

09 | 架构设计原则案例

2018-05-17 李运华

从0开始学架构

[进入课程 >](#)



讲述：黄洲君

时长 13:09 大小 6.07M



周二，我给你介绍了架构设计的三条核心原则，先复习一下：合适原则、简单原则和演化原则。我们在架构设计实践中，应该时刻谨记这三条设计原则，指导我们设计出合适的架构，即使是代表中国互联网技术最顶尖水平的 BAT，其架构的发展历程也同样遵循这三条原则。

今天我就以大家耳熟能详的[淘宝和手机 QQ](#) 作为案例，来简单分析一下。

淘宝

注：以下部分内容摘自《淘宝技术发展》。

淘宝技术发展主要经历了“个人网站” → “Oracle/ 支付宝 / 旺旺” → “Java 时代 1.0” → “Java 时代 2.0” → “Java 时代 3.0” → “分布式时代”。我们看看每个阶段的主要驱动力是什么。

1. 个人网站

2003 年 4 月 7 日马云提出成立淘宝，2003 年 5 月 10 日淘宝就上线了，中间只有 1 个月，怎么办？淘宝的答案就是：买一个。

估计大部分人很难想象如今技术牛气冲天的阿里最初的淘宝竟然是买来的，我们看看当初决策的依据：

当时对整个项目组来说压力最大的就是时间，怎么在最短的时间内把一个从来没有的网站从零开始建立起来？了解淘宝历史的人知道淘宝是在 2003 年 5 月 10 日上线的，这之间只有一个月。要是你在这个团队里，你怎么做？我们的答案就是：买一个来。

淘宝当时在初创时，没有过多考虑技术是否优越、性能是否海量以及稳定性如何，主要的考虑因素就是：快！

因为此时业务要求快速上线，时间不等人，等你花几个月甚至十几个月搞出一个强大的系统出来，可能市场机会就没有了，黄花菜都凉了。

同样，在考虑如何买的时候，淘宝的决策依据主要也是“快”。

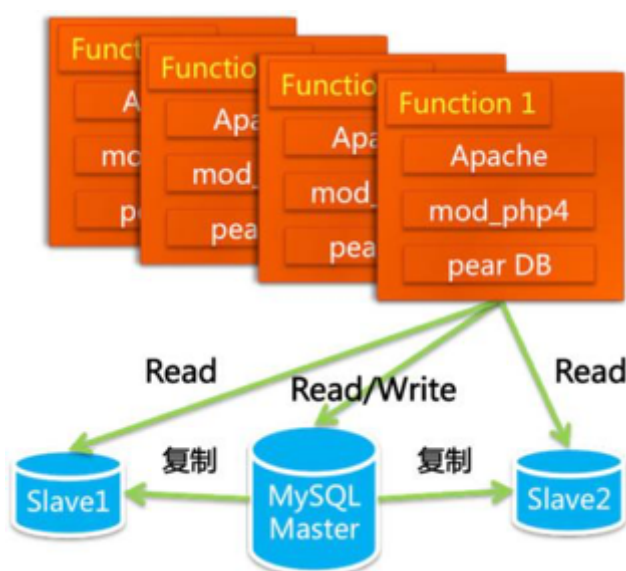
买一个网站显然比做一个网站要省事一些，但是他们的梦想可不是做一个小网站而已，要做大，就不是随便买个就行的，要有比较低的维护成本，要能够方便地扩展和二次开发。

那接下来就是第二个问题：买一个什么样的网站？答案是：轻量一点的，简单一点的。

买一个系统是为了“快速可用”，而买一个轻量级的系统是为了“快速开发”。因为系统上线后肯定有大量的需求需要做，这时能够快速开发就非常重要。

从这个实例我们可以看到：淘宝最开始的时候业务要求就是“快”，因此反过来要求技术同样要“快”，业务决定技术，这里架构设计和选择主要遵循的是“合适原则”和“简单原则”。

第一代的技术架构如图所示。



2. Oracle/ 支付宝 / 旺旺

淘宝网推出后，由于正好碰到“非典”，网购很火爆，加上采取了成功的市场运作，流量和交易量迅速上涨，业务发展很快，在 2003 年底，MySQL 已经撑不住了。

一般人或者团队在这个时候，可能就开始优化系统、优化架构、分拆业务了，因为这些是大家耳熟能详也很拿手的动作。那我们来看看淘宝这个时候怎么采取的措施：

技术的替代方案非常简单，就是换成 Oracle。换 Oracle 的原因除了它容量大、稳定、安全、性能高，还有人才方面的原因。

可以看出这个时候淘宝的策略主要还是“买”，买更高配置的 Oracle，这个是当时情况下最快的方法。

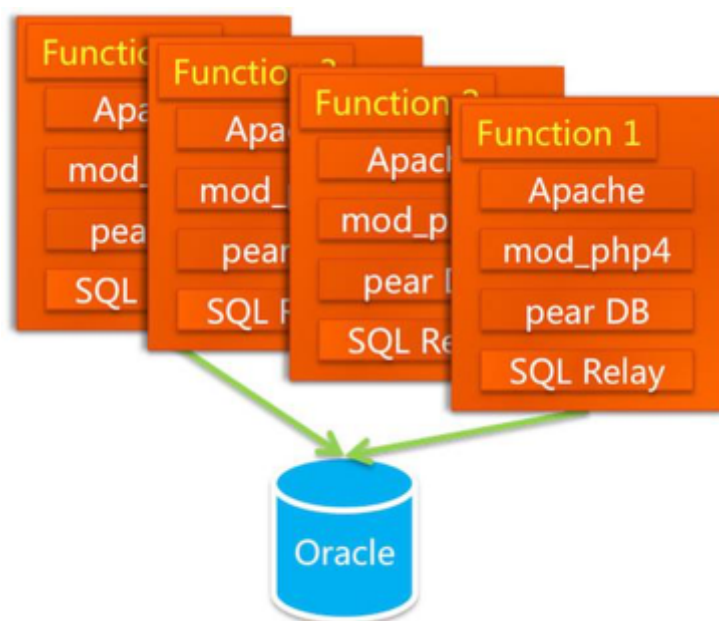
除了购买 Oracle，后来为了优化，又买了更强大的存储：

后来数据量变大了，本地存储不行了。买了 NAS（Network Attached Storage，网络附属存储），NetApp 的 NAS 存储作为数据库的存储设

备，加上 Oracle RAC (Real Application Clusters , 实时应用集群) 来实现负载均衡。

为什么淘宝在这个时候继续采取“买”的方式来快速解决问题呢？我们可以从时间上看出端倪：此时离刚上线才半年不到，业务飞速发展，最快的方式支撑业务的发展还是去买。如果说第一阶段买的是“方案”，这个阶段买的就是“性能”，这里架构设计和选择主要遵循的还是“合适原则”和“简单原则”。

换上 Oracle 和昂贵的存储后，第二代架构如图所示。



3. 脱胎换骨的 Java 时代 1.0

淘宝切换到 Java 的原因很有趣，主要因为找了一个 PHP 的开源连接池 SQL Relay 连接到 Oracle，而这个代理经常死锁，死锁了就必须重启，而数据库又必须用 Oracle，于是决定换个开发语言。最后淘宝挑选了 Java，而且当时挑选 Java，也是请 Sun 公司的人，这帮人很厉害，先将淘宝网站从 PHP 热切换到了 Java，后来又做了支付宝。

这次切换的最主要原因是因为技术影响了业务的发展，频繁的死锁和重启对用户业务产生了严重的影响，从业务的角度来看这是不得不解决的技术问题。

但这次淘宝为什么没有去“买”呢？我们看最初选择 SQL Relay 的原因：

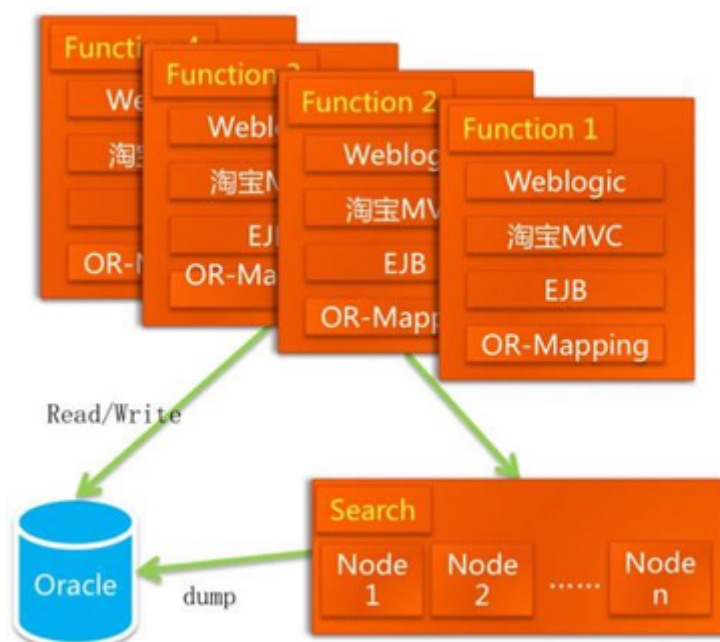
但对于 PHP 语言来说，它是放在 Apache 上的，每一个请求都会对数据库产生一个连接，它没有连接池这种功能（Java 语言有 Servlet 容器，可以存放连接池）。那如何是好呢？这帮人打探到 eBay 在 PHP 下面用了一个连接池的工具，是 BEA 卖给他们的。我们知道 BEA 的东西都很贵，我们买不起，于是多隆在网上寻寻觅觅，找到一个开源的连接池代理服务 SQL Relay。

不清楚当时到底有多贵，Oracle 都可以买，连接池买不起？所以我个人感觉这次切换语言，更多是为以后业务发展做铺垫，毕竟当时 PHP 语言远远没有 Java 那么火、那么好招人。淘宝选择 Java 语言的理由可以从侧面验证这点：

Java 是当时最成熟的网站开发语言，它有良好的企业开发框架，被世界上主流的大规模网站普遍采用，另外有 Java 开发经验的人才也比较多，后续维护成本会比较低。

综合来看，这次架构的变化没有再简单通过“买”来解决，而是通过重构来解决，架构设计和选择遵循了“演化原则”。

从 PHP 改为 Java 后，第三代技术架构如图所示。



4. 坚若磐石的 Java 时代 2.0

Java 时代 2.0，淘宝做了很多优化工作：数据分库、放弃 EJB、引入 Spring、加入缓存、加入 CDN、采用开源的 JBoss。为什么在这个时候要做这些动作？原文作者很好地概括了

做这些动作的原因：

这些杂七杂八的修改，我们对数据分库、放弃 EJB、引入 Spring、加入缓存、加入 CDN、采用开源的 JBoss，看起来没有章法可循，其实都是围绕着提高容量、提高性能、节约成本来做的。

我们思考一下，为什么在前面的阶段，淘宝考虑的都是“快”，而现在开始考虑“容量、性能、成本”了呢？而且为什么这个时候不采取“买”的方式来解决容量、性能、成本问题呢？

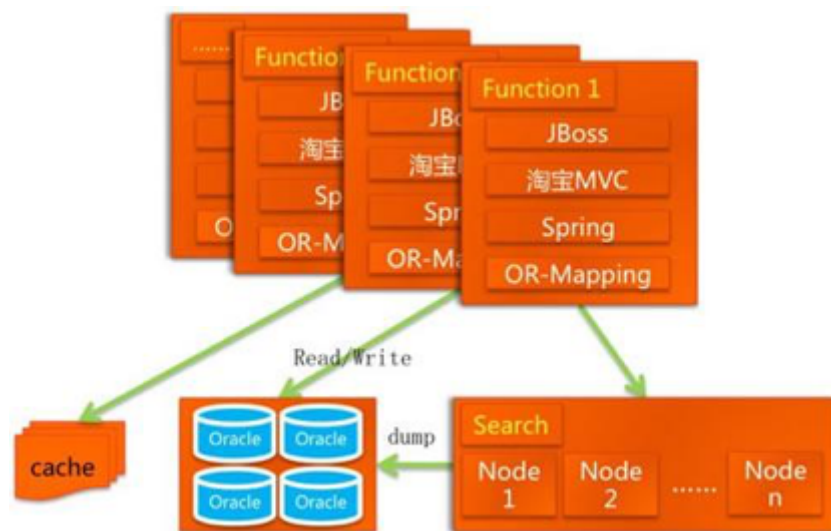
简单来说，就是“买”也搞不定了，此时的业务发展情况是这样的：

随着数据量的继续增长，到了 2005 年，商品数有 1663 万，PV 有 8931 万，注册会员有 1390 万，这给数据和存储带来的压力依然很大，数据量大，性能就慢。

原有的方案存在固有缺陷，随着业务的发展，已经不是靠“买”就能够解决问题了，此时必须从整个架构上去进行调整和优化。比如说 Oracle 再强大，在做 like 类搜索的时候，也不可能做到纯粹的搜索系统如 Solr、Sphinx 等的性能，因为这是机制决定的。

另外，随着规模的增大，纯粹靠买的一个典型问题开始成为重要的考虑因素，那就是**成本**。当买一台两台 Oracle 的时候，可能对成本并不怎么关心，但如果要买 100 台 Oracle，成本就是一个关键因素了。这就是“量变带来质变”的一个典型案例，业务和系统发生质变后，架构设计遵循“演化原则”的思想，需要再一次重构甚至重写。

Java 架构经过各种优化，第四代技术架构如图所示。



5.Java 时代 3.0 和分布式时代

Java 时代 3.0 我个人认为是淘宝技术飞跃的开始，简单来说就是淘宝技术从商用转为“自研”，典型的就去 IOE 化。

分布式时代我认为淘宝技术的修炼成功，到了这个阶段，自研技术已经自成一派，除了支撑本身的海量业务，也开始影响整个互联网的技术发展。

到了这个阶段，业务规模急剧上升后，原来并不是主要复杂度的 IOE 成本开始成为了主要的问题，因此通过自研系统来降低 IOE 的成本，去 IOE 也是系统架构的再一次演化。

手机 QQ

注：以下部分内容摘自《QQ 1.4 亿在线背后的故事》。

手机 QQ 的发展历程按照用户规模可以粗略划分为 4 个阶段：十万级、百万级、千万级、亿级，不同的用户规模，IM 后台的架构也不同，而且基本上都是用户规模先上去，然后产生各种问题，倒逼技术架构升级。

1. 十万级 IM 1.X

最开始的手机 QQ 后台是这样的，可以说是简单得不能再简单、普通得不能再普通的一个架构了，因为当时业务刚开始，架构设计遵循的是“合适原则”和“简单原则”。



2. 百万级 IM 2.X

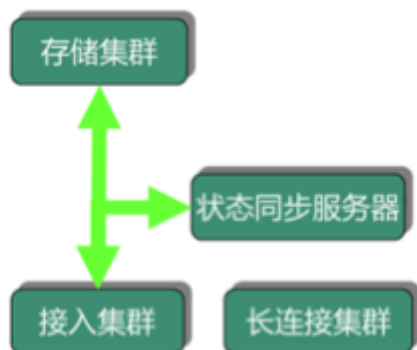
随着业务发展到 2001 年，QQ 同时在线人数也突破了一百万。第一代架构很简单，明显不可能支撑百万级的用户规模，主要的问题有：

以接入服务器的内存为例，单个在线用户的存储量约为 2KB，索引和在线状态为 50 字节，好友表 $400 \text{ 个好友} \times 5 \text{ 字节 / 好友} = 2000 \text{ 字节}$ ，大致来说，2GB 内存只能支持一百万在线用户。

CPU/ 网卡包量和流量 / 交换机流量等瓶颈。

单台服务器支撑不下所有在线用户 / 注册用户。

于是针对这些问题做架构改造，按照“演化原则”的指导进行了重构，重构的方案相比现在来说也还是简单得多，因此当时做架构设计时也遵循了“合适原则”和“简单原则”。IM 2.X 的最终架构如图所示。



3. 千万级 IM 3.X

业务发展到 2005 年，QQ 同时在线人数突破了一千万。第二代架构支撑百万级用户是没问题的，但支撑千万级用户又会产生新问题，表现有：

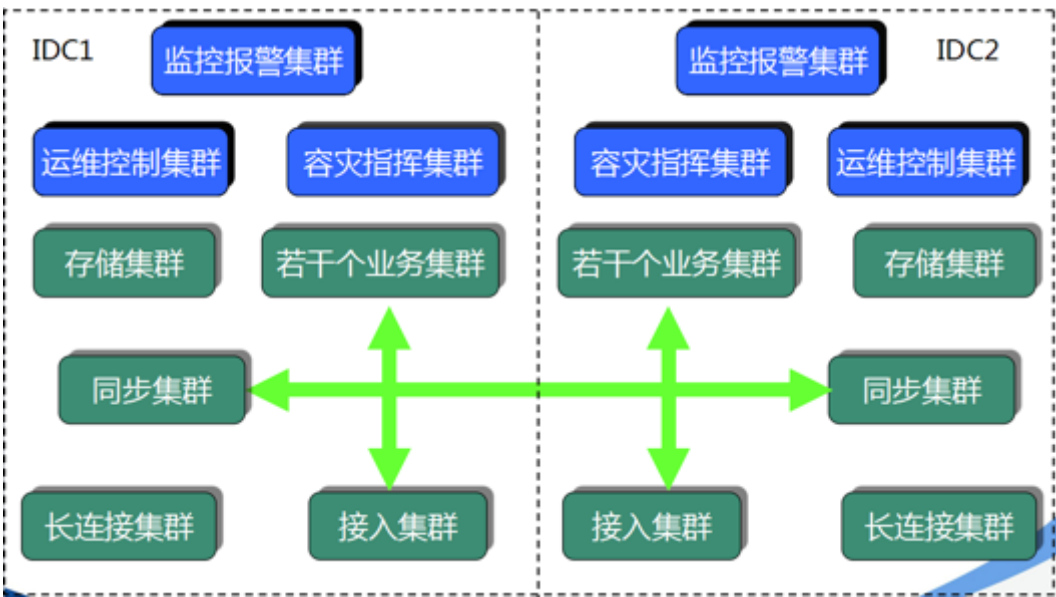
同步流量太大，状态同步服务器遇到单机瓶颈。

所有在线用户的在线状态信息量太大，单台接入服务器存不下，如果在线数进一步增加，甚至单台状态同步服务器也存不下。

单台状态同步服务器支撑不下所有在线用户。

单台接入服务器支撑不下所有在线用户的在线状态信息。

针对这些问题，架构需要继续改造升级，再一次“演化”。IM 3.X 的最终架构如下图，可以看到这次的方案相比之前的方案来说并不简单了，这是业务特性决定的。



4. 亿级 IM 4.X

业务发展到 2010 年 3 月，QQ 同时在线人数过亿。第三代架构此时也不适应了，主要问题有：

灵活性很差，比如“昵称”长度增加一半，需要两个月；增加“故乡”字段，需要两个月；最大好友数从 500 变成 1000，需要三个月。

无法支撑某些关键功能，比如好友数上万、隐私权限控制、PC QQ 与手机 QQ 不可互踢、微信与 QQ 互通、异地容灾。

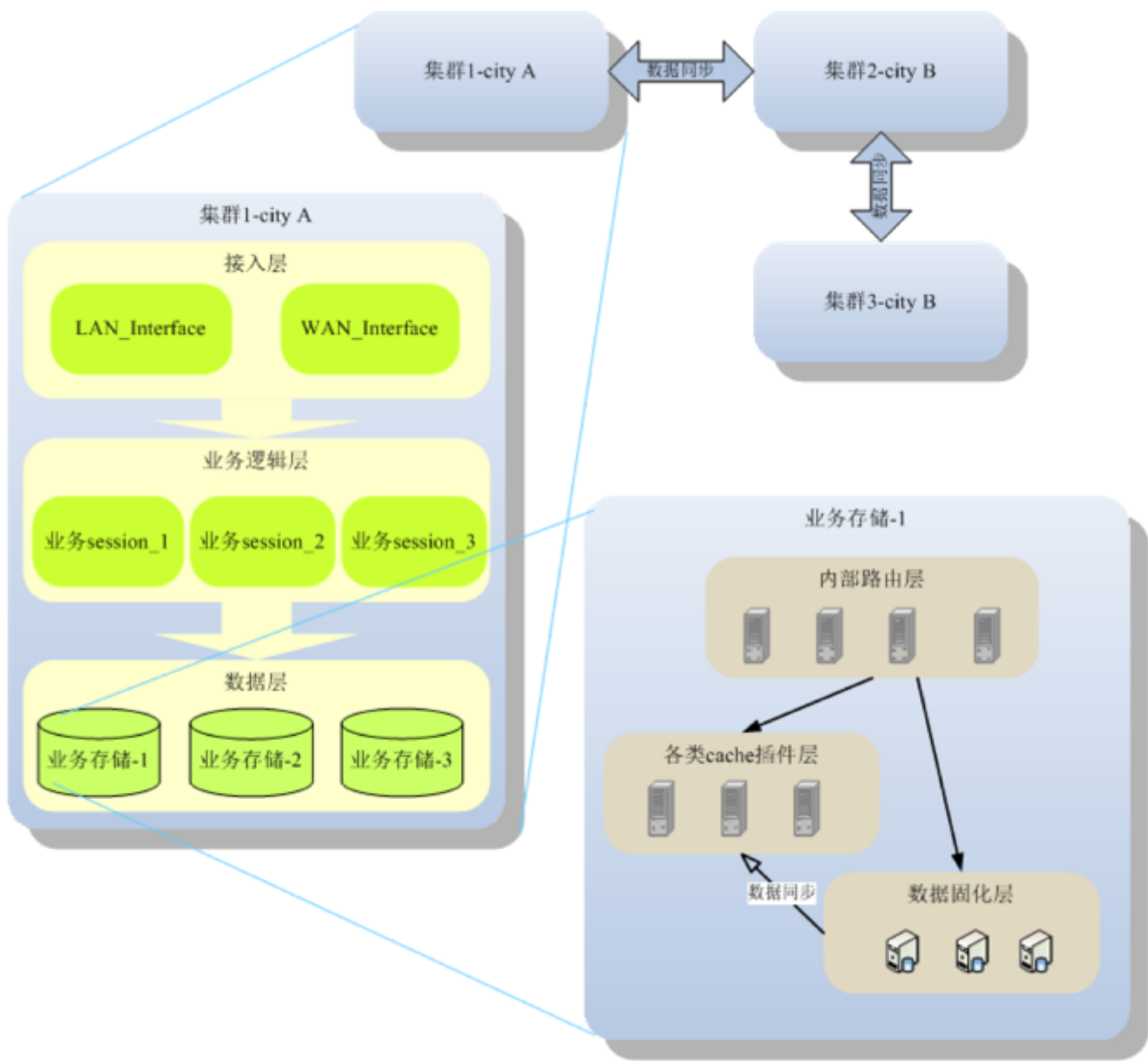
除了不适应，还有一个更严重的问题：

IM 后台从 1.0 到 3.5 都是在原来基础上做改造升级的，但是持续打补丁已经难以支撑亿级在线，IM 后台 4.0 必须从头开始，重新设计实现！

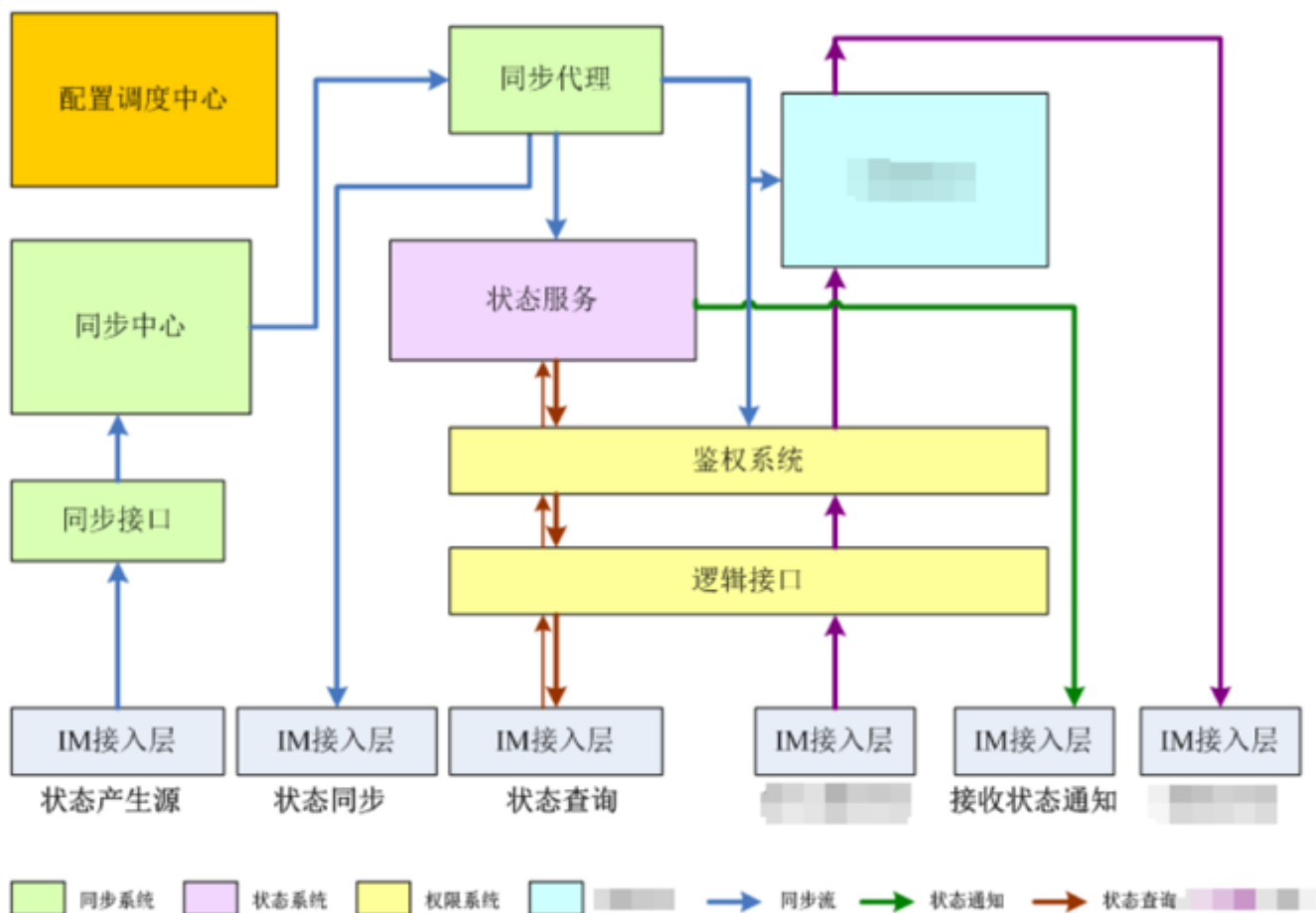
这里再次遵循了“演化原则”，决定重新打造一个这么复杂的系统，不得不佩服当时决策人的勇气和魄力！

重新设计的 IM 4.0 架构如图所示，和之前的架构相比，架构本身都拆分为两个主要的架构：存储架构和通信架构。

存储架构



通信架构



小结

今天我给你讲了淘宝和手机 QQ 两个典型互联网业务的架构发展历程，通过这两个案例我们可以看出，即使是现在非常复杂、非常强大的架构，也并不是一开始就进行了复杂设计，而是首先采取了简单的方式（简单原则），满足了当时的业务需要（合适原则），随着业务的发展逐步演化而来的（演化原则）。罗马不是一天建成的，架构也不是一开始就设计成完美的样子，然后可以一劳永逸一直用下去。

这就是今天的全部内容，留一道思考题给你吧。搜索一个互联网大厂（BATJ、TMD 等）的架构发展案例，分析一下其发展过程，看看哪些地方体现了这三条架构设计原则。

欢迎把你的答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

从0开始学架构

资深技术专家的
实战架构心法

李运华 资深技术专家



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | 架构设计三原则

下一篇 架构专栏特别放送 | “华仔，放学别走！” 第1期

精选留言 (43)

写留言



公号-代码...

2018-05-17

今日心得

29

通过文中对淘宝和手机 QQ 两个典型互联网业务的架构发展历程的详细拆解，可以看出，大型互联网架构发展的最重要甚至唯一的驱动力就是“情非得已”，伴随业务的高速发展，用户、数据、流量、业务复杂度都在呈指数级增长，总会突破现有商业、开源条件能够...
展开



何国平

2018-05-18

所以有些公司要求两三年就重构一次

17

展开 ∨

作者回复: 至少说明这个公司重视技术



小小笑儿

2018-05-18

👍 15

适合原则，简单原则，演化原则这三个点归纳得很精辟，本次课程也通过实例来讲解了各个原则的使用。但是我还有个小小的疑问，就是在对这些成功项目进行复盘的时候我们可以很清楚的辨别什么时候采用什么原则优先，而如果把我们将回到之前选择的时间点上，如何更准确的判断应该采用什么原则呢？

展开 ∨

作者回复: 合适原则第一考虑，优先满足业务需求；

简单原则第二考虑，挑选简单方案快速落地验证；

演进原则第三考虑，适当预测业务发展，感觉预测不准就不预测，等真的出现问题的时候演进即可



李彦余

2018-05-21

👍 6

架构这种话题不好讲，原则适用性更不好把控，概念性的东西太多，简单原则相对最好把控，至于合适原则与演绎，这个主要就看架构师的理解和实际能力了。

特别说到有些行业追求的是概念，新兴公司追求实用，更是不好取舍，最终又变成了怎么把控真正的需求，这个才是核心。

唯一可量化的就是复杂度了，所以非功能性的普适性更高。

展开 ∨



xinsz08

2018-05-18

👍 5

能不能讲下关于并发和pv，日活，在线人数的计算方式和关系

作者回复: 跟业务有关，不存在恒等式呢

一般来说：

$pv = \text{日活} \times \text{人均点击次数}$

并发均值=pv/86400
并发峰值=并发均值*N



米斯特.杜

2018-05-17

👍 5

探讨个问题，希望可以得到回答。之前看过云栖社区对faceu的cto的采访，他说当时他们上第一版后台系统时就直接选型了阿里云的drds，他自己也不知道用户量会不会上来。他还说如果当时他没选drds方案，到后来用户井喷式增长时系统再做调整估计等不到改好用户就跑光了。所以我的问题是，对于像这种近两年创业的公司可能做出爆款产品时，系统架构方面是否需要向前多考虑一些？

展开 ▾

作者回复: 站在巨人的肩膀上，其实他们也是选了简单的方案，同样遵循合适原则和简单原则。
2004年时遵循简单原则是买oracle，2016年遵循简单规则是买阿里云，一个道理



独步舞者

2018-05-17

👍 4

也是受那个时代技术的影响决定的，05年有分布式，阿里也会选择使用

作者回复: 分布式一直都有的，不是05年后才出现的



如风

2018-05-22

👍 3

很多跟我们公司类似，公司从创业再到上市，从无到有，再到每天几千万笔的流量都经历了，开始也是追求快，所有业务公用一个数据库，一出现问题，引发全部拖死，不断优化演进，业务及架构不断分离，很多东西也是买，买云服务，架构不只是简单的一个框架，涉及的东西太广了，从网络服务器，再到应用软件，数据存储等等，需要建立一套可行的运维监控体系...

展开 ▾

作者回复: 以前大部分这样，现在不一定要这样做，买云服务也是可以的，同样符合简单原则和合适原则



孙振超

2018-06-04

👍 2

各个公司的架构都是逐渐演进成当前的样子，在达到同样目的的过程中实现手段确并不完全相同，蚂蚁和阿里都进行了多地多中心部署的架构改造，但二者在诸如配置中心、跨Idc访问管控等方面都不尽相同，即使在蚂蚁内部也出现了后续实现推翻原始规划的情况。在多地多中心部署架构改造完成后，为进一步降低成本，避免大促活动中机器的浪费，又开始了弹性部署的改造，希望能够在大促高峰来临的前几个小时再临时增加服务器，等活...
展开 ▾

作者回复: 是的，有钱有人业务又复杂，可以自己弄



one day

2018-05-28

👍 2

在简单，演化，合适三原则之下我们也要考虑当时的技术水平，淘宝开始的技术是没有目前分布式成熟系统的，我们在架构设计时并不一定开始就搭建一个只是简单的系统，因为我们占在时代前进的道路上，原来是复杂的现在反而是简单，原来需要演化许多年的，现在也可能只需几天。根据业务，根据技术，根据人员，根据时间，根据可看到的未来。架构不确定，多思，就让设计的架构多撑几年

展开 ▾

作者回复: 时代不同，同样的原则，做法不同，2004年简单就是买Oracle，2018年就是买阿里云



MichaelJY

2018-05-21

👍 2

感觉所有的互联网产品都要遵循上面的原则。

首先要合适，然后简单，最后演进。

互联网时代机会稍纵即逝，所以要快速出一个基础产品，要快，要活下来先！然后随着业务的发展，逐步演进架构，当然这里面也涉及到一些技术架构的决策。

...

展开 ▾

作者回复: 传统行业的系统很多都是买的，客户不希望买了后不到1年就又要改，因此当然希望希望买一套设备顶10年，但后果就是买了大量没用的功能和特性，记得某位专家说过电信设备85%

的功能都是浪费的



SHLOMA

2018-05-20

👍 2

李老師，您好，簡單和合適原則是否是貫穿所有架構設計，而演化原則，則是指在當前架構無法通過優化手段適應當前業務量，重新設計架構？

展開 ▾

作者回復：演化原則也要在每次架構設計的時候遵守，避免過度設計



辉辉

2018-05-18

👍 2

適合互聯網：快，短，頻。是不是一定適合傳統IT行業？傳統IT行業，現在打標拼的技術部分不再是合適，簡單，演化。是各種高大尚的方案。

展開 ▾

作者回復：確實如此，很多朋友也都反饋這種情況，我的建議是把这些當做合規要求，在此基礎上應用架構設計三個原則



Bob

2018-05-17

👍 2

要是當年淘寶買了商用的PHP連接池，世界將會怎樣？PHP會真的成為宇宙第一語言嗎🤖

作者回復：很有可能，PHP是世界上最好的語言🤖

如果當時不是能夠請到sun和oracle的專家，估計就是php了



黄金的太阳

2018-05-17

👍 2

一直追劇追到現在，感覺受益匪淺，也是第一次留言，因本人能力有限，這篇文章中關於架構演進過程中各個環節希望能有更進一步關於架構圖的配套說明，例如

1.本次演进改进的点在哪里

2.为什么这样设计就能解决当时的问题

而不是直接一张图，没有任何说明，也许对于架构师大神们是很小白的问题，但咱们的...
展开 ▾

作者回复: 这些都是腾讯和手淘的公开资料，里面也提炼了演进和选型的原因，至于更多的细节，有兴趣可以直接买他们的书来看，书名《淘宝技术这十年》



追寻云的痕...

2018-05-17

👍 2

从0到1时牢记KISS原则，做大业务，等你有钱了自然请得起人了😁

作者回复: 有钱也不要浪费哦，好钢花在刀刃上😁



Dong Zhan...

2018-09-08

👍 1

我是做infra的，企业部分打交道多，主要做设计实施，售前设计接触过一点。前期拼方案的时候，演进原则不好说，简单和适用还是要的吧，为了降低竞标成本。当然最重要的是和客户感情上关系上生意上都谈拢，很多时候标书都可能是抄某个牌子来的。设计实施的时候简单适用还是很重要，演进还真不是特别快，一般能用未来3~5年就好。我在infra看，构架师和客户的沟通、说服、取信非常重要，特别不同部门不同级别；集成商将各...
展开 ▾

作者回复: 你说的这种角色类似在菊花厂叫“系统分析师”，对业务理解和行业理解要求很高，系统分析师和架构师都要求双商



syhasia

2018-09-05

👍 1

请问老师，多租户（大于1k）场景下，用户读次数远大于写，数据库需要分库分表吗？还是一个库多个schema（一个schema代表一个用户）就够了？

展开 ▾

作者回复: 分库分表主要是写请求扛不住，读请求扛不住用读写分离加缓存就够了



kai

2018-05-29

1

合适原则第一考虑，优先满足业务需求；
简单原则第二考虑，挑选简单方案快速落地验证；
演进原则第三考虑，适当预测业务发展，感觉预测不准就不预测，等真的出现问题的时候演进即可...

展开

作者回复: 拍脑袋就是：
淘宝QPS 100000，我们也要.....
docker很火，那就用它.....
业务肯定能一年翻10倍，所以系统要三高齐备.....



bigticket

2018-05-26

1

今天看到一条微博，收集了七年前appstore里微信的用户评价，骂声一片，很多是说连不上数据库的问题，而现在用微信已经很少出现bug，感觉非常能体现架构演进的原则，早期用户量少，架构设计主要考虑合适、简单两个原则，当用户量变大引起复杂度提高，从而导致无法正常运行时，倒逼架构进行演进优化。现在能支持的高峰大并发的方案很多，应该算比较成熟了，那下一个能打破现有架构设计的业务需求可能是什么？感觉又回到...

展开

作者回复: 如果是微信，现有架构基本不会演进了，因为促使微信架构演进的推动力已经没有了，一个是用户量（除非微信能全球普及，目前来看不太可能），一个是业务复杂度，这两个已经基本到头了，即使微信做新的业务，其实只是用了微信的账号和流量，不需要改变微信已有的架构，新业务自己搭建新架构即可