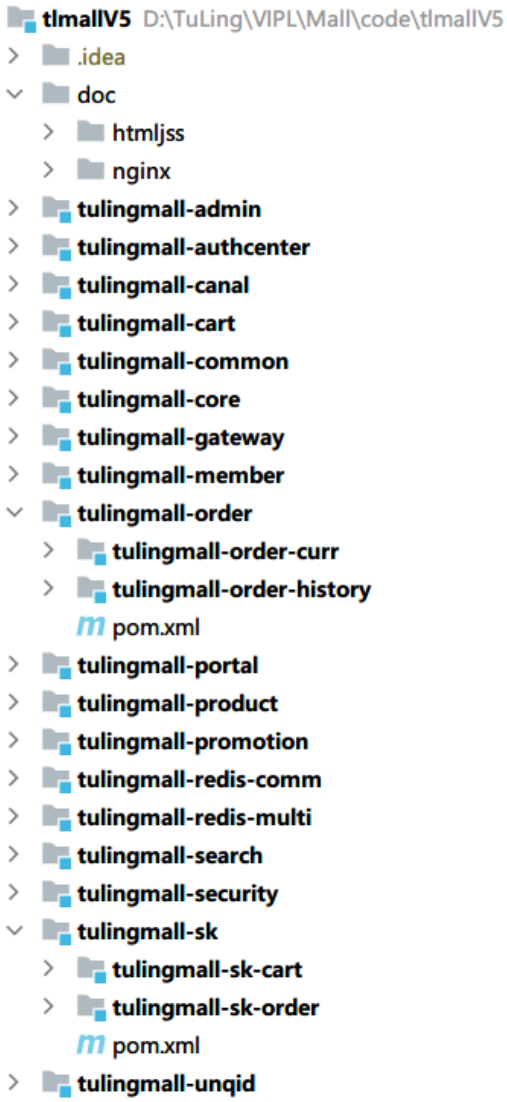
单独项目 tulingmall-front ， 基于 VUE 实现

后端

项目 tlmallV5 分为两大部分：

后台管理端 模块 tulingmall-admin

业务服务模块



tulingmall-authcenter 认证中心程序

tulingmall-canal 数据同步程序

tulingmall-cart 购物车程序

tulingmall-common 通用模块，被其他程序以 jar 包形式使用

tulingmall-core 遗留模块，主要包含 model 的声明，被其他程序以 jar 包形式使用

tulingmall-gateway 网关程序

tulingmall-member 用户管理程序

tulingmall-order-curr 订单程序

tulingmall-order-history 历史订单处理程序

tulingmall-portal 商城首页入口程序

tulingmall-product 商品管理程序

tulingmall-promotion 促销管理程序

tulingmall-redis-comm 缓存模块，被其他程序以 jar 包形式使用

tulingmall-redis-multi 多源缓存模块，被其他程序以 jar 包形式使用

tulingmall-search 商品搜索程序

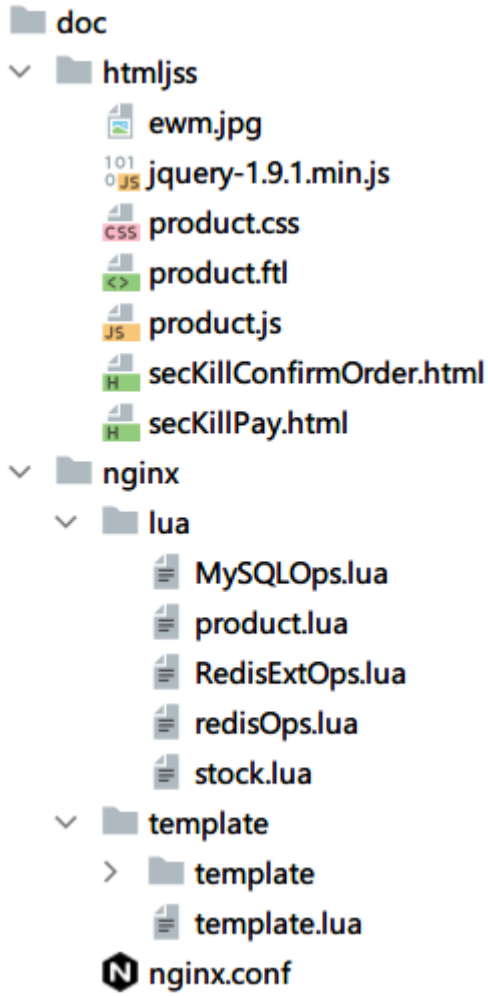
tulingmall-security 安全模块，被其他程序以 jar 包形式使用

tulingmall-sk-cart 秒杀确认单处理

tulingmall-sk-order 秒杀订单处理

tulingmall-unqid 分布式 ID 生成程序

秒杀 Nginx 相关



htmljss 秒杀静态网页、JS 文件、CSS 文件等

nginx 秒杀 nginx 配置、Lua 脚本、第三方 Lua 库

Database

整个项目中数据库被拆分为：

商品库 : tl_mall_goods

促销库 : tl_mall_promotion

用户库 : tl_mall_user

其他库 : tl_mall_normal

订单库 : tl_mall_order

购物车库: tl_mall_cart

课程关键 Table

用户库 ums_member ，用户/会员表

商品库 pms_product 商品表，主要包括四部分：商品的基本信息、商品的促销信息、商品的属性信息、商品的关联

商品库 pms_sku_stock 商品 SKU 库存表

订单库 oms_order 订单表

订单库 oms_order_item 订单详情表，订单中包含的商品信息，一个订单中会有多个订单商品信息，所以 oms_order 和它是一对多的关系

促销库 sms_home_brand 首页品牌推荐表

促销库 sms_home_new_product 首页显示的新鲜好物信息

促销库 sms_home_recommend_product 首页显示的人气推荐信息

促销库 sms_home_recommend_subject 首页显示的专题精选信息

促销库 sms_home_advertise 首页显示的轮播广告信息

促销库 sms_flash_promotion 秒杀/限时购活动的信息

促销库 sms_flash_promotion_product_relation 存储与秒杀/限时购相关的商品信息，也包含了秒杀/限时购的库存信息

项目环境

代码管理和项目发布

GitLab、Jenkins

中间件和基础设施

MySQL

Redis

ELK

RocketMQ

MongoDB

Canal

Promethus

Grafana

SpingCloudAlibaba 系列

项目中将学习的技术问题

分库分表、读写分离

分布式事务

全局唯一性 ID

分布式 Session

分布式链路跟踪

日志收集与展示

商品搜索

分布式锁

服务降级/限流/熔断/隔离

页面静态化

分布式任务调度

数据迁移方案

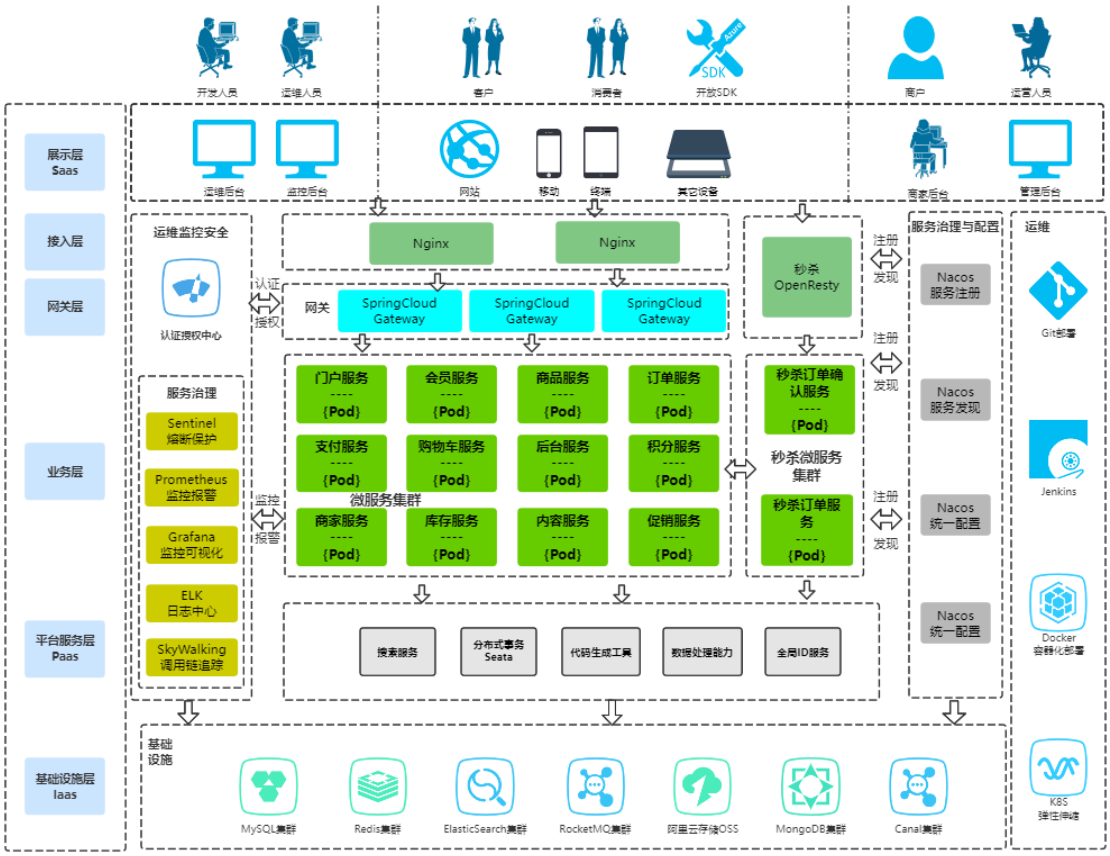
数据同步方案

多级缓存、缓存预热

高并发秒杀系统实现（秒杀系统的商详页静态化、秒杀系统的隔离、秒杀的削峰和限流等等）

项目架构图

五期项目的各个模块和中间件组合起来，发挥着各自的作用，组成了如下图所示的架构图：



项目学习必备知识清单

从上面的项目架构图可以看到，图灵商城项目牵涉到的知识点是非常多的，为了大家对后面课程学习的顺利进行，以下知识点需要大家在学习本项目前牢固掌握：

并发编程

SpringBoot + Mybatis + MySQL 下的 WEB 应用开发

图灵商城项目单体版

分布式基本概念

微服务的基本概念

事务一致性概念、多副本一致概念和 CAP 理论

SpingCloudAlibaba 系列组件（Nacos、Seata 等等）的作用和基本使用

Redis 各种数据结构和操作、Redis 主从和 Redis Cluster 基本概念

Lua 语言的基础语法

MongoDB 基本概念和操作、MongoDB 集群基本概念

消息中间件的基本概念和 RocketMQ 的基本操作、RocketMQ 集群基本概念

ELK 的基本概念和基本使用

JavaScript 基础语法

Linux 的基本概念和操作

Nginx 基本概念和操作

Git 的基本操作和 Maven 基本配置方法和常用命令

需求分析指引

一、需求分析定义

软件需求分析也称为系统需求分析或需求分析工程等，是开发人员经过深入细致的调研和分析，准确理解用户和项目的功能、性能、可靠性等具体要求，将用户非形式的需求表述转化为完整的需求定义，从而**确定系统必须做什么**的过程。

软件开发一般包括：可行性分析、需求分析、软件设计、软件开发、软件测试、软件实施、软件服务等步骤，需求分析是软件开发的第一步骤。

用户需求分析是指在系统设计之前和设计、开发过程中对用户需求所作的调查与分析，是系统设计、系统完善和系统维护的依据。

需求是需要与欲求的意思，需求是机体的一种客观需要，而欲求则是一种主观需要，包括人在胜利、环境、社会等方面的需要。

需求是一款产品的市场基础，成功的产品不但能满足用户的物质需求，也要满足**用户**的精神和心理需求。

二、软件需求分析目标

需求分析是软件计划阶段的重要活动，也是软件生存周期中的第一步，该阶段是分析系统在功能上需要“实现什么”，而不是考虑如何去“实现”。

对客户的信息化需求进行分析，将客户不规范的、随意的需求，转换成规范的、严谨的、结构化的需求，将客户不正确的需求转换成正确的需求、将客户不

切实际的需求转换成可以实现的需求，将客户不必要的需求砍掉，将客户漏掉的需求补上。

此外，软件的一些非功能性需求(如软件性能、可靠性、响应时间、可扩展性等)，软件设计的约束条件，运行时与其他软件的关系等也是软件需求分析的目标。

三、软件需求分析原则

需求分析通常来讲它们应符合以下一般原则：

1. 能够表达和理解问题的信息域

信息域反映的是用户业务系统中数据的流向和对数据进行加工的处理过程，根据信息域描述的信息流、信息内容和信息结构，可以较全面地（完整地）了解系统的功能。

2. 建立描述系统信息、功能和行为的模型

建立模型的过程是“由粗到精”的综合分析的过程。通过对模型的不断深化认识，来达到对实际问题的深刻认识。

3. 能够对所建模型按一定形式进行分解

分解是为了降低问题的复杂性，增加问题的可解性和可描述性。分解可以在同一个层次上进行（横向分解），也可以在多层次上进行（纵向分解）。

4. 分清系统的逻辑视图和物理视图

软件需求的逻辑视图描述的是系统要达到的功能和要处理的信息之间的关系，这与实现细节无关，而物理视图描述的是处理功能和信息结构的实际表现形式，这与实现细节是有关的。

需求分析只研究软件系统“做什么？”，而不考虑“怎样做？”。

四、软件需求分析内容

需求分析的内容是针对待开发软件提供完整、清晰、具体的要求，确定软件必须实现哪些任务。

具体分为功能性需求、非功能性需求与设计约束三个方面：

1. 功能性需求

功能性需求即软件必须完成哪些事，必须实现哪些功能，以及为了向其用户提供有用的功能所需执行的动作。

功能性需求是软件需求的主体，开发人员需要亲自与用户进行交流，核实用户需求，从软件帮助用户完成事务的角度上充分描述外部行为，形成软件需求规格说明书。

2. 非功能性需求

作为对功能性需求的补充，软件需求分析的内容中还应该包括一些非功能需求。

主要包括软件使用时对性能方面的要求、运行环境要求，软件设计必须遵循的相关标准、规范、用户界面设计的具体细节、未来可能的扩充方案等。

3. 设计约束

一般也称做设计限制条件，通常是对一些设计或实现方案的约束说明。

例如：要求待开发软件必须使用 Oracle 数据库系统完成数据管理功能，运行时必须基于 Linux 环境等。

五、软件需求分析过程

需求分析阶段的工作,可以分为四个方面：问题识别、分析与综合、制订规格说明、评审。

1. 问题识别

就是从系统角度来理解软件，确定对所开发系统的综合要求，并提出这些需求的实现条件，以及需求应该达到的标准。

这些需求包括：功能需求（做什么）、性能需求（要达到什么指标）、环境需求（如机型、操作系统等）、可靠性需求（不发生故障的概率）、安全保密需求、用户界面需求、资源使用需求（软件运行是所需的内存、CPU 等）、软件成本消耗与开发进度需求、预先估计以后系统可能达到的目标。

2. 分析与综合

逐步细化所有的软件功能，找出系统各元素间的联系，接口特性和设计上的限制，分析他们是否满足需求，剔除不合理部分，增加需要部分。

最后综合成系统的解决方案，给出要开发的系统的详细逻辑模型（做什么的模型）。

3. 制订规格说明书

即编制文档，描述需求的文档称为软件需求规格说明书。请注意，需求分析阶段的成果是需求规格说明书，向下一阶段提交。

4. 评审

对功能的正确性，完整性和清晰性，以及其它需求给予评价。评审通过才可进行下一阶段的工作，否则重新进行需求分析。

六、软件需求评估方法

需求评估分析方法通常有：模糊聚类分析、质量功能展开、KANO 模型分析、A/B 测试。其中以卡诺（KANO）模型最常用。

1. 模糊聚类分析法

是通过分析客观事物之间的不同特征和亲疏程度，建立模糊相似关系，从而对齐进行分类的方法。

在用户需求分析的运用中，判断用户需求之间的相似程度（亲疏关系），然后统计并建立相似性矩阵，继而寻找需求组合之间的相似程度，由此逐渐将用户需求逐一归类。

最终得到一个关系图谱，以更直观和自然的方式展示用户需求各个特性之间的差异性和相似性，模糊聚类分析法一般要求对需求进行数学建模分析。

2. 质量功能展开

是指把用户对产品的需求进行多层次的演绎分析，转化为产品的设计需求、工程部件特征、工艺要求、生产要求，用来指导产品设计并保证产品的质量，是一种以用户为导向的质量管理工具。

由于该方法所使用的主要图形就像房屋，所以它也被称为“质量屋”。

3.卡诺 KANO 模型

是 Noriaki Kano 博士提出的与产品性能有关的用户满意度模型，该模型能对用户需求进行很好的识别和分类，是对用户需求分类和优先排序的有用工具，以分析用户需求对用户满意的影响为基础，体现了产品性能和用户满意之间的非线性关系。

Noriaki Kano 将影响满意度的因素划分为五个类型，包括：必备需求、期望需求、魅力需求、无差异需求、反向需求。

兴奋（魅力）需求：用户意想不到的，如果不提供次需求，用户满意度不会降低，但是提供次需求，用户满意度会有很大的提升；

期望（意愿）需求：当提供此需求，用户满意度会提升，当不提供此需求，用户满意度会降低；

基本（必备）需求：当优化此需求，用户满意度不会提升，当不提供此需求，用户满意度会大幅下降；

无差异需求：无论提供或者不提供此需求，用户满意度都不会有变化，而且根本不会在意；

反向（逆向）需求：用户根本没有这个需求，提供之后用户满意度反而会下降。

利用 KANO 模型进行需求评估主要集中于对用户需求类型的分类讨论。为了便于分析可以设计相应的调研问卷。

问卷中需要对产品的某项功能分别设置正向和负向两个问题：“如果产品有这个功能，您觉得如何？” 、“如果产品的这个功能不存在，您觉得如何？”

每个问题采用态度量表的形式设计选项，即“我喜欢这样”、“我期望这样”、“我没有意见”、“我可以忍受”、“我讨厌这样”，具体形式如下表：

经过访谈调研后，根据归类矩阵，将调研问题进行归类来确定需求的类型，KANO 模型需求归类矩形如下表：

就能够比较明确地看到，哪些用户需求是必须有的，哪些是用户期望的，哪些是可有可无的，哪些需求又是用户自己不确定的。

将用户需求进行分类，在产品开发时，功能优先级的排序一般是：基本属性>期望属性>兴奋属性>无差异属性，去掉可疑结果的需求和相反的需求。

4. A/B 测试

是为 Web 或 App 界面或流程制作两个或多个版本，分别让组成成分相同（相似）的访客群组（目标人群）随机的访问这些版本，收集各群组的用户体验数据和业务数据，最后分析、评估出最好版本并且实现的综合成本低，正式采用。

比较常见的案例是对网站注册页进行 A/B 测试，确定哪一个方案的注册率高，更加满足用户的需求，实现的商业利益最大化。

需要注意在进行 A/B 测试时，每次必须只测量一个变量，多个变量测试，则无法判断是哪个变量导致的结果；测试的环境应当一直，例如测量时间应一致。

因为在不同的时间段，用户的访问量会有变动；测量的样本量要具有统计学意义，样本流量太小时，无法体现在线用户的真实行为。

七、需求分析优先级的方法

需求优先级的分析方法大致可以分成两大类：定性分析方法、定量分析方法；

一类是根据分析人员的经验主观地对需求进行优先级分类，称之为定性的分析方法，比如：四象限分析法、波士顿矩阵分析法；另一类是根据调查数据，对调查数据进行分析，得出需求的优先级分类，称之为定量的分析方法，比如：KANO 模型。

1. 四象限分析法

根据需求对于业务的影响，以及需求实现的紧迫程度，我们可以按照如下方式将需求归为 4 个象限，这也是需求归类的经典 4 分法。四象限分析法是很常见的一种定性分析需求优先级的方法，如下：

重要且紧急的事，影响业务正常进行，需要尽快处理；

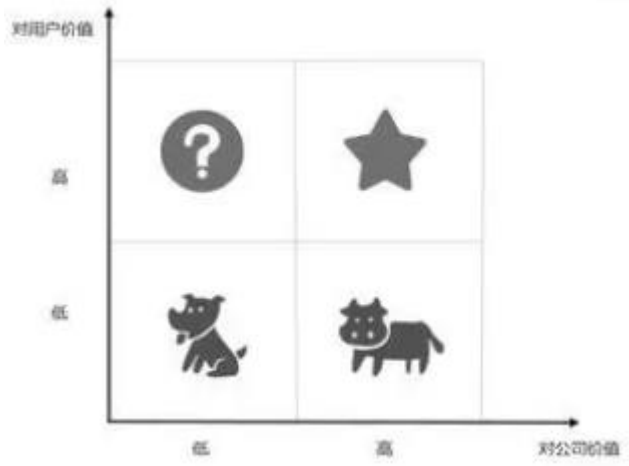
不重要但紧急的事，虽然对业务影响不大，但是需要尽快处理；

重要且不紧急的事，对业务影响大，但不需要短期内就完成；

不紧急且不重要的，对业务影响不大，也不需要短期内完成。

2. 波士顿矩阵

波斯顿矩阵是由波士顿咨询公司发明的一种方法，最早用于分析市场增长率和市场份额，现在也被经常用于对需求的分析之中，波士顿矩阵由用户价值维度和公司价值两个维度将需求分成了四个象限：



明星需求：对用户体验有价值，对公司战略也有价值的需求。明星需求是双赢的需求，需要优先得到满足，如一些促进用户活跃、转化的需求，具体的有，活跃度排名、优惠提醒等功能；

问题需求：对用户体验有价值，但对公司战略和目标没价值的需求。此类需求虽然看似对公司没直接价值，但是提升用户体验有助于提升用户的忠诚度，如一些提升用户体验的需求。具体的有，提供多种快捷登陆方式、提供辅助输入功能等；

金牛需求：对用户体验没价值甚至会对用户造成困扰，但是对公司战略有价值的需求。公司价值的体现，此类需求应该尽量考虑避免对用户造成影响。如一些运营需求等。具体的有，收集用户信息等；

瘦狗需求：对用户体验无价值，对公司战略也无价值的需求。此类需求应该过滤掉，例如一些伪需求。

3. 卡诺 KANO 模型法

Noriaki Kano 将影响满意度的因素划分为五个类型，包括：必备需求、期望需求、魅力需求、无差异需求、反向需求（详情见上文）。

八、如何确定软件需求

经过大量的需求调研工作之后，手上可能有客户提出的大量的、各种各样的需求。

这些需求有的是技术上可以实现的，有的是技术上不可以实现的；有些是管理上需要的，有的是管理上不需要的；有些是合理的，有些是不合理的，如何处理这些需求呢？

以“实现用户正确的需求”为原则，对于用户提出的需求进行严格的分析、甄别。

为了认清用户的需求，先要认清用户。在进行需求调研的时候，会跟各种各样的人员沟通，他们的技术、只是、性格、职位、工作内容各不相同。

但他们也有相似的地方：他们不是做软件的，也不是分析需求的，他们永远不会像你希望的那样去描述需求，他们的需求是用自然语言描述的，是抽象的、概略的、随性的。

那个这些抽象、概略、随性的用户需求转化成具体、详细、结构化的软件需求，是需求分析的重点，通常从以下几点着手认清和控制需求：

1. 将抽象的需求具体化

在需求调研的时候会发现，用户提出自己的需求时总是不会按照你希望方式去提出来，有的人因为不知道你想要什么，只为了应付领导布置的任务，有的是处于比较高的职位，习惯了从宏观的角度去讲问题，所以我们在整理需求的时候要将抽象的要求具体化。

2. 将自然语言描述的需求结构化

用户描述需求总是非常随意的，他们使用平常正常沟通的语言描述，这种需求的主要特点就是不严谨，容易有其一，这种需求不能直接让开发者处理的，开发者需要的需求是描述明确的、精准的、没有歧义的。

需求分析分析者作为用户与开发者的桥梁，有义务将用户用自然语言描述的需求结构化。将用户的描述转换成更精确的语言，更接近 IT 人使用的语言。

3. 注意避免理解偏差

理解偏差主要是需求分析者对用户所提的需求没有理解到位，用户明明想表达的是这个意思，却被理解成了另外一个意思。

这是一个沟通问题，说的人觉得自己说的很清楚了，可偏偏双方就是没有真正理解对方，所以下面是我们需要注意的：

提高沟通能力：多从对方的立场考虑问题，当双方描述某件事时，要从对方的角度思考这些描述；

提高沟通频次：一方面要引导对方多说话，另一方面对不理解的或者觉得理解起来有困难的内容，多向对方询问，换成你的表达方式让对方确认是不是这个意思；

学习对方领域的知识：用户有自己的知识领域，需求分析者也有自己的知识领域，前者满脑子是业务术语，后者满脑子是 IT 术语，有的时候两者真难沟通。每个人的知识面不同，要想沟通顺畅，两个人的知识面重叠的地方越多越好。

4. 识别超出项目范围的需求

用户的需求不能是漫无边际的，所有的需求都应该在项目范围之内，做需求分析的时候首先要确定好项目目标，要让用户知道需求边界在什么地方。

这个项目应该在项目启动时双方经过讨论达成共识，后面所有的工作都应该围绕这个目标展开。原则是即使在这个阶段的目标实现了以后再设置新目标，也不要不停的修改一个目标。

5. 识别错误的需求

对于那些毫无逻辑性、前后矛盾或者在技术上根本无法实现，类似这样的统统归为错误需求。

6. 识别技术上不能实现的需求。

当需求者面向用户时，代表的是身后的整个研发团队，要做好需求分析，需要对自己团队的技术能力有非常清楚的了解，哪些事情能做/不能做，又或者可以做但是需要太大的代价等，每个团队都有自己的技术边界。

九、整理需求

前期做了那么多的收集工作并确定需求之后，要做好需求的整理工作。需求整理是不是简单的将用户所提的需求全部一条条写下来就好了，而是一个综合分析的的整理过程。

通过整理，使得需求更有目的性、更系统性、更明确、更容易理解。需求经过整理后一般会生成需求调研报告与业务流程图，这是后面工作的纲领性文件。

当完成用户需求调查后，首先对《用户需求说明书》进行细化，对比较复杂的用户需求进行建模分析，以帮助软件开发人员更好地理解需求。

十、需求不明确带来的影响

1. 项目失控甚至烂尾

在开发时间和开发费用上的失控，因为需求的不完善，导致启动开发前无法准确预估需求的工作量和确定技术实现方案，走一步看一步开发过程中，发现需求有坑，不断发现新的问题。

有时因为一个简单的逻辑或设计不明确，在沟通明确后最终发现需要技术方案大调整，很多项目会变得失控甚至烂尾。

2. 技术脑补需求

假如需求不是明确的话，靠谱的技术同学，就会自己考虑逻辑和设计，就按他自己的理解和想法实现。

看上去省心，但一千个观众就一千个哈姆雷特，一旦实现的逻辑可能并不是产品期望的逻辑，到了测试环节，测试同学也有自己的理解，导致又要花时间沟通统一意见，或浪费时间返工修改。

3. 沟通成本高

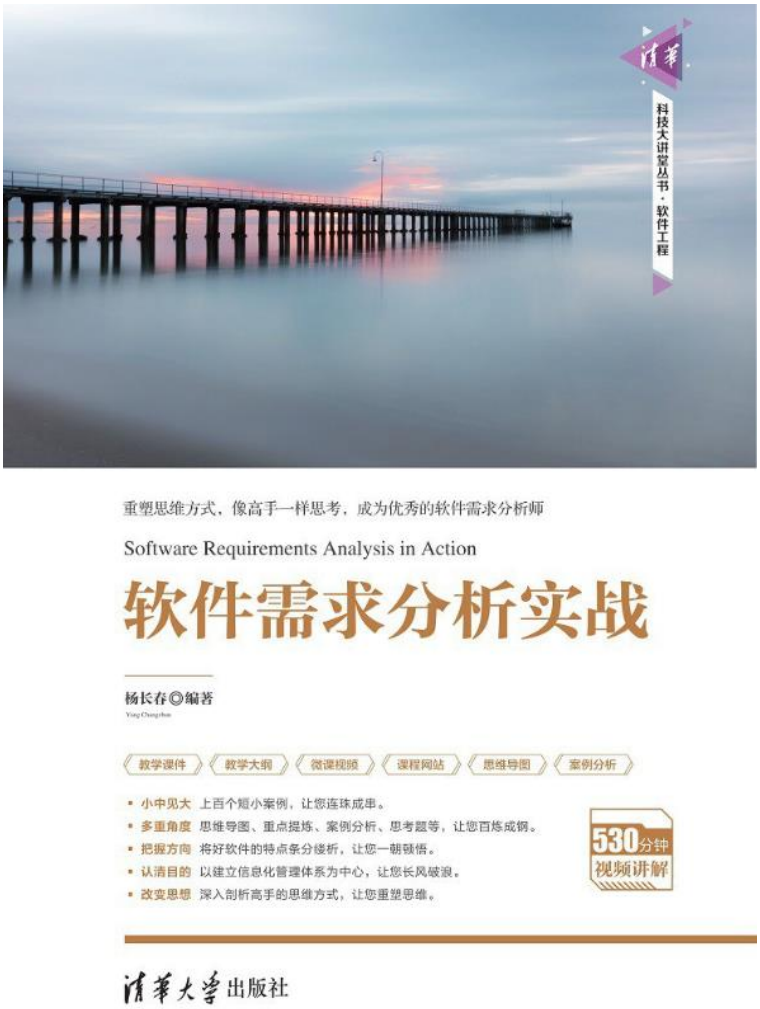
项目规模越大，参与人数越多，矛盾越凸显。

在面对的是人数众多的设计师，前端团队、后端团队、外部团队、测试团队等时，产品经理需经常与设计、技术和测试沟通需求逻辑，沟通的成本会很高。

4. 产品逻辑难以后续追溯

移动互联网时代，产品上线迭代节奏非常快，产品不断的迭代更新，或是人员的交接，经常需要回溯之前的线上逻辑，需求文档的缺失或不完善，会导致线上逻辑不明确，甚至后续的产品需求设计的逻辑与线上矛盾或冲突，为项目的开发带来麻烦。

十一、需求分析推荐书籍



附录：常用的软件需求说明书模板

软件需求说明书，又称软件需求规格说明书，英文名为 Software Requirements Specification（SRS），是需求人员在需求分析阶段需要完成的产物。它的作用是作为用户和软件开发者达成的技术协议书，作为设计工作的基础和依据，作为测试和验收的依据。

软件需求说明书应该完整、一致、精确、无二义性，同时又要简明、易懂、易修改。在一个团队中，须用统一格式的文档进行描述，为了使需求分析描述具有统一的风格，可以采用已有的且能满足项目需要的模板，也可以根据项目特点和开发团队的特点对标准模板进行适当的改动，形成自己的模板。

下面介绍几种常用的软件需求说明书模板：

RUP 版本

Volere 版本

国标 ISO 版本

SERU 版本

RUP 版本

RUP（Rational Unified Process），统一软件开发过程，是一个面向对象且基于网络的程序开发方法论。RUP 是由 Rational 软件公司（Rational 公司被 IBM 并购）创造的软件工程方法。RUP 描述了如何有效地利用商业的可靠的方法开发和部署软件，是一种重量级过程，适用于大型软件团队开发大型项目。

以下是一个标准的 RUP 文档模板：

1.文档概述

这个部分是欧美标准文档通用的格式：首先指出文档的目的、目标读者及各类读者的要点（目的小节)；然后说明项目的背景信息（背景小节)；接着是文档中出现的术语和缩略语（定义、首字母缩写词和缩略语小节)；再接着则是该文档所引用的参考资料（参考资料小节)；最后则是概要地表达本文档的内容（概述小节)。

1.1 目的

1.2 背景

1.3 定义、首字母缩写词和缩略语

1.4 参考资料

1.5 概述

2.整体说明

[让读者对整个软件系统的需求有一个框架性的认识。主要包括产品总体效果、产品功能、用户特征、约束、假设与依赖关系、需求子集等方面的内容。]

2.1 用例模型

2.2 假设与依赖关系

这个部分的内容是用来使读者建立一个框架性认识的。在写作指南中指出，应该写明产品的总体效果、产品功能、用户特征（也就是国标版本中的用户特点）、假设与依赖关系（专门开辟了一个小节)、需求子集（当系统需要多层分解时将使用它）。在 RUP 的相关示例中，还在本部分中使用了上下文关系图。

另外，这个部分显然最重要的是系统的用例模型！因为在 RUP 中，用例是需求组织的核心元素。在此应该给出系统的总体用例图以及用例的简述。

3.具体需求

3.1 用例描述

3.2 补充需求

[易用性、可靠性、性能、其他]

显然这个部分的主要内容就是对用例模型中列出的每个用例进行详细描述。除此之外，还应该说明外部接口、质量属性、设计约束等补充需求。

4.支持信息

根据该文档的写作指南中的信息，这个部分主要是用来提供目录、索引、附录、用户界面原型等信息的，以便令“软件需求规约”更易于使用。

模板说明：

在 RUP 中指出“用例视图是由活动图、用例图、类图组成的”。但上面的文档模板缺失活动图和类图。换句话说，这份需求模板最主要缺少的内容就是串起用例之间的行为逻辑的“流程”信息（用活动图表示），以及描述领域数据之间的关系、它们构成的信息（用类图表示）。

建议要么在这个需求规约的基础上补充这两类信息，要么就用一个 UML 模型来补充它（可以使用 Rose、Together 之类的建模工具）。

实际的应用中该模板中的信息不够完整，因为在 RUP 中所采用的需求描述策略是“模型为主、文档为辅”，也就是说，“需求模型”推荐使用 Rose 创建“用例规约”才是完整的。

Volere 版本

Atlantic System Guild 公司所提供的 Volere 需求过程与软件需求规格说明书模板则充分利用了现代软件工程思想与技术，是一个十分实用、完善的模板。

Part Ⅰ：项目驱动

1.项目的目标

1.1 该项目工作的用户业务或背景

1.2 项目的目标

2. 客户、顾客和其他风险承担者

2.1 客户

2.2 顾客

2.3 其他风险承担者

3. 产品的用户

3.1 产品的直接操作用户

3.2 对用户设定的优先级

3.3 用户参与程度

3.4 维护用户和服务技术人员

PartII:产品限制条件

4. 强制的限制条件

4.1 解决方案的限制条件

4.2 当前系统的实现环境

4.3 伙伴应用或协作应用

4.4 立即可用的软件

4.5 预期的工作地点环境

4.6 进度计划限制条件

4.7 该产品的财务预算

5. 命名惯例和定义

5.1 定义在项目中使用的所有术语，包括同义词

13. 操作需求
13.1 预期的物理环境
13.2 与相邻系统接口的需求
13.3 产品化需求
13.4 发布需求
14. 可维护性和支持需求
14.1 可维护性需求
14.2 支持需求
14.3 适应能力需求
15. 安全需求
15.1 访问控制需求
15.2 完整性需求
15.3 稳私需求
15.4 审计需求
15.5 免疫力需求
16. 文化和政策需求
16.1 文化需求
16.2 政策需求
17. 法律需求
17.1 合法需求
17.2 标准需求
PartV:项目问题
18. 开放式问题
19. 立即可用的解决方案
19.1 已经做好的产品
19.2 可复用的组件
19.3 可以复制的产品
20. 新问题
20.1 对当前环境的影响
20.2 对已实施系统的影响
20.3 潜在的用户问题
20.4 预期的实现环境会存在什么限制新产品的因素
20.5 后续问题
21. 任务
21.1 项目计划

21.2 开发阶段计划

22. 迁移到新产品

22.1 迁移到新产品的需求

22.2 为了新系统，哪些数据必须修改或转换

23. 风险

24. 费用

25. 用户文档和培训

25.1 用户文档需求

25.2 培训需求

26. 后续版本需求

27. 关于解决方案的设想

国标 ISO 版本（1998）

1. 引言

1.1 编写的目的

我经常会在一些需求规格说明书中看到类似于“为了更好地完成本次软件的开发，经过详细的用户调查……”之类的描述。每当看到的时候，我就会问作者：“这段话有什么用呢？”作者通常会回答“那应该写什么呢？”

其实如果你认真地阅读过相应的指南，就会发现本小节的本意应该是：指出本文档所针对的读者对象，以及每类读者对象应该重点阅读的部分。那么软件需求规格说明书有哪些不同的读者对象呢？它们的阅读重点又应该有什么不同呢？

第一个方面的区别显然是读者背景，很多软件需求规格说明书对业务背景的读者照顾不足，这样会导致用户评审变得困难。因此如果在此能够说明不同背景的读者应该重点阅读哪些内容是不错的办法。

第二个方面的区别是用户的层次：高层用户关心业务需求（目标与范围），也就是偏重于阅读任务概述、目标这些小节；中层用户关心业务事件、Stakeholder 关注点,也就是偏重于阅读每个子系统中的分解、流程图以及一些利益点分析；操作层用户关心业务活动，也就是偏重于阅读用例的描述。

第三个方面的区别是用户所属的业务区域，如果需求规格说明书中的内容能够体现这种区域划分，就能够使读者更好地评审。

1.2 背景

1.3 定义

[本文件中用到的专门术语的定义和外文首字母组词的原词组。]

1.4 参考资料

[列出用得着的参考资料。]

2. 任务概述

2.1 目标

[叙述该系统开发的意图、应用目标、作用范围以及其他应向读者说明的有关该系统开发的背景材料。解释被开发系统与其他有关系统之间的关系。]

2.2 用户的特点

[列出本系统的最终用户的特点，充分说明操作人员、维护人员的教育水平和技术专长，以及本系统的预期使用频度。]

2.3 假定和约束

[列出进行本系统开发工作的假定和约束。]

3. 需求规定

3.1 对功能的规定

[用列表的方式，逐项定量和定性地叙述对系统所提出的功能要求，说明输入什么量、经怎么样的处理、得到什么输出，说明系统的容量，包括系统应支持的终端数和应支持的并行操作的用户数等指标。]

这个部分是最为核心的内容，通常会占整份文档的 60%～70%的篇幅；可是模板中只是定义了该小节，里面该如何组织并没有明确地定义出来，那么如何保证每个人写出来的东西具有相近的风格呢？

这实际上就是通用型模板和企业模板的区别，作为通用型模板是无法定义到更细的程度的，因为不同的软件项目、不同的团队、不同的开发方法必然会采用不同的组织方法。因此我们在使用之前，必须对这个部分做进一步的细化。

另外，在该标准中建议直接用“条目”来定义需求，而这种方法最大的问题是数量大、粒度不统一，因此现代软件需求理论实际上更建议使用用例、用户故事、特性之类的组织单元来代替这种风格的表述。

3.2 对性能的规定

这个部分经常被理解成非功能需求的全部，同时导致了一种误解：所有非功能需求都应该单独地罗列出来，这样都经常出现“全局性”非功能需求，但实际上有很多是“局部性”的，我们应该让它和相应的功能需求同时出现。

其实，如果我们认真阅读指南就会发现，在针对“对功能的规定”小节的指南中，有一句“说明系统的容量，包括系统应支持的终端数和应支持的并行操作的用户数等指标”，这显然是非功能需求。

3.2.1 精度

3.2.2 时间特性要求

3.2.3 灵活性

3.3 输入/输出要求

这个部分具有很强的历史性，在当时输入、输出设备并不是只有键盘、鼠标、打印机，还有列表机、纸带机等形式多样、功能不同的其他设备。因此在软件开发之前,是有必要做相应描述的。

而在今天的很多软件中，这个部分是比较单一的，因此可以考虑将这个部分内容从模板中去除。

3.4 数据管理能力要求（针对软件系统）

在该标准制定时，数据库并不是最流行的或唯一的数据管理机制，开发人员可能会需要直接使用外部文件等方法。而今天的情况完全不同，数据管理基本上是交由数据库管理系统实现的，因此通常也不会需要本小节。

不过也有一些软件系统有例外的情况，例如地理信息系统、实验数据管理平台都可能涉及这一部分。对于地理信息系统而言，有许多数据不是直接存储到数据库中的，它涉及各种图层信息应该以什么格式保存、如何保存、如何组织等；对于实验数据管理平台也是这样，数据量极大，数据库无法解决，需要开发人员另想办法，这也就涉及到了数据管理能力方面的信息。

3.5 故障处理要求

这个部分在很多信息系统中是相对较弱的，因此可以将其合并到“其他专门要求”中，或者归类为“补充规约”也是不错的做法。

3.6 其他专门要求

4. 运行环境规定

这个部分的信息，现在的绝大多数软件也都会遇到；在 UML 中建议用部署图对其进行建模，并纳入“补充规约”中。在这份模板中定义了以下几个方面内容：

•设备：用到的硬件资源；

•软件：预期的操作系统、中间件、数据库、第三方软件等软件环境资源；

•接口：所需的外部系统接口；

•控制：这个部分有些过时，在诸如 CIMS(计算机集成制造系统）之类的软件中就经常会涉及控制信号方面的规定。

4.1 设备

[列出运行该软件所需要的硬设备。说明新型设备及其专门功能。]

4.2 支持软件

[列出支持软件，包括要用到的操作系统、编译程序、测试支持软件等。]

4.3 接口

[说明该系统同其他系统之间的接口、数据通信协议等。]

4.4 控制

[说明控制该系统的运行方法和控制信号，并说明这些控制信号的来源。]

国标 ISO 版本（2006）

新模板的第 1 和第 2 小节实际上起到的作用还是“引言”的作用，不过在具体的内容上还是做出了不小的调整。

首先是做了一些扩展，包括将“编写目的”扩充为“文档概述”，将“背景”扩展为“系统概述”，同时也更明确地指出了它应该包含的内容。

然后是做了一些剪裁和调整，包括将“定义”小节去掉了（放在最后的“注释”部分中），这和我们前面的分析比较类似，该信息更适合单独来管理；另外还将“参考资料”更精确地定义为“引用文件”，并将其作为单独的章节。

另外还做了一些补充，增加了“标识”和“基线”，它们是近代软件工程理论的产物，用来实现需求的跟踪和需求基线管理。

1. 范围

1.1 标识

[本文档适用的系统和软件的完整标识]

1.2 系统概述

[适用的系统和软件的用途；开发、运行、维护历史]

1.3 文档概述

[文档的用途和内容]

1.4 基线

2. 引用文件

3. 需求

3.1 所需的状态和方式

[软件项是否在多种状态和方式下运行]

3.2 需求概述

3.2.1 目标

[表述系统的目标和范围]

3.2.2 运行环境

3.2.3 用户特点

3.2.4 关键点

[关键功能、关键算法、关键技术]

3.2.5 约束条件

3.3 需求规格

3.3.1 软件系统总体功能/对象结构

[对软件系统总体功能/对象结构进行描述，包括结构图、流程图或对象图]

3.3.2 软件子系统功能/对象结构

[对每个主要子系统中的基本功能模块/对象结构进行描述，包括结构图、流程图或对象图]

3.3.3 描述约定

3.4 软件配置项能力要求

[可用功能、性能、目标或类似词代替“能力”]

3.4. X

[包括能力的说明、输入、处理、输出]

3.5 外部接口需求

3.5.1 接口标识和接口图

3.5. X 具体接口

[说明接口优先级、接口类型、数据元素特性、数据元素集合、接口通信方法、必须使用的接口协议等]

3.6 内部接口需求

3.7 内部数据需求

3.8 适应性需求

[提供的依赖于安装的数据有关的需求]

3.9 保密性需求

[诸如防止意外动作和无效动作所必须提供的安全措施]

3.10 保密性和私密性需求

3.11 环境需求

3.12 计算机资源需求

3.12.1 计算机硬件需求

3.12.2 计算机硬件资源利用需求

3.12.3 计算机软件需求

3.12.4 计算机通信需求

3.13 软件质量因素

3.14 设计和实现的约束

3.15 数据

3.16 操作

3.17 故障处理

3.18 算法说明

3.19 有关人员需求

3.20 有关培训需求

3.21 有关后勤需求

3.22 其他需求

3.23 包装需求

3.24 需求的优先次序和关键程度

4. 合格性规定

[可以独立，也可以直接在前面注明方法，包括演示、测试、分析、审查、其他特殊方法]

5. 需求可追踪性

6. 尚未解决问题

7. 注释

新版本相对老版本做了以下两点的更新：

①对“需求规定”进行了详细的分拆。

新模板一改上一版本中简单的风格，对“需求规定”小节提供了更加完整、全面的分解，并且将原先的“运行环境规定”部分也整合进来了。但是从最终的效果上来看，通用性被强化了，适用性被减弱了，因此在实际应用中应根据需要对其进行必要的剪裁。

另外，为了帮助大家能够更好地理解新模板中对“需求规定”小节所分出的24 个子部分，我们再对其做一些简要的说明，如下表所示：

②增加了一些提升文档功能性的内容

新模板的第 4～7 小节实际上是为了提高整个软件需求规格说明书的功能性；合格性规定是针对测试团队、验收团队的信息；需求可追踪性是针对项目管理人员的信息；尚未解决问题、注释则是为了帮助读者更好地理解、使用文档。

SERU 版本

SERU 是一套系统全面的需求方法论，适合中国国情，尤其对 ToB 类软件的需求文档编写有较好的指导作用。

1.文档概述

在本小节中将说明整个文档所针对的不同读者群（编写目的）、整个项目的背景和概况信息（背景)、文档中常用的技术缩略语及相关词条（定义)，以及文档编写时所需引用、参考的资料（参考资料）。

1.1 编写的目的

1.2 背景

1.3 定义

1.4 参考资料

在老版本的国标模板、RUP 提供的模板中都有相似小节，这个部分是规范的文档所应该具有的信息。

2. 任务概述

2.1 业务需求

2.2 Stakeholder 利益分析

2.3 用户特点分析

2.4 相关事实与假定

也就是项目目标、Stakeholder 关注点、用户特点、相关事实与假定之类的概述性信息，这部分信息通常是需求定义（项目立项）阶段就需要确定的，属于业务需求的范畴，是中高层用户代表所关心的内容。这部分内容单独地放在一个小节中。当然，这个阶段还将产生一个十分重要的内容，也就是项目范围，但为了便于阅读，将其作为第 3 部分（需求概述）的纲要出现。

3. 需求概述

3.1 系统概述

[主题域划分，用构件图表述]

3.2 主题域 1

3.2.1 概述

[用上下文关系图表示该主题域的范围]

3.2.2 业务事件

3.2.2.1 业务事件 1

[包括流程分析、领域类分析、用例分析]

3.2.2.n 业务事件 N

3.2.3 报表

3.2.3.1 Report1

[用领域类图片段表示涉及数据，用用例标识具体的报表项]

3.2.3.N Reportn

3.3 主题域 n

以上部分和第 4 部分（具体需求部分)，实际上是对老版本的国标模板中“3.1 对功能的规定”小节的分解，以便让软件需求规格说明书的作者更加有的放矢地组织所需的信息。

需求概述部分的主要内容是针对中层用户代表的，其核心内容在于对业务知识的梳理，目标在于导出系统的用例模型和领域模型，是需求分析第一阶段（理清框架和脉络）的工作任务。简单地说，就是：

•首先将系统按业务的特点分成不同的子问题域（主题域)，并且通过构件图整理出它们之间的接口。

•对每个子问题域（主题域）进行分解，标识出与系统相关的业务流程（业务事件是流程的起点）、报表类型。

•然后绘制每个业务流程的流程图，将流程中涉及的业务实体之间的关系、结构规则用领域类图片段表示出来，并根据“是否在系统边界之内”的原则从流程图中派生出用例图。

•同时对于每类报表而言，用领域类图片段表示出其涉及的业务实体，用用例图表示具体的报表项。

因此这个部分实际上就是采用一个业务驱动的树型结构（主题域、业务流程、报表类型）贯穿的业务知识，它框出了系统所需完成的行为需求，以及它涉及的结构需求。

这部分内容在老版本的国标模板中并没有涉及，但在新版本的国标模板中就增加了这方面的信息，即“3.3 需求规格”小节，但具体的组织方法未定义。而在 RUP 版本的模板中也没有明确地指出，但这部分信息实际上是从属于需求模型的。

4. 具体需求

4.1 主题域 1

4.1.1 用例模型

4.1.1.1UC_B_xx(B 类)

(1) 概述[编号、名称、概述、相关 Stakeholder]

(2) 事件流描述[前、后置条件，基本、扩展、子事件流]

(3) 相关需求与功能点

(4) 界面原型[交互过程与界面详解]

(5) 规约与约束

4.1.1.2UC_R_xx(R 类)

(1) 概述

[名称、用户部门与职位、业务意图、相关场景]

(2) 报表内容[领域类图、数据项]

(3) 输入/输出格式

(4) 其他

4.1.1.3UC_I_xx(I 类)

(1)使用者[名称、业务目的、时机、频率]

(2)内容与格式[交互过程、数据包说明]

(3)设计与实现约束[诸如协议格式要求、性能要求等]

4.1.2 领域模型

4.1.2.1xx 领域类

(1)概述

[类名称、别名]

(2)数据窗口分析

[涉及主题域、业务事件，各域数据]

(3)数据组成与格式

(4)其他

4.N 主题域 n

以上部分是针对具体的开发人员和操作层的用户代表的，它将描述最为细节的需求信息；由于该模板是针对“用例分析技术”的，因此选择“用例”作为行为需求的最小单元，“领域类”作为结构需求的最小单元；换句话说，就是填充用例和领域类的细节。

在这个部分中，参考了 RUP 版本的模板，并对用例模板进行了适当的扩展，将每个具体的报表项定义为一个报表类（R)用例，每个具体的接口定义为一个接口类（I)用例，同时为它们分别定义了不同的格式模板。另外，也对领域类应该填充什么方面的内容进行了约定。

而在国标版本（包括老版本和新版本）、Volere 版本的模板中，由于它们并不限定某种需求分析技术，因此采用了更通用的表示方法；因此在使用之前，建议还是应该对其做进一步的定义与约定。

5. 补充规约

5.1 设计约束
5.1.1 技术选择的限制条件
5.1_2 运行环境
[建议使用部署图表示]
5.1.3 预期的使用环境
5.2 质量属性
[本部分建议直接分解成需要开发的技术功能点]
5.2.1 安全性要求
5.2.1.1 访问安全性要求
5.2.1.2 数据安全性要求
5.2.1.3 通信安全性要求
5.2.1.4 其他安全性要求
5.2.2 可靠性要求
5.2.2.1 容错性要求
5.2.2.2 可恢复性要求
5.2.2.3 其他可靠性要求
5.2.3 易用性要求
5.2.3.1 界面友好性要求
5.2.3.2 易操作性要求
5.2.3.3 其他易用性要求
5.2.4 性能要求
5.2.4.1 数据访问性能要求
5.2.4.2 数据传输性能要求
5.2.4.3 其他性能要求
5.2.5 可维护性要求
5.2.5.1 公共数据要求
5.2.5.2 公共框架开发要求
5.2.5.3 公共程序库开发要求
5.2.5.4 其他可维护性要求
5.2.6 可移植性要求
5.2.6.1 适应性要求
5.2.6.2 易安装性要求
5.2.6.3 其他可移植性要求
5.2.7 其他质量属性要求
5.3 其他需求

5.3.1 培训需求

5.3.2 后勤需求

5.3.3 包装需求

除了结构需求、行为需求之外的其他需求都应放在“补充规约”这一部分中，在 RUP 版本的模板中采用了相同的组织方法；而在国标模板（包括新、老版本）、Volere 模板中则是将它们打开，每个部分一个小节，区别仅在于展开的程度各有不同。

我们根据在业务系统中经常涉及的内容对补充规约分成了三类：一是设计约束；二是质量属性；三是其他内容。

对于设计约束而言应该包括技术选择的限制条件（也就是非技术因素决定的技术选型）、运行环境（也就是预期的软、硬件环境，通常可以使用部署图表示）以及预期的使用环境。这些内容在新版本的国标模板中也有详细的划分。

而对于质量属性而言，建议直接分解到要开发什么功能上，并根据对开发的影响提供了一个树形结构。

此外还可以涉及培训、后勤、包装、升级等其他方面的需求，这在新国标版本的模板、Volere 模板中都归纳了一些，大家可以参考。