编者按：本文作者王庆友，前 1号店首席架构师，先后就职于 ebay、腾讯、1号店、找钢网，精通电商业务，擅长复杂系统业务建模和架构分析，目前在中国 B2B 第一电商公司找钢网担任首席架构师，微信号Brucetwins，欢迎一起聊架构。

目前讨论架构实操（术）的文章较多，讨论架构理念（道）的较少，本文基于作者在大型电商系统架构方面的一些实践和思考，和大家聊聊架构理念性的东西，希望能够抛砖引玉，推进大家对架构的认识。

什么是道，什么是术？道是事物发展的本质规律，术是事物发展的具体途径。规律只有一个，途径很多，条条大路通罗马，罗马是道，大路是术。道为本，术为途，如果事先知道罗马在哪里，那么遍地是路，路路相通。架构也是如此，如果能领悟架构的本质，就不会拘泥于现有的实践和理论框框，而以最直接的方式解决问题，无招胜有招。本文的内容包括架构的本质、架构的服务对象、架构师能力模型 、架构境界等。

架构本质

任何系统，自然情况下，都是从有序到无序，这是有科学依据的， 按照热力学第二定律，自然界的一切自发过程都有方向性，一个孤立系统会由有序变为无序，即它的熵会不断增加，最终寂灭。但生物可以通过和外界交互，主动进 行新陈代谢，制造“负熵”来保证自身有序，继续生存。

同样，一个软件系统随着功能越来越多，调用量急剧增长，整个系统逐渐碎片化，越来越无序，最终无法维护和扩展，所以系统在一段时间的野蛮生长后，也需要及时干预，避免越来越无序。

架构的本质就是对系统进行有序化重构，不断减少系统的“熵”，使系统不断进化。

那架构是如何实现无序到有序的呢？ 基本的手段就是分和合，先把系统打散，然后重新组合。

分：合理定位

合：有机整体

分的过程是把系统拆分为各个子系统 / 模块 / 组件，拆的时候，首先要解决每个组件的定位问题，然后才能划分彼此的边界，实现合理的拆分。合就是根据最终要求，把各个分离的组件有机整合在一起，相对来说，第一步的拆分更难。

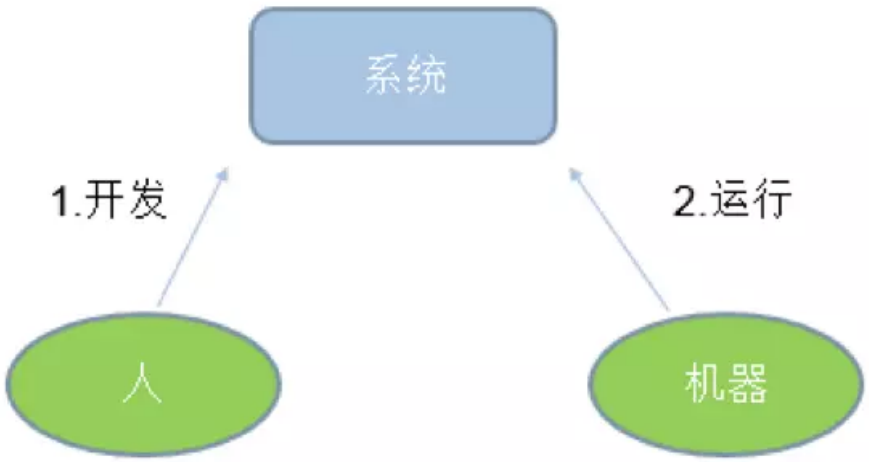
拆分的结果使开发人员能够做到业务聚焦、技能聚焦，实现开发敏捷，合的结果是系统变得柔性，可以因需而变，实现业务敏捷。

举个例子，在 Web 1.0 时代，一个 ASP 或 JSP 页面里，HTML 和脚本代码混在一起，此时脚本代码越多，系统越混乱（即熵增加），最终连开发者自己都无法理解。此时就需要对系统重新架构，办法是引入 view helper 模式，分离 HTML 和脚本，HTML 成为 view，脚本成为帮助类。然后再简单整合在一起。通过重新分和合，整个系统层次清晰，职责明确，系统的无序度降低，容易扩展。同时不同技能的开发人员， 如 UED 和程序员，可以负责不同部分，有效提高开发效率。

好的架构就像一篇优美的散文，形散神不散，表面看无序，实则高度有序。

架构分类和服务对象

架构一般可分业务架构、应用架构、技术架构，那么它们分别解决什么问题，服务于谁呢？ 我们首先看一个系统落地过程：



对于负责开发的人来说，怕的是业务太复杂，代码逻辑太乱，超出他能理解的范畴，系统无法维护。因此开发的需求是系统整体概念清晰，容易理解，方便扩展。

对于负责运行的机器来说，怕的是业务并发量太大，系统核心资源不够用（如数据库连接）。它希望在业务量增加时，系统能够支持水平扩展，支持硬件容错（如避免单点故障）。

开发的痛点主要由业务架构和应用架构解决，业务架构从概念层面帮助开发理解系统（动态的包括业务流程 / 节点 / 输入输出，静态的包括业务域 / 业务模块 / 单据模型）。

应用架构从逻辑层面帮助开发落地系统（应用种类 / 应用形式 / 数据交互关系 / 交互方式等），整个系统逻辑上容易理解，最近大家谈的比较多的 SOA 即属于应用架构的范畴。

机器的痛点主要由技术架构解决，如技术平台选型（操作系统 / 中间件 / 设备等），部署上希望支持多机房，水平扩展，无单点等。

强调一下，系统是人的系统，架构首先是为人服务的，业务概念清晰、应用逻辑合理、人好理解是第一位的（即系统有序度高）。现在大家讨论更多的是技术 架

构，如高并发设计，分布式事务处理等，只是因为这个不需要业务上下文背景，比较好相互沟通。具体架构设计时，首先要关注业务架构和应用架构，这个架构新 手要特别注意。

架构师能力模型

架构师只做分和合的事情，但综合能力要求很高，要求内外兼修，下得厨房，上得厅堂，下图通过典型的架构方式介绍一个架构师的能力要求：

在此基础上，架构师要有技术的广度（多领域知识），又有深度（技术前瞻），对主流公司的系统设计非常了解，知道优劣长短，碰到实际问题，很快有多种方案可供评估。

抽象思维是架构师最重要的能力，架构师要善于把实物概念化并归类。比如面对一个大型的 B2C 网站，能够迅速抽象为采购->运营->前台搜索->下单->履单这几大块，对系统分而治之，庖丁解牛，早已目无全牛。

抽象思维是往高层次的总结升华，由实到虚；而透过问题看本质则是由虚到实，往深层次地挖掘。比如看到一段 java 代码，知道它在 JVM 如何执行；一个跨网络调用，知道数据是如何通过各种介质到达目标 (操作系统内核 / 网卡端口 / 电磁介质等）。透过问题看本质使架构师能够敏锐地发现底层之真实，系统性端到端地思考问题，识别木桶的短板并解决之。

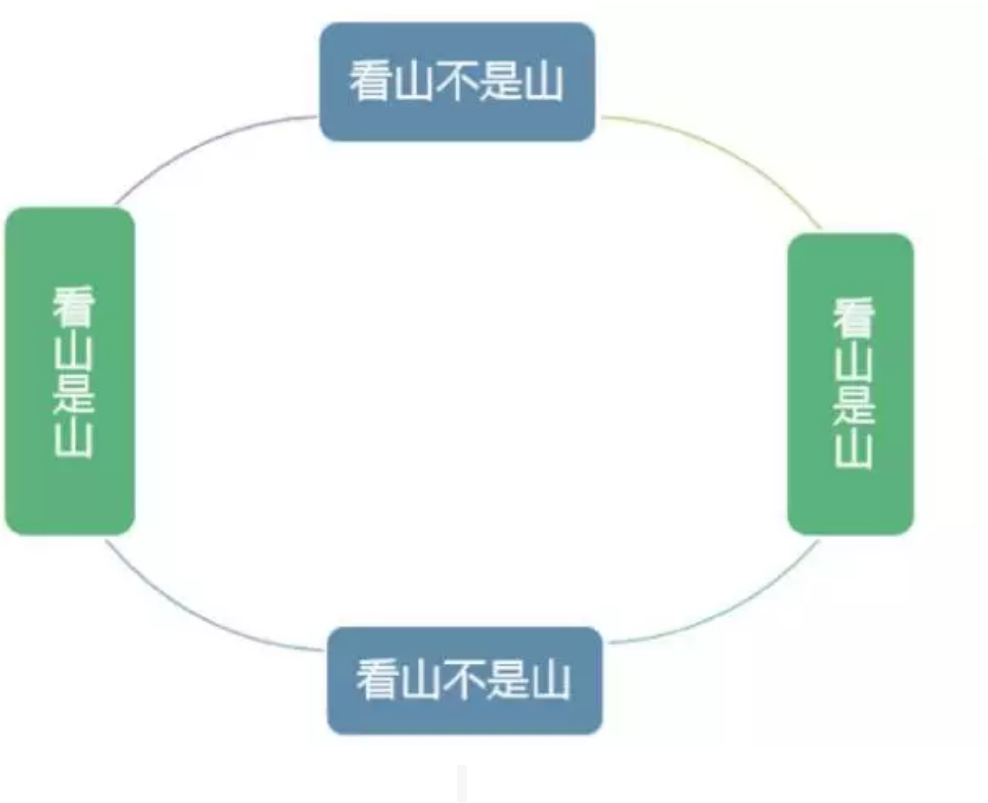
能落地的架构才是好架构，良好的沟通能力确保各方对架构达成共识，愿意采取行动；良好的平衡取舍能力确保架构在现有资源约束下是最合理的，理想最终照进现实。

总结下，架构师的能力要求包括：

- 兼具技术的广度（多领域知识）和深度（技术前瞻）
- 兼具思维的高度（抽象思维）和深度（问题到本质）
- 兼具感性（沟通）和理性（平衡）

架构境界

架构师从境界上由浅到深可以分为四层：第一看山不是山，第二看山是山，第三看山不是山，第四看山是山。



刚接手项目时，对业务不了解，时时被业务方冒出的术语弄得一愣一愣的，如果把现有问题比作山，则是横看成岭侧成峰，根本摸不透，此时看山不是山。

经过业务梳理和对系统深入了解，可以设计出一个屌丝的方案，把各个系统串起来，解决当前的问题，对当前这个山能够看清楚全貌，此时能够做到看山是山。

通过进一步抽象，发现问题的本质，原来这个问题是共性的，后续还会有很多类似问题。设计上进行总结和升华，得出一个通用的方案，不光能解决当前的问题，还可以解决潜在的问题。此时看到的已经是问题本质，看山不是山。

最后回到问题本身，去除过度的抽象，给出的设计简洁明了，增之一分嫌肥，减之一分嫌瘦，既解决当前问题，又保留最基本的扩展，此时问题还是那个问题，山还是那个山。

第一境界给不出合适方案，不表。

第二境界的方案只解决表面问题，往往设计不够，碰到其它类似问题或者问题稍微变形，系统需要重新做。

第三境界的方案往往过度设计，太追求通用化会创造出过多抽象，生造概念，理解和实现均困难，此时系统的无序度反而增加，过犹不及。

第四境界的方案，在了解问题本质的基础上，同时考虑现状，评估未来，不多做，不少做。

佛教讲空和色，色即事物现象，空即事物本质，从这个意义上说，第一重境界无色无空，第二重境界过色，第三重境界过空，第四重境界站在色和空之间，既色又空，不执着于当前，不虚无于未来。

不空不色，既空既色，道法自然，本性如来，架构之髓也。