开篇词 | 跳出单点思维模式，才能真正理解架构设计

你好，我是潘新宇，你也可以叫我 Jessen。目前我在京东集团担任团队架构负责人，常年深耕于业务一线从事业务后台的架构工作。

其间，我成功带领团队完成架构转型，将多个电商模块从烟囱式升级到平台化，最终演升到中台化；将懒加载模式的技术架构升级为能够支撑并发百万的读服务，落地了能够随机切库实现高可用的写服务，以及将并发扣减从单机几十的 TPS 优化至单机万级左右。

跳出单点知识去思考架构模式

在经历了多个平台化项目的实战后，我发现虽然业务场景不同，但很多平台的底层技术架构都是类似的。比如在构建一个外卖平台时，需要查询用户信息、商品信息、价格信息及外卖订单信息等业务功能，底层的技术都是要实现一个高性能、高可用的读服务。

如果能够掌握上述各个不同业务形态下的底层技术套路，我相信你在做架构设计时一定能够无往不利、事半功倍。**因为不管业务形态如何变化，只要掌握底层技术套路，你就可以直接进行架构移植。**

总体而言，架构是一项需要经过各种不同场景项目的磨炼才能够真正掌握的技能。而刚工作几年的初、中级工程师，更多的是参与日常业务需求的开发，接触的业务场景和难点问题都比较受限。我相信很多同学都有过这样的经历：

- 换工作跨行业后（比如从新闻资讯类公司跳到了电商公司），无法施展自己之前的架构经验，或者不知道如何复用；
- 在学习了一些技术点后，比如 Redis，明白了它性能快的原因，但仍然不知道如何通过它架构一套支持百万并发的读服务；
- 又比如在学习了多线程编程的技巧及数据库的 ACID 相关特性后，实际面对一个类似库存扣减的业务时，仍不知如何去设计一个能够支持高并发且不出现超卖的架构方案。
- 很多时候，你知道怎么做架构设计，但不能真正理解这样设计的原因，情况稍有点变动，马上就栽跟头。

出现以上种种情况的根本原因，是你没有将架构技能模式化、抽象化，而是从某一个技术点去学习，并长期在固定的“环境”中去应用，导致这些技能无法复用，做到一通百通。

如果你有仔细研究各大互联网公司的后台开发和架构岗位的招聘要求，也能发现一些价值信息。虽然各公司业务各异，但对技术知识的要求其实是非常相似的，除了要求你掌握某种开发语言和相关框架外，还要掌握分布式、多线程、缓存、数据库等。

（以上职位信息来源拉勾网）

看到这里，你可能会说：“招聘描述上明明要求学习这些共性的技术点，而不是某个架构模式。”但其实恰恰相反，**技术点都隶属于共性的架构模式，正因为架构模式需要技术点，它们才会被写进招聘要求里**。

也许你还会继续追问：“为什么不直接写要求你具备设计支撑百万并发的架构能力、掌握构建 7*24 不挂的高可用系统的能力呢？”因为太抽象了！如果你看到这样的招聘要求，是否清楚该如何准备面试呢？

往往这些抽象的、可复用的模式才是隐藏在招聘信息背后最重要的内容。比如，为什么各大后台开发岗位都需要你掌握隶属于 NoSQL 的 Redis 呢？就是因为 Redis 在后台架构里可以极大地提升读写的性能。

在微博的爆点吃瓜事件、电商的大促秒杀、分布式的并发防重、高并发的库存及支付扣减等场景里，Redis 都在架构方案里占据重要的角色，这才是它变得火爆的真正原因。因为 Redis 被架构验证，所以才被写进招聘要求里，并不是因为这个技术现在很“热”，才导致各大架构需要用到它。

希望你能够抓住事物的本质弄清逻辑关系。

业务场景的缺失，是你架构能力难以进阶的“原罪”

那么，说到学习架构模式或者构建架构体系，很多同学有这样的意识，也付出过努力，但是不少人在学习过种种资料后仍然困惑不得要领：“**学习这么多高度抽象的模式，面对具体场景时依然无从下手，还不如踏踏实实地去学习具体的技术点呢。**”

在我看来，虽然经久不衰的架构模式存在必有它的道理，但经过高度抽象、验证过的架构模式和准则太模式化了，导致很难准确地理解和应用在工作中。

比如常见的架构准则都会提到“架构应该是简洁的、职责单一的”，但对此，我想不少同学会困惑，到底什么算简洁？所有代码都写在一个单体模块是简洁的，毕竟架构上，它就只有一个模块，没有分层和垂直拆分，非常简单吧。

但我们知道事实并非如此。随着业务发展、客户增加，系统复杂度和耦合度也会越来越高，此时就会出现单体应用与业务扩展、客户体量、测试难度之间的矛盾。这会给整个部门乃至协作部门带来非常大的挑战，因此单体应用并不简洁。

究其原因，你没有经历过这些模式被推导出来的过程，因此代入感很低。

我相信你一定有这样的感受，对于亲身经历过的、事故频发、历经千难万苦的系统架构演化，从中获得的经验和道理一定能够让你刻骨铭记，并很容易在其他项目上复用。

但在**现实中，并不是每个人都有机会参与到此类项目的研发和架构设计，很多时候大厂的一些系统架构在早期已经固化了，后续介入的人只是参与日常需求开发，实际参与过互联网一线大厂日均上亿次调用的系统架构设计的人毕竟是少数**。你很难有时间和机会去亲身经历后台系统的架构设计演化。

以上种种困惑，我将在这个专栏中为你解答。

跳出被动灌输，理解架构设计

秉承可直接复用的原则，我尽可能把关键步骤全部展示给你，同时争取将一个技术点和方案的优缺点、适应场景、能够解决的问题通盘告诉你。

在这个专栏中，我们会一起学习各种架构模式，但并不是直接灌输给你，而是**基于实际案例的演化过程**，带你搞懂什么是高性能读业务、高并发扣减业务和高可用的写业务的架构模式，以及为什么要这样设计，让你知其然并知其所以然。

具体来讲，我按总-分-总的思路，将专栏分为 5 个模块。

- 模块一：后台业务系统的架构模式

我将对业务后台系统进行一个边界定义，对不同边界的后台系统分门别类，并分别介绍其实现上需要满足的架构要求。这个模块，我希望帮助你建立对后台系统的全局业务视角，打造对各类系统按业务（垂直拆分）及技术特点（水平拆分）进行架构设计的能力。

模块二~模块四，将对后台系统架构设计的三大问题——高性能、高可用、高并发——进行深入探讨，并通过实际案例告诉你在解决问题时如何科学拆解、在选择实现方案时如何进行取舍。

- 模块二：构建高性能的读服务

读业务最重要的是提升性能。首先，我会介绍一个基本满足性能要求使用缓存懒加载模式的读业务实现方案；然后针对此方案存在的问题，手把手带你实现一个简单能规避缓存过期、穿透雪崩等问题的升级方案，以及读数据热点的应对方案。此外，我还会选取实际工作中的高频问题，教你如何大幅度降低读服务的测试回归时间，从而提升线上质量。

- 模块三：构建高可用的数据写入服务

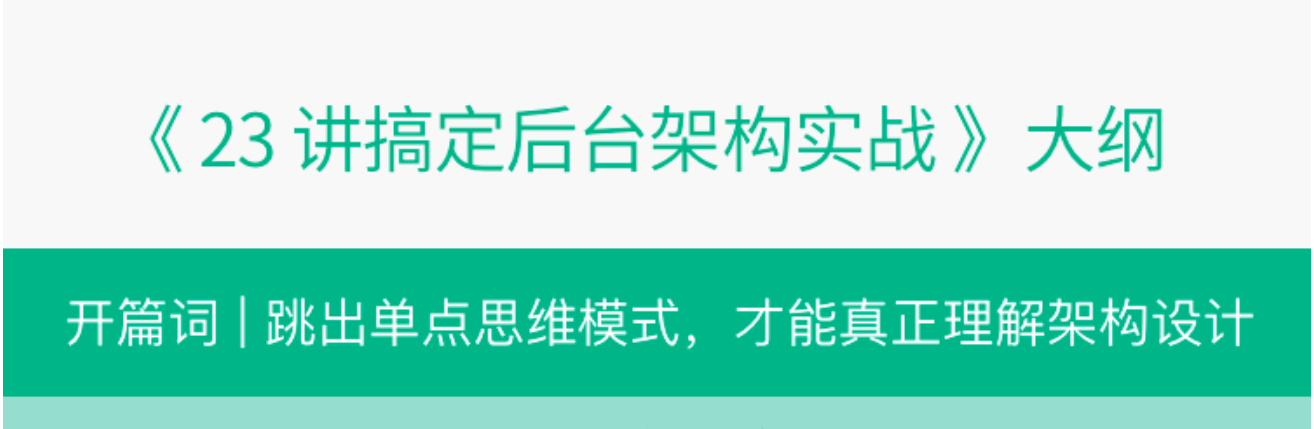
写类型的业务最重要的是满足高可用。我将介绍如何使用分库分表完成一个基本的、高可用的写入方案；然后随着需求的增加，介绍如何构建一个无状态的写入存储，实现随时随地可以切库的高可用方案；最后介绍在分库分表后，如何满足跨分库路由的多维度查询。

- 模块四：构建高并发的扣减服务

扣减类业务最关键的是保障高并发的应对能力。首先，我会介绍扣减类服务在实现上需要满足哪些功能性诉求；然后，介绍 3 种递进式的实现方案及它们的适用场景（从纯数据库到纯缓存，再到数据库 + 缓存）；最后针对秒杀场景，讲解如何实现千万级并发均命中相同数据。

- 模块五：通用架构模式

将会对一些共性的问题进行深入讨论，带你掌握在上述架构方案之外的、构建一个微服务必不可少的架构技能，包含对外 SDK 设计、部署策略、服务解耦和分布式事务、立体化监控，以及如何应对系统重构等内容，进而帮助你构建一个更加完整的架构体系。



模块一：后台业务系统的架构模式

模块二：构建高性能的读服务

模块三：构建高可用的数据写入服务

模块四：构建高并发的扣减服务

模块五：通用架构模式

结束语 | 抓住本质，是成为技术专家的不二法则

讲师寄语

学完本专栏后，我希望你能够掌握实现“三高”系统架构的通用方法论，并能够在自己所负责的系统中落地应用。

我还希望你去面试或者入职一家新公司后，能够看穿披在业务面纱后的技术实现共同点，可以直接复用本专栏向你介绍的或者你沉淀消化后的架构模式。

通过模式化的架构思维去分析和思考，这是我希望这个专栏可以传递给你的一个长期理念，相信你在成为后台系统架构师之路上会如虎添翼，一路乘风破浪，从技术骨干逐步成长为能够独立支撑一个技术方向的架构师。

三人行必有我师，让我们一起在讨论中碰撞出意想不到的知识火花。