# 架构师

ARCHITECT

特刊

# ArchSummit

深圳·2019

ArchSummit 全球架构师峰会 | Geekbang. 极客邦科技 InfoQ

# 目录
## Contents



ArchSummit 全球架构师峰会是 InfoQ 中国团队推出的重点面向高端技术管理者、架构师的技术会议，54% 参会者拥有 8 年以上工作经验。

# 卷首语：架构就是不断的选择、妥协与修正

作者：ArchSummit 组委会

在武侠世界里，高手之间需要不断的切磋，水平才能有所提升。在信息开放环境，和不同的高手交流，是保持自己信息处于最新状态的有效方法。

《人人都是产品经理》这本书大家应该不陌生。其实技术人也一样，不管是那个技术方向，都需要了解一些架构知识。比如前端的工程师需要了解架构，用什么框架，Web 方式还是 Native 方式？跨平台需要考虑如何测试，如何监测性能，如何确保安全性？这些都会遇到无数个架构选择。

机器学习工程师更需要了解架构，算法工程师开发出最前沿的算法模型，但是如果他不懂架构，不知道如何和大规模数据结合起来落地，就会重现“产品和研发打架”的情形。

架构能力取决于你的选择。有人说，人生就是不断选择的过程，其实即使作出选择，后期也会持续的变化、修正，这是一个不断积累的过程。从前端到后端，从大数据到人工智能，每个领域都有很多框架轮子。但是真的有必要过度追逐最新的技术吗？首先要看是否对业务或者系统有帮助，还要考虑投资回报比。此外还要关注所采用的框架作者或者社区是否能长期维护这个框架，现在很多技术像潮汐，来的快，退的也快，最好选择使用较为广发，靠得住的技术。

架构设计对于可扩展性要求较高，很大程度涉及到 API 的设计思路和服务边界，用 API 来支撑起服务的复杂度。系统简单的时候，如何选择都可以；但时间久了，系统复杂度很难管理。以搜索引擎前端为例，看上去很简单，但是有不同的团队在增加功能，需要适配不同的设备，Android、iOS 等，这就需要设计出“既不过于复杂，也不过于简单，恰到好处”的 API。

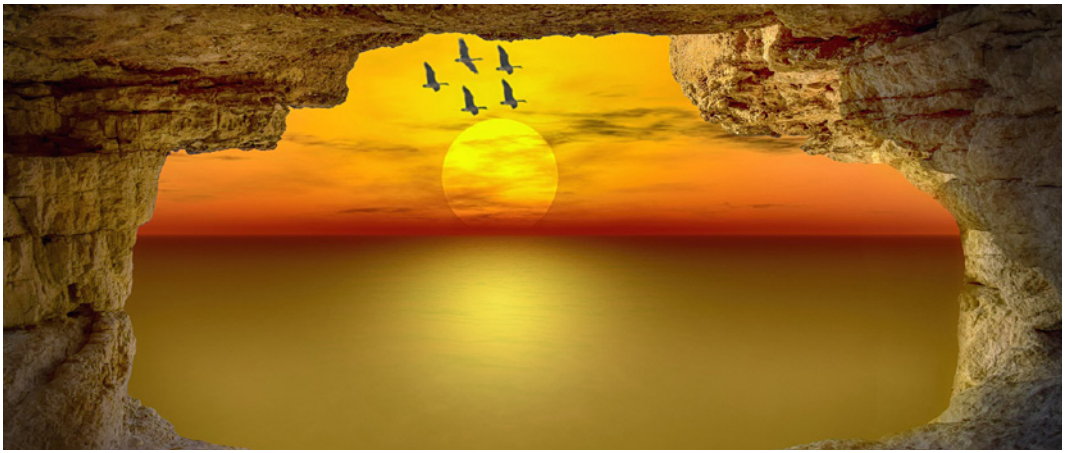
易用性是架构师该重点考虑的，因为你设计出来的产品是要给别人用的，用“同理心”去解决别人可能会遇到的问题。

另外，架构是需要不断演化不断优化的，无法维护那就重构。大多数技术负责人都很痛苦的一点是，很多历史遗留的 API 很难淘汰，维护很困难，用户又很多。这就需要在技术和业务之间做平衡，技术服务于业务的方向是对的，但是也不能受限于业务。就像最近讨论较多的“大中台、小前台”架构模式也是很好的尝试。

伟大的架构需要长期的磨练和时间的验证，过去 10 年里，很多大公司提了技术方案名称，但都消逝在时间浪潮里。技术的前进需要领头公司在系统上大量的投入，即使不确定方案是否奏效。架构师也是一样，在一线底层搭建系统实践，训练自己，沉住气，沉淀厚度。

# 蘑菇街陈辉谈架构师业务思维修炼术

作者 陈辉



架构师是一个承前启后的岗位，正如一个路由器，对各种业务需求加以分析处理后路由到下游产品和研发团队。所以，对于业务的理解能力是必不可少的。7 月 12 日深圳架构师峰会上，邀请到了蘑菇街架构师陈辉老师来演讲架构师该如何提升自己的业务思维。以下是陈辉老师的回复整理。

目前，我的日常工作主要包括电商架构体系的规划和建设，结合对业务方向的变化以及存在的问题，打造一个能够稍稍跑在业务前面的架构。

传统电商架构向业务中台架构的演进工作，当前蘑菇街的业务结构不仅仅包括电商，还包括直播和 mogu 内容，需要对整体的技术架构做一些演进。

此外，也会重点去做基础平台的架构升级，架构逐步云原生化的工作筹备。平时还参与静态化服务器的开发，主要还是从部署架构上的变化带来成本优化。

## 架构师业务思维关键词

### 主动性

做业务不是等着别人来找你，这就变成了做需求的了，业务思维很重要的一点就是要能主动出击。

### 平衡性

不能一味的用业务的模式来解决问题，也不能固执的用技术思维来确定方案，两者需要做好平衡。

### 前瞻性

架构先行一定是需要有一定的业务前瞻性，虽然架构师都知道设计出高扩展性的架构，这个扩展性很大一部分体现在对业务的适配上。

大部分情况下很多开发或者架构师都会犯的错误就是过度设计，这个过度设计很多时候并不是没有业务思维，而是技术思维占据主导作用，用很技术的手段来思考业务问题。

案例：电商这边会有一个比较基础的系统称为库存中心，库存中心主要负责电商下单过程中的库存扣减和交易逆向流程回补工作。系统设计技术难点主要还是集中在防超卖上，类似《库存中心高并发实践》的文章在网上都能搜到。

当时在设计架构的时候，从技术思维上，蘑菇街花了更多的精力在怎么能达到更好的性能上，而没有好好结合业务发展的形态，只是幻想后期业务变化快，在扩展性上大做文章。可是一段时间后发现，其实系统能力超过了业务能力几倍，但是业务变化起来，系统要跟上业务节奏就会比较累。

## 有和没有“业务思维”的区别

在带基础平台之前，主要作为电商和基础平台的对接人，发现两边的开发其实会有非常大的差异。电商的开发总觉得中间件怎么那么难用，基础平台的开发总不理解业务到底在做什么。两边很多合作类的项目要开展，但是协作成本很大。在此过程中我明显的体会到，缺少业务思维的开发往往容易钻技术牛角尖，在沟通协作上，大家很难从公司的维度去达成一致，更多的会从技术的角度去聊问题，这样带来最大的问题就是方案到最后可能是技术最优雅，但从成本、长期收益等角度看，存在很多潜在问题。

架构师很大的一个工作就是要把自己的想法落地，除了技术上能让大家接受外，更多的其实是要能带着大家创造价值。具备业务思维，可以更好的去权衡利弊，能够让沟通的维度更多，但方向更一致。

## 培养业务架构思维的阶段

从个人经验来看，尤其是对于业务架构师来说，思维的培养需要经历这几个阶段：

一、心中有解决方案，架构师很大的一部分设计来源于之前经验和解决方案的积累，所以解决方案的积累阶段比较重要。这部分的建议，一方面在日常工作过程中，一定不要只关注眼前做的那部分东西，要学会横向去看，去对比，对标行业的解决方案和思路；另外可以多关注类似极客时间 App 的专栏，或者 QCon，ArchSummit 这样的大会，上面会有很多不错的思路可以借鉴，开阔眼界真的很重要。

二、对业务的理解和把握能力，一般参与到业务中会经历几个阶段：理解分析业务、参与到业务中、对业务有前瞻性；

三、权衡技术思维和业务思维的阶段，这个阶段就需要实践来看什么样的场景下该偏重什么样的思维。说到权衡的维度，在蘑菇街业务背景下，我的标准就是客户价值，比如，我的设计是否真的解决了客户的实际问题，哪怕是一个偏平台类的系统设计，我也会从实际的客户价值上去判断，不能因为是平台型的就一定会很技术性。

四、经验之外的创新，架构设计不可能一味的用固有的体系，如何权衡业务，如何带来创新，是这个阶段需要考虑的问题。

## 业务转型，架构先行

架构师业务思维如何推动公司转型？业务要变化，并不是今天说变，明天就直接变了。业务的演变需要一个过程，技术要做的是怎么缩短这个等待的时间。这其中很关键的能力就是能不能提前做好技术布局？比如 2016 年，蘑菇街和美丽说融合，以当时两家平台的体量，在一个月内完成从数据到系统的全部融合是个非常不可思议的项目，如果不是之前在系统上做了平台化的能力，相信这个项目就很难去落地。电商系统的平台化应该也是发展到某个阶段需要去落地的实现，平台化的能力主要是让交易、商品等电商的


核心链路可以支持多平台的正向、逆向流程，支持多平台业务的混合部署、隔离运行。电商的发展还是能够看到一些预期和苗头的，所以技术上的提前布局，后期发展就水到渠成，容易很多。

业务稳定了，还需要调整、创新？对于这个问题，其实在整个架构升级和调整的过程中，蘑菇街也会做一些技术上的创新和试验。比如蘑菇街会在局部尝试 Serverless 概念，以及 Cloud Native 开发。并不是说完全适配业务、成本最低化、没有任何创新的架构就是最合适的，在考虑性价比的同时适当的做一些技术层面的尝试，来决定后续是否可以在这上面更深入进行。技术的创新很多时候可以带给业务更多的可能性，所以创新无止尽。

在阿里提出中台战略的很长一段时间里，我自己也一直在思考蘑菇街的技术架构应该怎么去演进，才能更好的适应大中台小前台这样的业务组织模式。随着蘑菇街业务的复杂度不断提升、业务不断多元化发展，也正在摸索自己的中台架构，期望通过一种平台化的基础层来支持多渠道业务的扩展，建立核心的组件库来支持业务的复用，并通过轻量级的业务组装形态来完成业务的迭代。

## 团队成员的业务思维培养

首先是要走出去，和业务线的同学待在一起，和客户待在一起，了解他们在干什么，遇到什么困难，有什么解决思路？其次是，不仅仅是技术能力，对业务的理解要深入，对周边知识体系的了解都能决定对问题的看法，都可以影响做出来的设计，最关键是要有创新，要学会取巧，架构设计也能有创意，适合业务的才是最好的。

## 嘉宾介绍

陈辉：2015 年加入蘑菇街。目前在蘑菇街负责基础技术平台（中间件、稳定性、效能工具、网关）和电商技术架构相关工作，主要职责是在业务方向下，明确技术规划和技术架构，并落地技术输出。曾负责或参与蘑菇街前后端技术架构升级、静态化改造、网关统一化、上云等横向性工作。

# 专访阿里亚顿：Serverless 正在颠覆开发模式，包括对工种的定义

作者 杨凯


Serverless 是一种“无服务器架构”模式，它无需关心程序运行环境、资源及数量，只需要将精力聚焦到业务逻辑上的技术。目前很多公司已经实现 DevOps 化，正在向 Serverless 迈进。而前端工程师也要关注 Serverless，因为它可能会改变前后端联调方式，亦可大幅度降低 Node.js 服务器维护门槛。

ArchSummit 全球架构师峰会上，来自阿里的高级前端专家亚顿将分享《BFF in Serverless》话题，在此之前，亚顿老师把他在实践 Serverless 过程中的一些技术解决思路分享给大家，以飨读者。

## 采用 Serverless 理念对 BFF 层进行改造

亚顿说，在传统基于 Node.js 的 BFF 层，其痛点主要在于存在较高的发布和运维成本，而引入 Serverless 的关键目标就是要解决这两个问题。因此，为了提高发布速度、降低运维成本，团队将 BFF 层的函数全部转换为可动态执行的脚本并保存到数据库中，同时提供统一入口用于函数的路由分发，这是阿里团队改造中最核心的功能。围绕这一核心，为了能提高用户体验以及开发效率，团队还打造了针对用户的统一入口和针对开发者的控制台。

在改造的过程中，至关重要的两个问题是：

1）存量应用如何平滑迁移？如果新方案和传统方式差异过大，那较高的迁移成本将会阻碍改造计划的全面推广落地。

2）稳定性如何保障？即如何确保函数运行的沙箱环境的隔离性和安全性，防止函数因自身

影响整个平台或其它函数的运行。这是我们最应该关注的两个问题。

## Serverless 架构分层设计实践

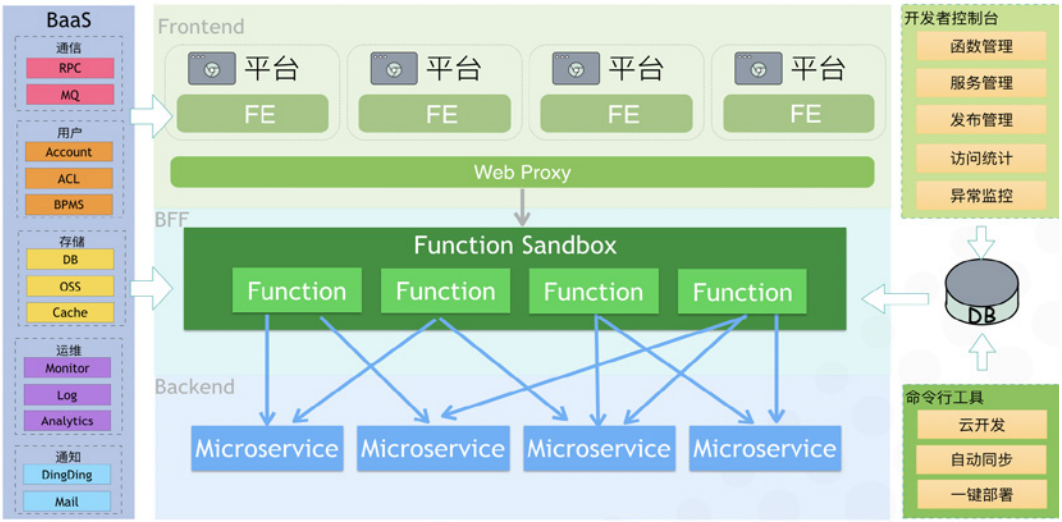
在架构分层中，主要包含运行与开发两种时态。在运行时阿里团队将其分为 FaaS 和 BaaS 这两大核心模块，即提供安全运行函数片段能力的 function sandbox runtime 和可以在函数中调用各种后端服务的 BaaS Service，其关注的重点是稳定和能力。而在开发时，主要提供了支持在线开发、配置、调试、发布、回滚、监控等能力的一站式开发者控制台及独立 CLI，使开发者可以轻松创建和管理函数，其注重的是开发者的体验。

也许有人会问，当前 BFF 有什么样的最佳实践，有哪些公司已经标准化这样的做法？亚顿回复说，首先，目前进行 BFF 实践的团队或公司，几乎无一例外都采用了 Node.js 来实现，其实这并不是偶然。BFF 就是 UI 的粘合层，而 UI 通常都由前端人员在开发和维护，所以 BFF 层也自然由前端人员采用了对其来说比较顺手的工具来实现。所以，这是最重要的实践理念：服务自治，即吃自己的狗食。其次，对于 BFF 层的价值，是将后端服务聚合和裁剪，为 UI 提供 API。故第二个实践理念：BFF 只处理 UI 逻辑，业务逻辑下沉至后端服务。在不同公司由于基础设施和场景不同，所以很难存在一种标准化的实践方案，但整体方案上只要不偏离上面两个理念，我们认为这就是当前场景下的最佳实践。

前端在使用 Serverless 服务时，亚顿认为最主要痛点在于基础设施的不完善。CNCF 针对 Serverless 给出了他们的定义：Serverless = FaaS + BaaS，然而目前大家主要还是聚焦在前者，对 BaaS 的关注度相对较少。虽然阿里有了一套完善的函数运行时，但真正的业务通常是无法通过一个纯函数的执行而中间不调用任何其它依赖（比如 RPC、DB、Cache、MQ 等）就能独立完成的。所以，亚顿团队花了大量的精力将相关依赖封装起来，形成一套统一的 BaaS SDK 供函数调用，使其能完成以往在 BFF 中能完成的所有工作。

## Serverless & GraphQL

Serverless 是否应该与 GraphQL 结合？如果是的话是否会提升应用开发的复杂度，如何权衡？亚顿说，对于 UI 层来讲，使用 GraphQL 提供 API 确实是一个不错的实践，可以真正的实现按需查询，但其也存在相应的问题。比如在有 BFF 的情况下，它增加了 BFF 层的工作量，需要



将所有后端服务都封装一遍来支持 GraphQL 协议；另外也增加了 Android、iOS 的工作量，你也得写两份的查询语句。目前就大家遇到的场景来说，GraphQL 其所能够提供的价值，和增加的工作量相比优势不够显著。不过大家仍然认为它一个不错的实践，如果在有适当的场景将会继续进行尝试。

当前很少看到 Serverless 大规模使用的案例，究其原因，最大障碍在于配套依赖的不完善。对于使用各种云服务的公司来说，目前大多数云服务商提供的 FaaS 服务相对来说是独立存在的，没有完全和自己的后端服务打通，这给 FaaS 的大规模应用带来了极大的不便；而像 BAT 这样的公司，由于其各自内部中间件服务非常丰富，要将 FaaS 与这些中间件服务逐个打通，也是一个不小的挑战。

这就衍生出另外一个问题，传统模式向 Serverless 模式的转变存在哪些阻力，如何克服？亚顿认为：研发模式转变，需要考虑三个问题：

1）新方案能否提供足够的价值；

2）线上应用如何迁移；

3）新方案带来的新问题，我们能否接受。

第一点其优势想必已不必多谈。对于第二点，如果是一个历史包袱不多的新团队或公司，这并不是一个问题，但对于已有大量线上应用的团队或公司，应用的平滑迁移是需要重点考虑的。第三点，亚顿认为目前 Serverless 存在的最大问题是缺乏标准。由于团队将原来的 BFF 应用打散成了一个一个的函数，那么如何将这些函数有效的组织起来是需要思考的问题。不仅是在组织上缺乏标准，在实现上同样如此。目前各大云厂商都是基于自己的理念各自实现其框架，这导致以后几乎不可能完成云厂商的平滑切换。可喜的是已经看到 CNCF 发布了第一版 Serverless 白皮书，使我们离标准化更近了一步；同时也出现了 Serverless Framework 这样的框架，抹平了不用平台服务的差异，能一定程度上解决这个潜在风险。

## Serverless 对人及技术管理的影响

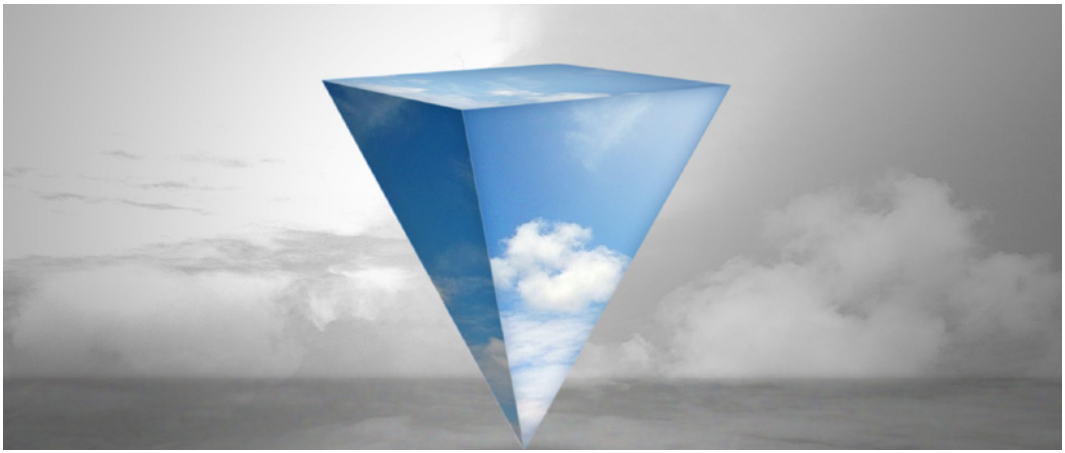
Serverless 只是全面云服务大趋势下的一个缩影，基础设施最终都将由 Provider 提供，作为 Developer 只需关注在这种模式下如何有效的设计和组织业务架构。脱离 BFF 场景，当 BaaS 的能力逐渐增强之后，前端可以独立完成以往需要后端才能完成的那部分工作，这将使前端向全栈的方向进一步演化；而后端将进一步下沉，将原有的一部分业务组装逻辑交由前端完成，自身去实现更加底层的通用业务封装，也就是常说的“大中台，小前台”。

亚顿个人认为后续工种将不会再分为前端、后端，而是产品研发和中台研发：产品研发负责所有的上层业务逻辑组装（如下单支付），而其中要使用到的一系列底层业务平台（如用户中心、订单中心、支付中心、物流中心），由中台研发负责。

所以，亚顿认为，对业务流程的深入理解和全局把控，将是今后前端人员的一项新的挑战和方向。

# 作为中台倡导者，百度如何利用“搜索中台”实现月级别孵化新产品

作者 徐川 Tina



如何提高软件开发的复用度是全世界的 IT 企业都面临的问题，但“中台”这个词目前却找不到任何对应的英文词汇，某种程度上来说，它具有一定的中国特色。最近半年，关于中台的讨论热度甚至达到了前所未有的程度。

往前追溯，中台思想是在 15 年马云带领阿里技术团队参观完游戏公司 Supercell 后提出。Supercell 的做法是将游戏开发过程中公共和通用的游戏素材和算法整合起来，为小团队提供工作的工具和框架，从而支持好几个小团队能够在短时间内开发出一款新的游戏，并鼓励员工充分试错。半年后，阿里宣布了组织架构的全面升级，全面启动中台战略，“中台”这个词因此大火。

国内让“中台”讨论热度达到沸点的是百度、腾讯等企业，他们针对实现“中台”做出的调整企业架构整合技术体系的举动：面向 ToB，建设“中台”。百度高级副总裁王海峰曾表示，百度于 2018 年 12 月份进行的组织架构调整中，将打造技术中台作为战略方向之一。

追看历史，大公司的管理层提出“中台”策略之目的是为了快速支撑和响应业务，而具体怎么去实现“中台”，更是没有统一的方法，需要结合公司自身的情况。

那这些“中台倡导者”是如何实现他们的中台的呢？百度搜索在阿拉丁的基础上构建了多个垂直领域的搜索产品，也扩展了更多的流量入口，包括小程序、独立站等，产品形态日趋复杂。为了满足多场景、多产品的高频创新需求，以低成本落地通用能力为目标，进行了平台升级，完成了垂直搜索产

品技术中台的打造。

百度垂直行业 & 阿拉丁架构技术负责人张安站在接受 InfoQ 采访时表示，利用百度的搜索中台孵化新的垂直搜索产品，从学习中台系统、开发、部署，到完成线上效果验证的全流程，所用的时间可以缩短到一个月内。

以下是张安站介绍其所在团队是如何通过技术实现各个业务的创新叠加，促进各个创新业务的快速可持续发展的解读。

## 1. 如何看待中台与平台的关系

平台是提供特定的服务，平台一般有确定的接口和功能集合，业务可以按照规范使用平台；平台提供的是通用的、确定的服务，而不关注业务是如何使用的。在企业内部，平台是跨部门、跨体系共享技术的重要方式，降低了企业的技术研发成本，因此很多互联网企业采用了平台化战略。

中台在某个领域内提供了可定制的通用解决方案，通过可定制能力来满足细分领域的需求。一般来说，具备多场景的复杂领域都可以产生自己领域的中台，比如搜索、电商、推荐等，比如搜索中台可以为各个垂直搜索领域提供服务，业务可以基于搜索的通用能力和定制能力发展自己领域的搜索业务。

在架构分层上，业务是构建在平台之上的；而业务和中台则是共生关系，他们属于同一层，业务需要在中台定制自己业务的特殊逻辑，整个通路本质上都属于这个业务。

举例说明一下，平台好比一个烤箱 + 一批调制好的食物（比如面包，蛋糕，这是平台对外的服务能力），使用方只需要按照约定的接口或者平台操作，就可以获得到烤制好的食物。中台好比一个烤箱 + 一批基本调制好的食物，但是允许业务方进行定制（比如烤箱的火力大小，食物中添加更多食材），甚至可以发明一些黑暗料理。中台以可定制的形式，给业务提供了灵活性，而中台自身能力的集合，则节省了业务的创新成本。

## 2. 中台是否需要微服务化

技术中台支持的业务非常多，变更频率也非常高，这样就导致了如果不采用微服务化，那么各个业务不但开发阶段就要刻意的避免影响其他业务，要做非常全的回归测试，上线的时候也要排队，这个是更要命的。很多业务关键模块每天都有 1 次以上的变更，那这样上线的协调、上线过程中的 check 都是一个复杂的工程了。此外还有容量管理，因为不同业务的流量来源不同、资源开销不同，导致容量管理会非常复杂，异常时候跨机房流量调度，稍有不慎可能会影响其他业务。其他潜在问题还有很多。


所以，微服务化是做好中台的必然选择。可以说，微服务化是我们中台技术体系的基础，我们的容量管理、资源审计、全流程的无人值守设计，都依赖各个

核心模块的微服务化。此外，基于这些微服务，我们也低成本实现了这些业务的 SET 化部署，可以做到灵活的流量调度和容灾建设。

我们在离线对业务开放的定制模块，还实现了 serverless，serverless 对业务带来的是开发成本的极大节省。目前离线的业务开发者，只需要写内容加工处理的函数即可；其他的复杂工作，都交由中台来完成。

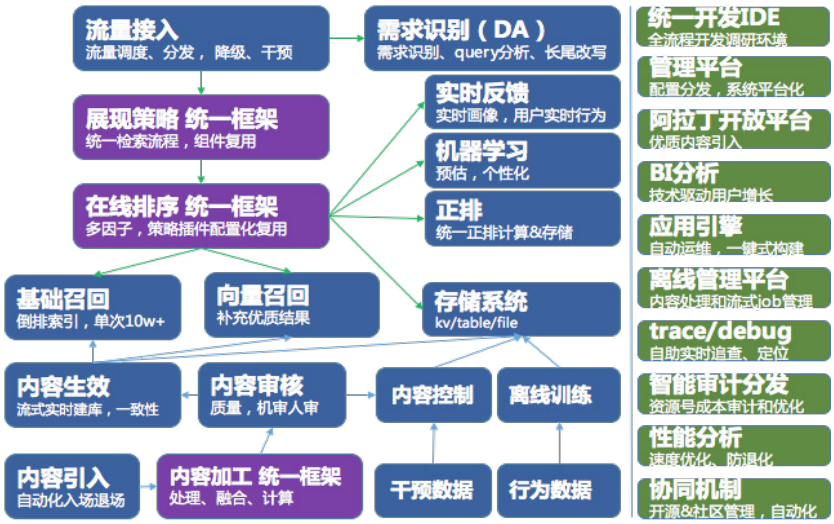
## 3. 百度垂直搜索为什么要做中台，在做中台之前如果要支持新业务是什么样子的?

我们的中台，实际是从平台建设的基础上发展起来的。之前我们的产品形态相对确定，因此可以提供完善的平台化能力，来满足这些相对确定的需求；如果有新的需求，那么就直接在系统核心模块增加功能、扩展能力即可。

现在，我们不但深耕了更多的搜索结果，满足用户更完善的获取信息甚至服务的需求，也扩展了更多的流量入口，包括小程序、独立站等，随着产品思路的调整，产品形态日趋复杂，之前的平台化的思路就行不通了。原有系统核心模块无法满足业务多样性的扩展需求，一些扩展成本非常高、上线和维护的风险特别大，使得之前的系统优势转换为了制约当前业务发展的缺陷。而转为中台战略，使得我们同时高效高质量的支持更多业务，成为可能。

## 4. 中台建设的思路是什么，对个人和团队的能力要求是否有变化?

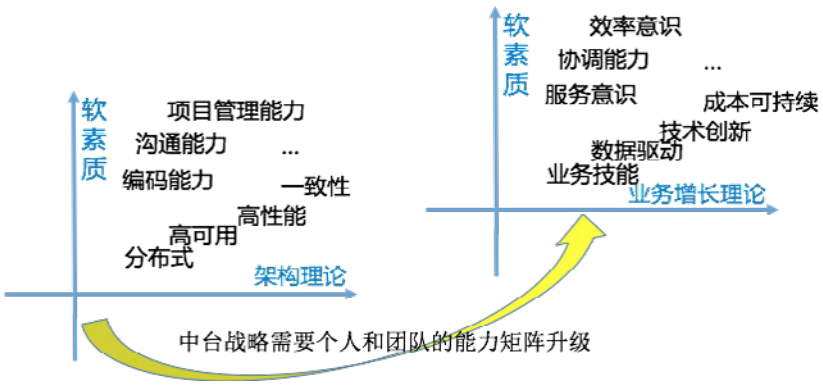
我们中台的技术思路是：提供完备的通用能力、定制能力，持续完善领域技术沉淀能力。业务视角，中台提供了灵活的可定制业务框架，使得业务可以聚焦业务特有逻辑的开发；中台还提供了可以复用的业务组件，使得业务可以通过配置化来复用优秀的中台能力；中台还提供了完备的文档建设、视频教程，支持业务快速上手、快速迭代，同时还提供了面向全流程开发效能提升的完整自动化工具链。我们系统架构图如下。




业务方最直接看到的是展现策略统一框架、在线排序统一框架、内容加工统一框架，以及可以通

过框架配置化使用的丰富的业务组件；这些业务组件沉淀了大量的领域知识和经验，可以极大加速业务的创新和迭代。

在架构升级的背后，包含了个人和团队的能力矩阵的升级。我们团队之前从事的偏基础架构的工作，关注的是架构的可用性、性能、成本等核心指标。团队转型到业务驱动后，使得团队除了继续承担架构的核心职责外，还要持续为业务交付可持续增长的技术方案，这对团队提出了更要的要求。




## 5. 中台是如何推动的，以及中台和前台是怎么协作的

我们充分考虑的业务落地 / 升级的成本。通用机制，我们尽量考虑的是无缝升级、透明升级。

当然，一些大的改进，部分功能是不可能做到完全的向前兼容的（或者说不像拖着沉重的历史包袱前行），所以不可避免的会出现需要业务改动代码上线的情况，在这种情况下，我们团队的业务专家就会发挥作用了，不单单是在设计之初就会考虑到落地成本、进行针对性的设计，落地时也会参与到业务的具体升级方案的设计和人力预估对于一些关键升级，我们的组件有时候会提供更高抽象的 lite 版本，lite 版本损失了灵活性但是却提供了更强大的集成能力，从而使得业务可以更低成本升级。

简单来说，我们以低成本落地通用能力为目标，统合综效，与业务高度协同，实现升级的完整性。因为升级本身会带来业务的收益，在保证效率的前提下，前台还是非常配合的，中台技术落地普遍得到了业务的支持。

此外，我们在完备了中台能力的基础上，正在推进全系统的内部开源，希望构建高效协同的开放生态，吸引更多的开发者，追求创造性协作，共建中台技术体系。

## 6. 是否有实际案例说明中台能提高前台研发效率?

首先给出我们已经做到的：月级别孵化新的垂直搜索产品，完成线上小流量；用不超过一个季度的精雕细琢，完成产品的全流量上线。这个时间包含了业务从学习我们系统，进行开发，然后部署整套系统到生产环境，并完成线上效果验证的全流程；对于熟悉我们系统的同学，这个周期会更短。

我们怎么做到的?

除了我们建设的完备的技术能力（定制能力、通用能力、创新能力），还有在我们深刻理解业务的基础上，在这些技术能力之上的再次抽象：我们提供了 3 层的能力输出。

L1：接口形式获取垂直搜索的结构化数据，以方便业务在此基础的二次加工、新场景的快速试验

L2：配置化能力，实现业务 0 code 即可上线。我们提供了完整的搜索通路能力。这对于业务的迭代效率非常关键；此外 PM 可以通过这个通路验证想法，从而辅助产品决策。

L3：从离线到在线的可定制能力，以及完备的架构组件和策略插件。

这样不同业务的发展阶段，不同的角色都可以获得所需，这是我们能够获得高效的一个根本的技术保障。


我们为了追求效率的卓越，还专门成立了研发效率团队，聚焦全流程、全周期的效率提升，包括我们建设了在线的 debug 系统、离线的 trace 系统，都是帮助业务能够自助和高效的追 case、查问题；我们还建设了统一集成开发环境，实现开发环境和仿真环境的自动打通，实现低成本的产品效果验证，这种例子还有很多。最终我们为业务提供了卓越的研发效率。

我们提供的不仅仅是简单的生产工具和开发方式的升级，而是逐渐完善的技术体系，我们为业务带来的是生产力的升级，为业务带来了质变的研发效能提升。

## 7. 中台未来有何演进计划?

未来依然围绕技术中台的核心目标，进行不断的技术创新，来满足业务需求，促进业务可持续增长。一个是作为技术中台本身，需要密切关注业界技术发展的趋势，实现技术先行，通过技术发展来实现业务增长。一个从各个业务沉淀行业经验，来实现技术驱动的产品创新，来引领技术的发展。


总之，我们会通过持续的技术演进来不断完善中台能力集，以应对未来场景的千变万化，促进业务的可叠加创新，实现技术驱动增长。

## 嘉宾介绍

张安站，百度垂直行业 & 阿拉丁架构技术负责人。2014 年加入百度后一直在大搜索从事架构和平台化相关的研发工作。目前主要负责大搜索垂直行业和阿拉丁的整体架构和研发中台，为众多业务提供了可定制和可复用的垂直搜索产品研发的中台技术体系，可在月级别孵化多场景的垂直搜索创新产品，善用技术思维解决效能问题，通过技术实现各个业务的创新叠加，促进各个创新业务的快速可持续发展。

# 唯品会自研微服务框架 OSP，解决拆分、扩容难题

作者 杨钦民

马尔文·康威 1967 年提出康威定律：“设计系统的架构受制于产生这些设计的组织的沟通结构。”

根据康威定律，当互联网公司业务和团队发展到一定规模，微服务架构是一种必然的演化趋势。近几年，随着电商业务的快速发展，唯品会逐渐实施了微服务架构，面对如此大规模的电商业务，如何进行微服务框架体系的建设；面对电商大促，如何快速大规模扩容；面对如此复杂的电商业务，如何更好的拆分微服务，都是要解决的一系列难题。

唯品会业务架构架构师杨钦民老师将在 ArchSummit 全球架构师峰会分享唯品会微服务基础中台和最佳实践的演讲，我们提前邀请杨老师对分享内容进行预热解读，希望对你 / 团队有帮助。

## 唯品会微服务基础中台架构设计思路

围绕微服务，唯品会自主研发了微服务框架以及一系列配套的系统。

- OSP（开放服务平台）微服务框架，提供高性能、高可扩展的远程调用机制，实现了契约化多语言服务接口，同时提供了强大的服务化治理能力，可以实现负载均衡、路由选择以及自我保护等。
- Service Center 统一的服务治理中心，对基础服务化项目提供的服务进行治理，将所有服务化项目的配置集中在一起，实现一处配置、多处运行的目标。

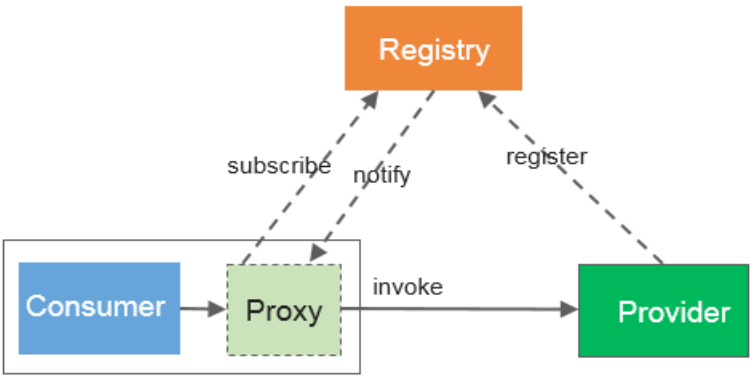
- Mercury 全链路跟踪监控平台，实现了全链路调用链跟踪、指标统计、监控告警等，通过 Mercury，应用管理人员 / 架构师等可以全方位把握应用整体拓扑结构、定位全网应用瓶颈。应用开发人员可以定位线上服务性能瓶颈、持续优化代码和 SQL、帮助快速解决线上问题，IT 运维 / 监控人员可以快速故障告警和进行问题定位、把握应用性能和容联评估、提供可追溯的性能数据。
- Janus 服务网关，为业务服务提供统一对外的、高性能的 HTTP 网关，针对外网支持 HTTPS、HTTP2、HTTP、自定义协议等，针对内网可以自动适配到 OSP 协议。
- Salus 服务安全管理平台，面向 OSP 和 RESTful 形式的服务，提供服务安全管理（认证、鉴权、防篡改）的手段。
- 基础中间件，CfgCenter 应用配置中心实现应用配置管理，Saturn 分布式任务调度平台具备高可用以及分片并发处理能力等，Asgard 一站式存储服务平台可以实现统一管理、统一监控存储服务，VMS 消息系统具备组内广播、消息回溯、消息延时、灰度消息等。

## 唯品会开发微服务框架 OSP，和 Service Mesh 有哪些异同?

唯品会在设计 OSP 微服务框架之初，就已经单独抽出了代理层 Proxy，类似 Service Mesh 的 Sidecar。客户端与微服务框架代理层 Proxy 部署在同一机器的不同进程，各自独立部署，服务治理逻辑从客户端业务逻辑中解耦出来。

和 Service Mesh 相比，OSP 所具备的优势包括：

- 加强运维管理，运维人员可以单独针对 Proxy 进行独立升级、维护，极大加强运维管理能力。
- 框架可以持续演进，Proxy 作为一个独立的代理层，与服务隔离，并且独立部署和运维，每次框架发布新版本时，无需业务研发部门介入，运维就能独立进行升级和部署，因此服务框架可以持续进行演进。
- 支持多语言，Proxy 采用自己的开发语言进行开发，独立演进，而每个服务均可以采用合适的开发语言，二者互不影响。




## 唯品会拆分微服务过程中遇到的难题

在拆分复杂电商业务系统的过程中，遇到的难题如下所述。

- 如何按照组织架构划分以及搭建比较清晰的系统架构，如同康威定律所说：组织架构等同于系统架构，一个好的系统架构需要匹配组织架构，同时组织架构在很大程度上会影响系统架构的走向。
- 如何界定服务边界，将复杂电商业务拆分成不同的服务时，首先面临的是如何可以清晰的界定不同服务的边界，减少服务间的耦合，其次面临的是服务拆分的粒度，粒度过粗可能会造成服务间的耦合，粒度过细可能会拆分出过多的服务。
- 如何进行服务聚合，服务拆分之后，当面临复杂业务时，需要考虑多个服务的聚合，是由一个团队进行聚合分别提供各方，还是由各业务方分别聚合。

## 在微服务框架体系建设实践中，选择开源还是自研?

在建设微服务框架体系建设过程中，针对不同业务体量、不同技术储备的公司，我们需要思考以下几个关键点：

- 是选择开源微服务框架，还是选择自研微服务框架？如果一些大公司业务体量非常大，技术储备非常多，发展到一定的阶段，可以根据公司自身情况，考虑自研发微服务框架体系。而中小型初创公司，由于业务体量不是很大，同时技术储备也比较少，技术人员的技术实力也不够深厚，建议选择各种开源微服务框架，构建自己公司的微服务框架体系。
- 是否选择自构建 Kubernetes 集群。同样，对于有自建服务器机房且业务体量庞大的公司，选择自建 Kubernetes 集群是再好不过了。而针对中小型初创公司，建议选择云服务商，可以更快构建 Kubernetes 集群。

- 是否选择 Service Mesh 框架。Service Mesh 是近两年的热点，是否选择跟风演进到 Service Mesh 框架，也是一个考量。大公司甚至大集团，业务线非常多，技术体系也比较丰富，一般会有多种开发语言并存，同时大公司的技术实力也非常雄厚，此时建议演进到 Service Mesh 框架，国内已经有多家有实力的大公司在积极自研发 Service Mesh 框架。而针对中小公司、初创公司，需要根据自身客观情况考虑，一般都是只有一种开发语言，所以针对此种情况，建议可以选择 Spring Cloud 框架、Dubbo 框架等。

## 2019 年 Kubernetes 在应用开发和运维上有非常重要的能力透出，在唯品会有哪些深入应用?

唯品会基于 Kubernetes+Docker 技术框架研发了 Noah 云平台，涵盖应用的开发、交付、运维和运营全生命周期，整个平台包括主机层、容器层、云平台层、镜像管理、CI/CD，支持灰度分批发布、容器 Auto Scaling、集群节点管理、多集群时间监控等。

- 开发阶段提供 CI 镜像流水线，本地开发 DockerVM，功能测试环境、联调测试环境给开发和 QA 完成镜像的发布；
- 生产阶段提供容器发布、容器调度、容器服务注册发现、容器网络互通、集群管理等功能完成运维工作；
- 提供容器的运行时日志、监控、告警等功能做业务监控。

除此之外，杨钦民老师将在深圳大中华喜来登酒店分享半天的微服务架构培训，分享电商平台微服务架构演进实战，帮助听众了解大规模容器云实施实践，以及微服务框架体系建设经验。

# 字字珠玑

# 高级技术专家带你了解阿里的开发流程规范

作者 孔凡勇（云狄）

此前，阿里高级技术专家孔凡勇（云狄）老师撰写了在 Alibaba 成为优秀的技术主管，需要在“开发规范、开发流程、技术规划与管理”方面有自己的深入思考文章。受广大读者的需求，我们邀请云狄老师解读了部分读者提出的问题，帮助大家深入了解阿里的开发流程和规范化实践过程。以下是 Q&A 形式的文字整理。更多细节关注 7 月 12 日深圳 ArchSummit 全球架构师峰会上云狄老师的演讲

InfoQ：请先介绍一下你现在所做的主要工作内容?

我目前一多半时间用在开发、任务分解分配、开发实践、技术架构评审、代码审核和风险识别上，剩余的时间则花在为保障系统按时交付所需的各种计划、协作、沟通、管理上。

InfoQ：对于新业务和新团队，在整体技术规范和标准上存在很大的差异，需要标准化统一。是因为不一样而统一，还是因为导致业务损失而统一？或者是从长远角度考虑而统一?

像 Google、Facebook 都有自己制定的一些开发规范，阿里去年也开源了内部的一套开发规约，其实这些标准的开发规范都是长期积累的一些经验值。

制定这些开发规范，主要基于以下几点考虑：

- 降低故障率：尤其是杜绝重复故障率，防患于未然。如：集合的 SubList 使用方式、SimpleDateFormat 的线程安全等问题。
- 提升协作效率：现在很多公司都提倡中台化与标准化，团队内部协作、跨团队合作都需要保证高效率，高效协作的一个前提是规范统一标准化。
- 工匠精神：工程师对于代码，一定要“精益求精”，不论从性能，还是简洁优雅，都要具备“精益求精”的工匠精神，认真打磨自己的作品。

InfoQ：引领团队走向统一标准化开发道路，正如您此前提到的，命名规范、统一模版、API 规范等等，这些措施给当时的业务带来了哪些明显的收益?

最重要的一点是提升开发效率，提高代码质量。如果个人的代码模板和团队不一样，会增加代码的 merge 成本；如果你的 API 不规范，会额外带来开发与沟通成本；如果你的命名不规范，有同样会带来沟通成本。

应用命名规范、模块的划分、目录（包）的命名，我觉得非常重要，如果做的足够好，别人导入项目后可能只需要 10 分钟就可以大概了解系统结构。

也许有人会问，开发规范如何从 0 到 1 开始做，而且还不能影响开发进度？制定规范，更多是改变一些传统做法，去适应新的变化。现实情况下肯定会影响进度，开发规范不一样，流程可能也变化，具体影响范围需要根据具体项目来评估，不存在标准答案。

**InfoQ：你们团队在开发流程上是如何做需求管理的？需求太多怎么协调？**

如何做好需求管理，对于技术 TL 来说是非常重要的。对于产品经理提过的需求，首先要有一个初步的过滤，到底是真实的客户需求还是伪需求。其次按照产品需求的重要程度和时间迫切度可以分为四种类型：紧急重要型、紧急非重要型、重要非紧急型和非紧急非重要型。

对于有效的需求管理，首先是关注重要紧急型需求，其次才是不紧急重要型需求、紧急非重要型需求、非紧急非重要型需求。在判断其重要性和迫切性上有个基本的概念：先解决负面灾难，再追求正面功能，“查缺补漏优先，锦上添花其次”的原则。

很多时候需求的变更或增加是因为我们面临太多选择和想要的太多，没有适当的控制自己的欲望，并以自己的喜好来决定需求，这些因素很容易导致产品没有明确的方向、团队成员疲于奔命，但是却没有实际的成果。所以技术 TL 一定要能够评估出重新审视产品和筛选需求的优先级，识别每一个需求的必要性、重要性和实现成本。通过深思熟虑给团队明确方向并专注，聚焦资源的支配，确保团队的精力都聚焦在产品的核心需求上。

**InfoQ：架构评审的时候，是不是要重点考虑未来的扩展方向？还是仅仅满足当前的需求即可？**

可扩展性，是将软件未来的发展过程也考虑在内，而进行的一种设计选择，可看作是一种面向未来的柔性设计（Supple Design）。

系统架构可扩展性设计需要把控好当前需求架构与未来不确定性架构，如果权衡不好会有两种情况，设计不足与过度设计，对于支撑业务的发展会有一定的损害。

设计不足，则意味着系统复用性扩展性和灵活性都很差，系统僵化，不能应对将来的需求变化，或者将来修改和维护的代价和成本会很高，这当然是设计错误。

过度设计，则意味着为了实现这个设计要付出的额外代价，例如成本上升，缺陷可能性加大，提升维护成本，甚至降低系统性能。而可维护性和系统的高性能都是系统的隐性需求，这些需求没实现好，当然也是设计错误。

无论是面向当前需求还是未来变化需求，我们都需要遵循一个简单之美的原则。最简单的才是最好的。大巧若拙，大道至简，有时候越简单的反而越难实现，而且越接近真理。

我极力推荐大家多了解下微内核插件架构，比如我们经常使用的 jQuery、Eclipse 都采用的微内核插件架构，使用微内核架构，用户代码就可以通过插件的形式，集成到已有的核心系统中，具有高度的可扩展性。

大家都期望当前维护的系统具有高度可扩展性，以减少眼前的工作负担。但是一方面这类系统实际上需要很强的预见性才能设计出来，难免出现偏差。另一方面，高可扩展性也未必没有弊端，是否具有优势也与当前的项目场景有关。这个需要架构综合去做权衡，有时候维护一个高可扩展性的软件系统，可能反而不如简单系统外加合适的变更方案更有效。

**InfoQ：在内部推动规范化过程中遇到的挑战多吗？又是怎样克服的？**

首先来讲，规范必须是大家的共识，每一条规范都需要说清楚原因，并且给大家讲清楚，让大家知其然也知其所以然。知道为什么这样子做比较好之后，执行的意愿会大大加强。而不会有太强的抵触情绪。

推动一些规范化的事情主要包括技术规范和一些流程规范。技术规范推进整体来说还算比较顺利，作为工程师对于的技术追求大家基本都是一致的，比如 CodeReview 规范、架构设计规范、发布计划规范等。

对于跨团队的一些流程规范，这里面是有些挑战的。很长一段时间以来，也许每个人都习惯了自己的工作方式，处于一个舒适区。今天突然要换一种协作方式，或多或少有些抵触情绪。这里我更多是拿一些数据说话，这样能够去说服别人，比如 PRD 一些输出规范，必须要有一些上

线后的期望值数据，无论营收还是 UV、PV，最后落实到客户价值和商业价值，如果达不到期望值，你给协作团队的信任值会降低，后面你的需求优先级也会被降低。

InfoQ：平时是如何灌输技术规划思想的？又是如何培养团队里其他人实践技术规范操作？

技术团队也需要紧密衔接业务团队做一些技术规划，一方面我对一些创新业务要用到的技术做一些储备和研究。另外随着已有的业务不断发展，系统架构的腐蚀是避免不了的，我们需要制定一些技术规划，来保障业务的健康发展，思考如何对现有系统架构、性能进行优化，作为技术人我们也要有自己的技术 KPI。

一言以蔽之，技术规划是融合多因素的发展愿景，基于对未来整体性、长期性、基本性问题的思考，制定设计全面长远的发展计划和行动方案。

如何去做技术规划，需要结合业务产品和技术，思考三个问题：Why、How、What。

Why：为什么要做这方面的技术规划，目前的性能有问题还是架构存在问题？例如财年的订单量预估翻番，相应的订单查询 RT 有所保障。

How：如何去实施对现有架构、性能等方面的优化，结合业务现状以及未来规划出发，整理潜在的技术方案、架构、技术栈。

在技术规划落地前，我们首先要有明确的目标，比如查询 RT 降低 50%，采取缓存、分库分表方案。

系统架构存在耦合的情况，目标需要进行解耦，以及面对未来不定性业务发展方向，我们如何保障系统的可扩展性，支持业务的快速发展，采取一些微内核、微服务架构思想快速支持新业务发展。

What：技术规划是为了更好的适应明天的变化，目的是变的更好。技术规划在执行过程中，将任务计划拆解成阶段性里程碑，要把控好技术风险和管理风险，需要进行定期检查，定期总结，有问题要及时去调整。

InfoQ：你们团队的工程师文化是怎样的？或者说阿里的工程师文化有没有什么特色？

一提到工程师文化，大家首先想到的应该是 Google、Facebook 之类的公司，在《重新定义公司：谷歌是如何运营的》这本书里对工程师文化有更详细的解读。这里简单谈下我对工程师文化的理解。

简单来说工程师文化就是一切以解决问题为导向的工作文化，工程师有一定的自由度和技术创新。层级尽量扁平化，第一线要具有极大的自由度，权责要向下转移。

阿里以及我们团队的的一些工程师文化主要如下几点：

1. 建立共享代码所有权
2. 建立合理的软件抽象
3. 工作中重复劳动力，抽象为工具自动化
4. 注重代码审查，编写高质量代码
5. 保持一个开放、尊重的工作环境
6. 工程师的决策权
7. 建立学习与分享的组织
8. 招聘最优秀的人

第一要建立共享代码的所有权，阿里包括中间件等很多代码都是可以共享的。

第二是要建立合理的软件抽象。比如说像 Dubbo、RocketMQ、RPC 等中间件，都是基于业务场景抽象出来的中间件，这就是对软件的一种抽象能力。

第三是工作当中的重复劳动力抽象为工具的自动化。尤其像 Google，Facebook 是非常崇拜这种文化。这种文化在阿里内部也是极度盛行的。


如果你的日常工作有好多重复性的工作，那你一定可以开发出一种自动化工具来解决重复性工作。

第四是注重自动代码审查，编写高质量代码。去年阿里开源了一些整体开发规约，包括开源了 P3C，Eclipse 等插件，目的都是为了能够保障编写出高质量的代码。

第五是要保持一个开放，互相尊重的工作环境。工程师要有足够的决策权，比如在需求评审时，技术人员对于产品需求是有一票否决权的，当需求规范不够标准的时候，技术人有权利拒绝你的需求。大家在这种互相尊重的工作环境中才能更好的行使权力。

第六是建立学习与分享型的组织。我的团队基本上每周都会举行一些技术分享会，包括读书分享会，轮流做一些基础的分享，引导大家共同学习。

最后是要招聘最优秀的人。几乎所有人都希望和比自己优秀的人合作。像 Google 也贯彻这类招聘理念。

**InfoQ：为了让代码更好维护，您的团队会经常重构代码吗?**

这是肯定的。每一个财年每个季度都会做一些技术上的规划。比如目前的系统存在耦合性问题，要对代码模块做优化和重构。比如说在代码质量的圈复杂度相对比较高，可能会进行抽象、重构，这是经常要做的一件事情，它对于一个系统未来的可扩展性非常重要，我认为多投入一些时间是非常值得的。

**InfoQ：开发团队的绩效 KPI 在阿里是怎么评估。是否有一些可量化的指标呢?**

技术人员的绩效不能和运营、销售人员类比来做可量化的指标衡量。不可能根据提交代码的行数来评价一个人的工作量。也不可能根据代码 bug 数来界定到底是产品方还是业务方导致的。

我更看重一个人的整体工作效率和质量，在横向比较中这些是能够可视化的，在保证效率的前提下，能够保证质量，甚至说在团队中能起到一些正面的积极的影响。另外我比较关注他是否能遵守团队制定的一些标准规范，包括整个集团的开发规约。

关于整个系统的稳定性非常重要的一点是，必须能保质保量不出现生产环境问题。

其次是 KPI 里会体现出关于系统设计，分析命题，解决问题的能力，我比较关注。最后一点是对这个团队的贡献和影响力，比如说在团队能够推广一些技术，包括一些相关的规范，引入一些核心的技术，能够提高团队的生产力。

# 阿里大文娱实践分享：业务架构如何影响中台建设？

作者 赵钰莹


阿里大文娱的特点是业务基本都是收购而来，起初倡导各自独立奔跑，最终却发现整体呈现“大而不强”的状态，在这种情况下实践中台，阿里大文娱面临着各种挑战，在做和不做以及怎么做上进行了深入思考。

近半年，中台相关话题热度颇高，随之而来的是各种角度的解读和实践分享。在这些内容中，阿里巴巴的中台战略最易被提及。2015 年 12 月，阿里巴巴最先在国内提出“大中台、小前台”的理念，并以此进行组织升级，旨在建设“敏捷的前端 + 强大的中台”。从去年到今年，腾讯、百度、京东接连开始组织架构调整，为建设中台做准备。

在这种热潮中，企业对于要不要做中台、如何做中台以及中台的价值不免产生疑惑。除了中台，企业也可以通过很多方式实现数据融合，这之中的差别体现在哪？如何有效抽离出不同业务中具备共性和通用的模块？作为阿里大文娱事业群基础平台负责人，纬洲对于目前要实施的文娱中台战略做过哪些思考？带着上述问题，InfoQ 对阿里大文娱事业群基础平台负责人纬洲进行了独家采访，谈谈阿里大文娱对于中台的看法。

## 阿里巴巴中台战略

阿里大文娱中台复用了部分阿里巴巴中台的基础能力，因此在了解阿里大文娱中台之前有必要简单看看阿里巴巴中台的演进历史。

对此了解的读者想必知道，这一战略最初的灵感来源于芬兰一家名为 Supercell 的游戏公司（2016 年 6 月，该公司被腾讯以 86 亿美元收购），这家公司当时仅有 300 名员工，却可以接连推出爆款游戏，比如《部落冲突（Clash of Clans）》、《皇室战争（Clash Royale）》等。

据了解，Supercell 内部有一个强大的技术平台，将游戏开发过程中公共通用的游戏素材和算法进行整合，为小团队提供工具和框架，支持众多小团队独立进行游戏研发，而不用担心基础却又至关重要的技术支撑问题。恰恰是这家小公司，开创了中台的早期“玩法”，而这种想法也启发了阿里巴巴。

2016 年，时任阿里业务平台事业部、淘宝基础平台技术部负责人的玄难详细介绍了阿里巴巴业务中台的建设思路，整体来看分为四个阶段：

- 单一业务系统阶段：一个业务系统只需要几台机器支撑，在业务发展过程中逐渐改造成 Java 体系。
- 分布式业务系统阶段：此时的淘宝团队人员上千，团队效率和系统稳定性都面临挑战，主要业务是淘宝集市和淘宝商城。阿里巴巴将单一系统拆分为多个高内聚，低耦合的中心化系统，将上千人的团队拆分为业务相对集中的小团队，独立设计、独立承接需求、独立发布等。用户中心、商品中心、交易中心均在这一阶段出现。
- 业务中心平台化阶段：2011 年，阿里巴巴在战略上进行调整，将淘宝拆分为淘宝、天猫和一淘三个独立的事业部，此时事业部业务系统和规则上的矛盾开始暴露。为了解决这一问题，阿里巴巴将基础能力与每个业务方的特性业务进行拆分，将业务之间的逻辑进行隔离。总结来看，平台化的关键点是业务抽象建模和系统架构的开放性，其中业务抽象解决共性的 80% 问题，系统架构开放性解决 20% 的个性化问题。
- 业务中台化阶段：平台化只能解决领域内部的问题，但是每一个业务的执行都是跨领域的。随着生态复杂度、业务复杂度、系统复杂度的升级，阿里巴巴又开始遇到新的问题，比如信息获取成本高、互联互通成本高、服务具有不确定性、低水平重复建设等，电商业务中台开始进入研发阶段。

回看整个过程，纬洲表示，随着业务复杂性越来越大，单一后台明显不能满足阿里巴巴前端业务的快速变化，除非将系统设计得特别复杂，这些业务之间又存在大量共性需求，阿里云的出现让基础设施和后端网络架构变得相对独立。在这种情况下，需要通过中台进行统一技术能力整合。阿里巴巴的中台也为阿里大文娱提供了很多基础能力，比如数据、中间件等，但阿里大文娱的业务存在很大不同，因此没有办法完全复制集团中台。

## 阿里大文娱中台战略

阿里大文娱实现中台的前提是用户、业务与技术栈的统一。

正如前文所言，阿里大文娱的业务情况与集团存在很大不同，最大的特点是阿里大文娱的业务，包括大麦、UC、优酷等都是收购而来，后台系统存在很大不同，短时间内难以统一，阿里大文娱的中台战略如何实施呢？

### 从独立发展到统一化

收购之初，阿里大文娱内部的各项业务还处于独立发展的状态，然而在这种模式下，阿里大文娱整体呈现“大而不强”的状态。2016 年，阿里大文娱正式成立，两年多的时间里，已经形成从内容生产（阿里文学、UC）到投资发行（阿里影业）、票务流通（淘票票、大麦）、视频平台（优酷）等，涵盖的领域非常全面，每一项业务量和用户量均不在少数，但是还没有出现明显的旗舰业务。

在这种情况下，阿里巴巴大文娱开始思考如何打造阿里大文娱的旗舰业务以及新用户的获取渠道，通过部署中台战略与集团进行统一，很可能将集团的老用户转化为阿里大文娱的新用户。此外，阿里大文娱的各项业务内部存在很多共性的地方，比如交易、资讯、娱乐等模块，如果将这些通用能力抽离出来形成中台，那么阿里大文娱就可以实现全局数据打通，这对于利用数据分析驱动业务增长具备巨大价值。

具体来说，以一位明星为例，其身份可能是演员、歌手甚至是作家，与其姓名相关的词条可能有演唱会（通过大麦网卖票）、电影（通过淘票票卖票）、电视剧（优酷）、文学作品（书旗小说），如果将整个阿里大文娱的数据打通，很容易计算出该明星相关作品对整个平台产生的价值，并可以对未来的相关安排（是否上线该明星主演的其他电视剧等）进行智能分析以确保价值最大化。

经过多轮思考，阿里大文娱决定部署中台战略。在实践阶段，其第一大难点是各业务系统高度不统一。以优酷为例，纬洲表示，原来优酷 CDN 部分的 Linux kernel、Web 服务都是团队自己更改或重写的，当然这一部分在被收购之后并入了阿里云。在移动内容管理（Mobile CMS）层面， 2010 年的时候，因为当时业务的快速发展，优酷当时的技术团队基于 Ruby 进行开发，因为其组件相当丰富，所以仅用时七天便完成了整个过程。实际上，除游戏开发外，很少有人使用 Ruby 搭建服务器，这是因为 Ruby 本身的一些问题让其并不适合服务器端搭建，Ruby 的缺点其版本的向下兼容性较差，甚至有时完全不兼容。后来整个移动服务后端，选用了 facebook 开源的 Tornado 框架，使用 Python 和 C 语言进行重写。当然，这中间还经历了土豆的技术架构融合。

然而，整个阿里系的技术栈都是基于 Java 实现，这让优酷在被收购之后正式成为阿里大文娱的一员的时候，又用了两年左右的时间进行整个技术栈的重构，目前其上所有系统全部都用 Java 重写过。在文娱业务整合的过程中，因为淘票票来源于淘宝电影和支付宝电影业务，技术栈和集团相同外，大麦、UC、以从 UC 分拆的书旗小说为基础的阿里文学，还包括影业的云智影院管理业务等都经历了这一阶段。

### 以核心业务为抓手开始文娱中台的建设

在用户、业务与技术栈达成统一后，阿里文娱中台才有建设的可能。系统建设进入第二阶段文娱中台的开局阶段，文娱中台服务的业务众多，各个业务又有其不同的特点，大麦、淘票票主要以 C 端

交易业务为主，一个是演出票业务，一个是电影票业务，文学以付费阅读为主，UC 是以信息服务为主，音乐以会员业务为主，优酷业务以视频播放、广告和会员业务为主，整个文娱还有众多的 B 端业务，如：大麦的麦座场馆业务，影业的云智影院管理业务等。这么多业务除了有各自业务的特性外，还有很多共性的业务，如：交易中台等等。如果水平推进，一方面整个产品技术团队的架构和人员都会变得异常庞大和复杂。反过来，对业务发展的支撑，很难有明确的抓手或者主业。文娱中台的架构设计和实施会变得异常复杂，且实施周期很长。所以必须找到简单易行，分阶段改造的方法。

纬洲介绍，该阶段的首要任务是“核心业务抓手”，在阿里文娱的场景下，优酷和 UC 两个业务是核心业务，从 UC 的业务特性上看，未来主要是信息分发和渠道。无论是从现有的用户规模和未来的 5G 发展两个角度来看，文娱中台的建设需要首先服务核心业务发展，而阿里文娱的核心业务中，能成为文娱中台建设的抓手的首先是优酷。“核心业务抓手”可以理解为优先支持优酷的业务发展，目前阿里大文娱的中台团队也以优酷产品技术为主。其他业务先可以按照中台的规划独立奔跑，等文娱中台的改造对优酷的支持达到无缝的程度，就可以以此为基础，建设整个大文娱统一的中台，比如：全文娱统一的媒资，全文娱统一的交易等等。

仔细琢磨，这个概念其实也很好理解。在一些传统企业，如果想要实施中台，势必需要对内部组织架构进行调整，不仅仅需要引进相关研发人才，可能需要对各垂直业务团队的人员进行调整，因为在统一中台的情况下，有些功能其实中台团队可以统一实现，在这种情况下，很多企业实践中台往往引起剧烈震动。相比之下，互联网公司的灵活度和敏捷性更高，但实施中台依旧没有那么顺利。

纬洲提及，如果无法让核心业务迅速通过中台获得价值提升，很难保证整个战略的顺利推进，毕竟整个公司的业务发展节奏很快，需要支持团队具备敏捷响应能力，这种能力各垂直业务团队是可以提供的，如果在实践中台后响应速度变慢，很可能影响最终业务发展。因此，依托核心业务抓手，支撑核心业务发展是很重要的一环。

随着通信（5G）以及人工智能等技术的发展，优酷作为目前的核心业务自然对未来的发展方向做过深入思考。纬洲透露，类似视频这种消耗带宽，或者可以给用户带来更强体感的应用一定会在 5G 时代获得很大突破。对此，优酷在技术层面进行了很多调整，比如通过技术手段对老片进行高清和超高清的修复；对不同内容提供不同的介质去实现高清及超高清的体验；对 VR、AR、包括 MR，计算机视觉等技术的应用进行布局等。

## 关键问题解读

当然，中台的实践经历会因为每家公司的业务不同而有所差异，任何一种技术手段其实都很难适合所有企业，但纬洲还是结合整个阿里大文娱的实践过程对企业实施中台分享了一些切实可行的建议。

### 1、中台是否具备普适价值?

正如上文所言，没有任何一种技术适合所有企业，这本身也太过绝对，但这句话可以延伸为“中台到底（不）适合什么样的企业？”

对此，纬洲的观点是创业公司，尤其是规模很小的创业公司，其本身的业务又是某一垂直领域，其实不需要中台来提供数据统一和服务统一，因为业务很简单，本身的复杂性又不高。另外，实施中台对企业本身的信息化程度也是有要求的，信息化较高的领域，比如金融、汽车、电信运营商等可能会更需要也有能力搭建中台，并从中获得价值。

业内某专家认为，判断组织是否已经达到应用中台的阶段，可以参考一些原则，比如数据或资源是否都上系统了、是否有很多内部需求可以合并同类项、是否可以基于内部需求开发一些具有共同性的新工具、以及是否可以将基本功能作为独立的业务抽出来并对外提供服务，这四点又分别对应系统化、中心化、平台化和中台化四个阶段。企业需要判断清楚究竟处于哪个阶段，对整体设计要有清醒认知和明确规划，才能更好的将中台为己所用。

### 2、很多方式都可以实现数据融合，为什么需要中台?

综上不难看出，数据统一或者说获取全局数据是实施中台的价值之一。然而，企业实现数据融合的手段并不只中台一种，为什么一定要做中台呢?

纬洲表示，企业确实可以通过其他方式对数据进行融合，但这一定存在数据导入迁徙的过程，很难做到实时；其次，中台的价值并不局限于实现数据融合，还可以提高后续开发的敏捷性，灵活响应业务变化，这是一种创新的组织架构而不是一个简单的数据平台。

## 结束语

采访最后，纬洲表示，未来的理想状态一定是阿里大文娱中台可以支撑旗下全部业务，但这并不代表其他垂直业务团队无事可做。在如今的体量下，阿里大文娱的业务均具备自身的独特性，以文学为例，其主要的商业模式是付费阅读，而不是广告，这跟优酷的商业模式完全不同。在这种情况之下，前端产品不可能全部由中台团队实现，各垂直业务团队依旧需要保留，并不会因为实践中台完全打破原有架构。

## 嘉宾介绍

纬洲，阿里巴巴资深总监、大文娱事业群基础平台负责人。纬洲拥有 20 年互联网从业经验，曾任优酷副总裁、阿里巴巴影业集团副总裁，现任阿里大文娱中台负责人。在优酷时期，创建了优酷移动团队和 OTT 团队，优酷在其任期内占据在移动视频的领先地位长达 6 年，是一位资深移动互联网专家。

# 七牛云许式伟：我所理解的架构是什么

作者 许式伟

## 从软件工程说起

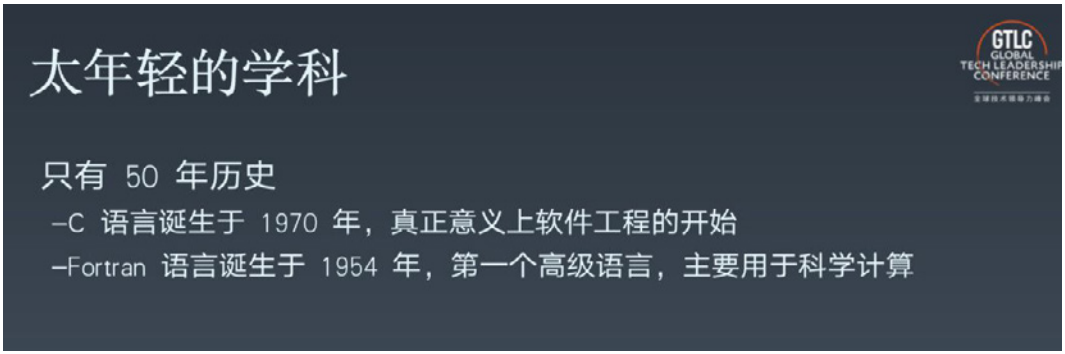
大家好！我已经很久没有做技术类的演讲了，因为我最近确实比较忙，很少会出来。为什么会突然又想谈一下架构呢？这是我个人的宿愿，我是技术出身，虽然现在比较少写技术相关的东西，但我在公司内部做了很多分享，分享课里我讲的东西与架构相关的占三分之二，基本都是和架构相关的。

所以今天借这个机会谈一谈我自己理解的架构到底是什么。

国内现在比较少真正意义上符合“架构师”这个词的定位的角色，我们的教育和工作氛围很难出真正意义上的架构师，比较凤毛麟角。我自己理解的架构师是从软件工程概念开始的，也许大家都学过软件工程，但如果我们把软件工程这门课重新看待，这门学科到底谈的是什么？是软件项目管理的方法论？

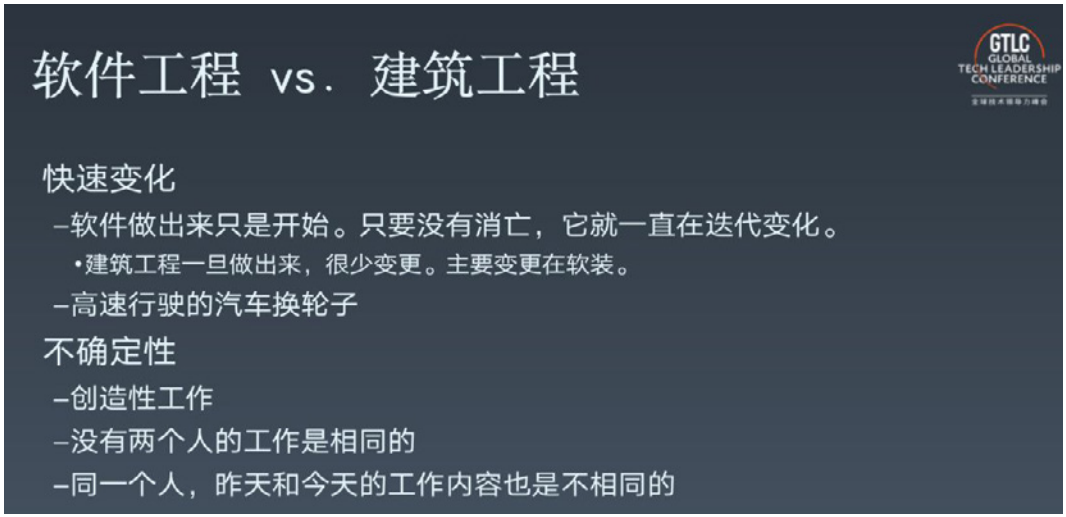
无论如何，软件工程是一门最年轻的学科，相比其他动辄跨世纪的自然科学而言，软件工程只有 50 年的历史。这门学科的实践太少了，任何一门学科的实践时间短的话，都很难沉淀出真正有创意的实践总结，因为这些经验总结总是需要很多代人共同推动来完成。

为什么只有 50 年时间呢？我们来看看 C 语言，一般意义上可能认为它是现代语言的开始。C 语言诞生于 1970 年，到现在是 49 年。再看 Fortran，它被认定为第一个高级语言，诞生于 1954 年，那




时候主要面向的领域是科学计算。Fortran 的程序代码量不大，量不大的时候谈不上工程的概念。这也是为什么软件工程这门学科很年轻，它只有 50 岁，在这样一个年轻的学科里我们对它的认知肯定还是非常肤浅的。

我在极客时间里的课程里一上来就做了软件工程和建筑工程的对比。对比可以发现二者有非常大的区别，具体在于两点：




（1）快速变化。建筑工程在完工以后就结束了，基本上很少会进行变更，除非对它进行软装上的变更，软装更像是今天的软件。但其实软件工程里，软件生产出来只是开始，而且只要软件的生命周期没有结束，变更就一直存在，很像建筑里的软装一样，而且比软装变化剧烈得多。

（2）不确定性。为什么软件工程有很大的不确定性？因为没有两个人的工作是一样的，虽然大家都在编程，但是编程的内容是不一样的。每个人昨天和今天的工作也是不一样的，没有人会写一模一样的代码，我们总是不停地写新的东西，做新的工作。这些东西是非常不同的，软件工程从事的是创造性的工作。

大家都知道创造是很难的，创造意味着会有大量的试错，因为我们没有做过。这会导致软件工程有非常大的不确定性。

以上这两点都会导致软件工程区别于传统意义上的所有工程，有非常强的管理难度。过去那么多年，工业界有非常多的工程实践，但是所有的工程实践对软件工程来说都是不适用的，因为二者有很大的不一样。

今天站在管理的视角再看软件工程，我们知道管理学谈的是确定性，我们如何去创造确定性是管理学中的追求，否则管理管什么呢？某种意义上来说管理学的目的就是要抑制不确定性，产生确定性。比如说开发的工期，时间成本是否能确定。其次，人力成本，研发成本和后期运维的成本是不是确定性的。所以软件项目的管理又期望达到确定性。这是一对矛盾。软件工程本身是快速变化的，是不确定的。但是软件工程管理又希望得到确定性，这就是软件工程管理上的矛盾。我们的目标是在大量的不确定性中找到确定性，这是我认为这件事情最核心的点。

## 程序员的三个层次

软件工程管理到底在管什么？和所有的管理活动一样无非就是人和事。所有的工程项目都希望找到最好的人，当然是在能给出的预算以内找到最好的人，有的人可能找不起。不同项目最大的差别就是事，不同的事在哪里？从做事的角度来讲我们招到的人可能会分三个层次（程序员三个级别），大家经常开玩笑说我是做搬砖的，所以第一个 level 我把他叫软件搬砖师，再然后是软件工程师、软件架构师。

软件搬砖师可以有很多。但今天数量其实还不算太多，因为我们知道这门学科只有 50 年的历史。但是好的一点是，产生软件搬砖师并不难，我做了一个长达四年的实践：从小学二年级开始教小学生编程。结论是做搬砖师不难，小学生也能做到。这是很有意思的一件事情，编程并不是非常复杂的学问，只要具备基本的逻辑能力，把常规的业务代码按部就班地垒出来，基本上可以算打到搬砖师水准。我自己认为这并不难。

软件工程师会相对难一些，我心目中的软件工程师首先在代码上会非常追求可读性、可维护性。另外，毕竟我们工程是群体协作，所以在群体协作上还是有自己的方法论和思考。比如说代码评审、单元测试。在我看来搬砖师和工程师的区别有很大不同。只要看他写的代码有没有注意可维护性，会和同伴交流的时候刻意去追求让同伴更好地理解自己的思想，是不是对单元测试比较抗拒，是不是比较乐意去做代码评审并且非常认同这件事情的价值，基本上通过这些事情就可以评判这个人是搬砖师还是工程师。

## 软件架构师的能力要求

谈到软件架构师，由于我毕业后两年在从事架构性质的工作，因此对软件架构师的特性有一些总结。首先在用户需求上，有判断能力和预见能力，此处的判断可以理解为对需求的鉴别，虽然这可能与产品经理最为相关，但架构师需要具备自己的判断力，当然这也包括对未来需求的预见能力；产品迭代上，有规划能力，判断需求哪些应该先满足，哪些后满足。架构师应该源于程序员，但不应局限于程序员视角。系统设计上，有分解和组合能力。技术选型上，有决策力。技术选型应该被认为是架构的一部分，我们非常反对开发人员随意选用开源组件，这是一件需要认真探讨的事情。人力资源上，有统筹能力，通俗地讲是“看菜做饭（看人下菜）”。

综上不难看出，架构师对综合能力要求比较高。这是因为我认为架构师需要对软件工程的结果负责，在不确定性和快速变化中寻找确定性。全局看软件发布流程，其比较重要的子过程有：需求分析（需求梳理 => 产品定义），系统设计（子系统划分 => 模块定义），模块设计（模块详细设计），编码实现，单元测试，代码评审，集成测试，灰度发布，正式发布等一系列过程。虽然有些过程看起来不属于架构师的范畴，但是这些活动过程属于软件工程的一部分，架构师一样需要全面参与把控。如果没有架构师把控就没有人观察得到全貌。正因为如此，软件架构师的要求相对较高。

如上所言，软件架构师需要具备产品经理的部分能力，因为需要对用户需求进行分析，并进行判断和预判，以及对产品迭代优先级进行把控。我自己习惯用如下图片表达软件架构师和产品经理之间的关系。


我认为，产品是“桥”，连接了两端，分别是用户需求和先进的技术。我一直认为，用户需求的变化非常缓慢，那么为什么产品会产生迭代？这是因为技术在迭代。本质上讲，产品迭代是技术迭代导致的需求满足方式的变化，所以产品实际上是一种需求满足的方式。

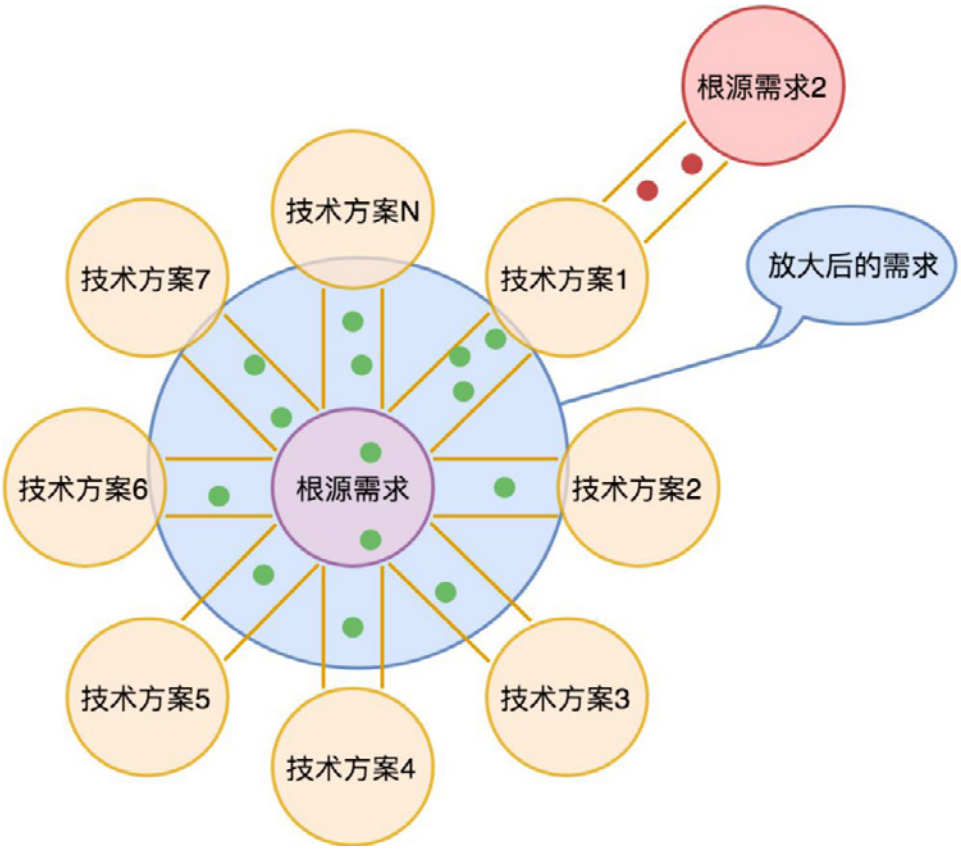
从这个意义上讲，架构师更多是从技术方案的角度看产品，而产品经理更多是从用户需求来看，但二者一定会碰头，只要能力提升到角色所期望的样子，越厉害就越具备两侧的能力。所以我认为，产品经理和架构师是一体两面，本质上对人的能力、诉求是相通的。产品经理在做产品架构，架构师在做技术架构，但最终目的一样。

## 从产品和需求视角看架构师

如果展开讲解产品定义过程，首先需要进行需求梳理，关心用户反馈。但是，很多用户反馈并不代表其根本性需求。有很多用户反馈需求的时候，往往已经带着他自己给出的解决方案。这种需求反馈已经属于二次加工的需求，而非原始需求。这个时候我们要多问多推敲，把它还原到不带任何技术实现假设的根源需求。

如上图所示，根源需求可能会有非常非常多的技术方案可以满足它。我们下面示意图中的小圆点是一个个用户反馈的需求。在用户提这些需求的时候，往往可能会带着他熟悉的技术方案的烙印。

产品都是通过提供相应的技术方案在满足用户的根源诉求，但技术一直在迭代进步，从而导致原有的解决方案过时落后，这种情况下需要新的解决方案出现。如果对用户反馈的需求全部满足，产品就会变得十分庞大，编程一个四不像的东西。

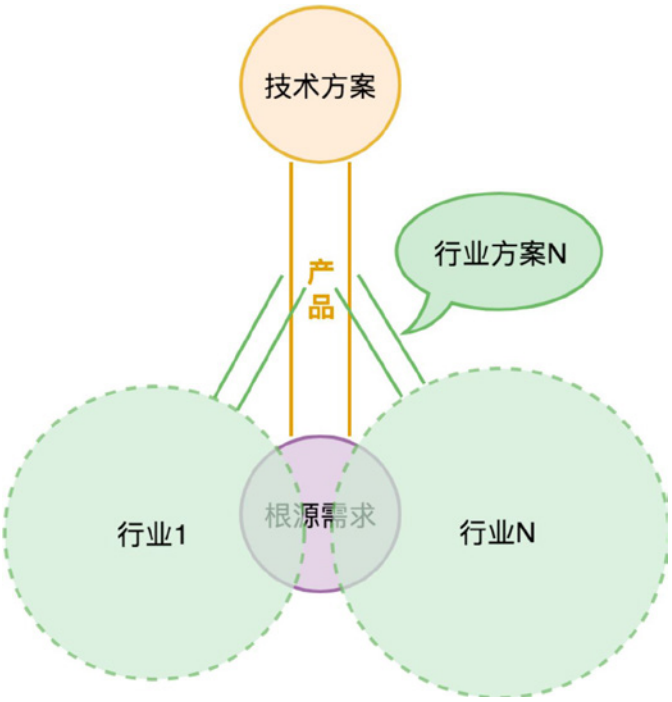



其次，在这个过程中，有些用户需求是稳定的，有些是变化的。举例来说，计算机系统结构从计算机诞生之后到现在没变过，但电子设备的形态发生了很大变化，从最早的大型机，到个人电脑，到笔记本，到手机，再到手表，形态变化剧烈。但为什么计算机系统结构能够适应需求而不用改变架构，这其实是非常值得思考的事情，其根源就是对变化点的抽象，找到系统需求的变化点，预见变化并做对应的开放式设计。本质上讲，架构师关心产品的核心根源就是预测变化。

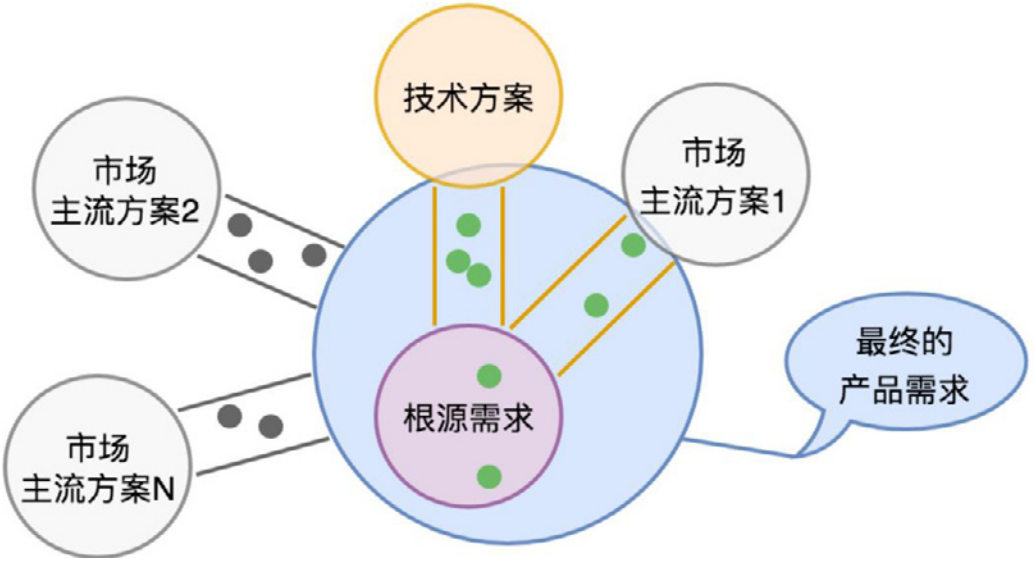
最后，理清产品边界。同样以计算机为例，经过多轮迭代，多样化外设（键盘等）变化较大，但CPU、内存演进较小，所以在变化点上做相应的开放式设计是必要的。同样的，需要与合作伙伴做边界设定，把变化开放出去让合作伙伴做，只有这样的产品才能达到较好效果。

从产品和解决方案角度来看，产品往往需要适应很多行业，但这个过程会让产品变得非常庞大。在我看来，产品应该为行业解决方案提供能力，行业解决方案优先选择合作伙伴做，以更加开放的心态看待这件事情，避免把行业方案视作产品的一部分。

梳理需求中比较关键的点是市场策略，需要解决的需求有非常多现成的方案，但哪些方案是主流的，哪些是最关键的都需要思考。虽然不能放大产品需求覆盖面，但也需要为某些关心既有市场的玩家做桥梁，这些桥梁也是产品的功能点。我倾向于认为关键市场可能会把既有玩家的能力适配到产品



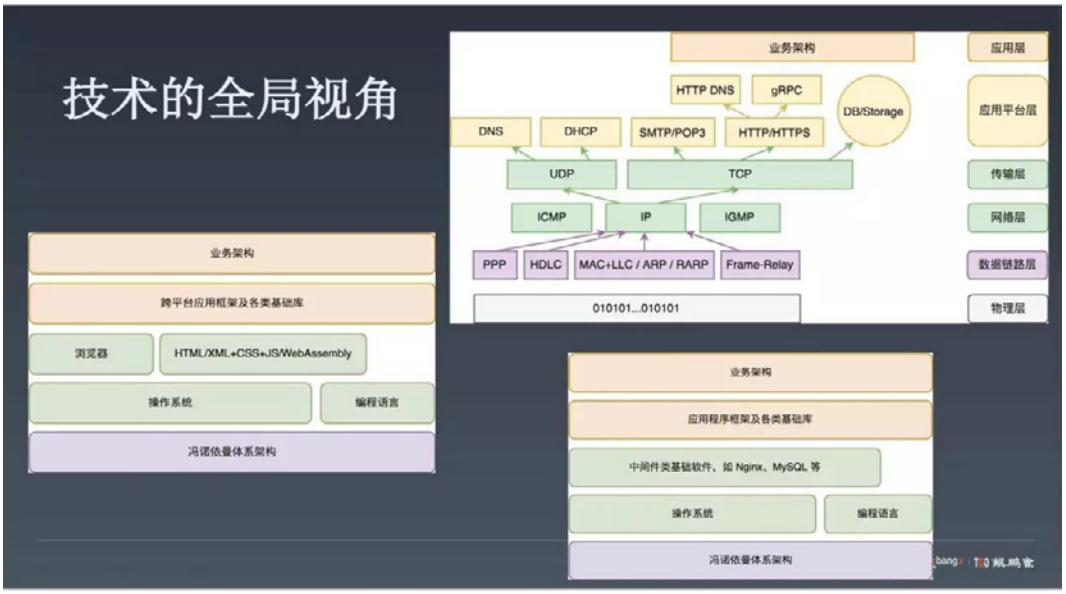

上作为很重要的功能，但是大部分市场主流方案我们还是提供“桥”，而不是自己解决掉。




### 从技术视角看架构师

以上是从产品和需求维度看架构师，从技术视角看，架构师很重要的能力是具备技术的全局视角，所谓的技术全貌是指从底到上的核心骨架，比如最底下的硬件结构、操作系统、编程语言，甚至浏览

器等，只有掌握每一层的核心思想，才能在架构设计中没有技术盲点。




从培养架构师的角度来看，为什么真正意义上的架构师比较难找？这是因为需要构建两个层次的能力：

1. 懂用户、懂市场，有一定市场洞察能力。作为技术人员，可能会不自觉、甚至不愿意和用户打交道，更希望坐在家里安静码代码。但是，作为架构师，不和用户打交道，成长会比较受限，不接触用户就无法理解用户需求，亲自和用户打交道倾听来的需求和探讨完全不一样。因此，架构师要尊重用户反馈，并学会思考需求分析和推演，这比技术能力更重要。架构的第一步就是需求分析，如果需求分析没做好，后续自然没办法做得很极致。

2. 建立技术上的全局视角。

以上两点是架构师最核心的两个能力。


主办方：Geekbang. 极客邦科技 InfoQ