08 | 架构设计原则

2018-05-15 李运华

08 | 架构设计三原则

李运华

00:00 / 17:34

前面几期专栏，我跟你系统的聊了架构设计的主要目的是为了解决软件系统复杂度带来的问题，并分析了复杂度的来源。从今天开始，我会分两期讲讲架构设计的3个原则，以及架构设计原则的案例。

成为架构师是每个程序员的梦想，但并不意味着把编程做好就能够自然而然地成为一个架构师，优秀程序员和架构师之间还有一个明显的鸿沟需要跨越，这个鸿沟就是“不确定性”。

对于编程来说，本质上是不能存在不确定的，对于同样一段代码，不管是谁写的，不管什么时候执行，执行的结果应该都是确定的（注意：“确定的”并不等于“正确的”，有bug也是确定的）。而对于架构设计来说，本质上是不确定的，同样的一个系统，A公司和B公司做出来的架构可能差异很大，但最后都能正常运转；同样一个方案，A设计师认为应该这样做，B设计师认为应该那样做，看起来好像都有道理……相比编程来说，架构设计并没有像编程语言那样的语法来进行约束，更多的时候是面对多种可能性时进行选择。

可是一旦涉及“选择”，就很容易让架构师陷入两难的境地，例如：

- 是要选择业界最先进的技术，还是选择团队目前最熟悉的技术？如果选了最先进的技术后出了问题怎么办？如果选了目前最熟悉的技术，后续技术演进怎么办？
- 是要选择Google的Angular的方案来做，还是选择Facebook的React来做？Angular看起来更强大，但React看起来更灵活？
- 是要选MySQL还是MongoDB？团队对MySQL很熟悉，但是MongoDB更加适合业务场景？
- 淘宝的电商网站架构很完善，我们新做一个电商网站，是否简单地照搬淘宝就可以了？

还有很多类似的问题和困惑，关键原因在于架构设计领域并没有一套通用的规范来指导架构师进行架构设计，更多是依赖架构师的经验和直觉，因此架构设计有时候也会被看作一项比较神秘的工作。

业务千变万化，技术层出不穷，设计理念也是百花齐放，看起来似乎很难有一套通用的规范来适用所有的架构设计场景。但是在研究了架构设计的发展历史、多个公司的架构发展过程（QQ、淘宝、Facebook等）、众多的互联网公司架构设计后，我发现有几个共性的原则隐含其中，这就是：合适原则、简单原则、演化原则，架构设计时遵循这几个原则，有助于你做出最好的选择。

合适原则

合适原则宣言：“合适优于业界领先”。

优秀的技术人员都有很强的技术情结，当他们做方案或者架构时，总想不断地挑战自己，想达到甚至优于业界领先水平是其中一个典型表现，因为这样才能够展现自己的优秀，才能在年终KPI绩效总结里面骄傲地写上“设计了XX方案，达到了和Google相同的技术水平”“XX方案的性能测试结果大大优于阿里集团的YY方案”。

但现实是，大部分这样想和这样做的架构，最后可能都以失败告终！我在互联网行业见过“亿级用户平台”的失败案例，2011年的时候，某个几个人规模的业务团队，雄心勃勃的提出要做一个和腾讯QQ（那时候微信还没起来）一拼高下的“亿级用户平台”，最后结果当然是不出所料的失败了。

为什么会这样呢？

再好的梦想，也需要脚踏实地实现！这里的“脚踏实地”主要体现在下面几个方面。

1.将军难打无兵之仗

大公司的分工比较细，一个小系统可能就是一个小组负责，比如说某个通信大厂，做一个OM管理系统就有十几个人，阿里的中间件团队有几十个人，而大部分公司，整个研发团队可能就100多人，某个业务团队可能就十几个人。十几个人的团队，想做几十个人的团队的事情，而且还要做得更好，不能说绝对不可能，但难度是可想而知的。

没那么多人，却想干那么多活，是失败的第一个主要原因。

2.罗马不是一天建成的

业界领先的很多方案，其实并不是一堆天才某个时期灵机一动，然后加班加点就做出来的，而是经过几年时间的发展，不断步步完善和初具规模的。阿里中间件团队在2008年成立，发展到

现在已经有十年了。我们只知道他们抗住了多少次“双11”，做了多少优秀的系统，但经历了什么样的挑战、踩了什么样的坑，只有他们自己知道！这些挑战和踩坑，都是架构设计非常关键的促进因素，单纯靠拍脑袋或者头脑风暴，是不可能和真正实战相比的。

没有那么多积累，却想一步登天，是失败的第二个主要原因。

3.冰山下面才是关键

可能有人认为，业界领先的方案都是天才创造出来的，所以自己也要造一个业界领先的方案，以此来证明自己也是天才。确实有这样的天才，但更多的时候，业界领先的方案其实都是“逼”出来的！简单来说，“业务”发展到一定阶段，量变导致了质变，出现了新的问题，已有的方式已经不能应对这些问题，需要用一种新的方案来解决，通过创新和尝试，才有了业界领先的方案。GFS为何在Google诞生，而不是在Microsoft诞生？我认为Google有那么庞大的数据是一个主要的因素，而不是因为Google的工程师比Microsoft的工程师更加聪明。

没有那么卓越的业务场景，却幻想灵光一闪成为天才，是失败的第三个主要原因。

回到我前面提到的“亿级用户平台”失败的例子，分析一下原因。没有腾讯那么多的人（当然钱差得更多），没有QQ那样海量用户的积累，没有QQ那样的业务，这个项目失败其实是在一开始就注定的。注意这里的失败不是说系统做不出来，而是系统没有按照最初的目标来实现，上面提到的3个失败原因也全占了。

所以，真正优秀的架构都是在企业当前人力、条件、业务等各种约束下设计出来的，能够合理地将资源整合在一起并发挥出最大功效，并且能够快速落地。这也是很多BAT出来的架构师到了小公司或者创业团队反而做不出成绩的原因，因为没有了大公司的平台、资源、积累，只是生搬硬套大公司的做法，失败的概率非常高。

简单原则

简单原则宣言：“简单优于复杂”。

软件架构设计是一门技术活。所谓技术活，从历史上看，无论是瑞士的钟表，还是瓦特的蒸汽机；无论是莱特兄弟发明的飞机，还是摩托罗拉发明的手机，无一不是越来越精细、越来越复杂。因此当我们进行架构设计时，会自然而然地想把架构做精美、做复杂，这样才能体现我们的技术实力，也才能够将架构做成一件艺术品。

由于软件架构和建筑架构表面上的相似性，我们也会潜意识地将对建筑的审美观点移植到软件架构上面。我们惊叹于长城的宏伟、泰姬陵的精美、悉尼歌剧院的艺术感、迪拜帆船酒店的豪华感，因此，对于我们自己亲手打造的软件架构，我们也希望它宏伟、精美、艺术、豪华……总之就是不能寒酸、不能简单。

团队的压力有时也会有意无意地促进我们走向复杂的方向，因为大部分人在评价一个方案水平高低的时候，复杂性是其中一个重要的参考指标。例如设计一个主备方案，如果你用心跳来实现，可能大家都认为这太简单了。但如果你引入ZooKeeper来做主备决策，可能很多人会认为这个方案更加“高大上”一些，毕竟ZooKeeper使用的是ZAB协议，而ZAB协议本身就很复杂。其实，真正理解ZAB协议的人很少（我也不懂），但并不妨碍我们都知道ZAB协议很优秀。

刚才我聊的这些原因，会在潜意识层面促使初出茅庐的架构师，不自觉地追求架构的复杂性。然而，“复杂”在制造领域代表先进，在建筑领域代表领先，但在软件领域，却恰恰相反，代表的是“问题”。

软件领域的复杂性体现在两个方面：

1.结构的复杂性

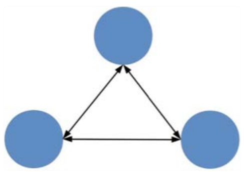
结构复杂的系统几乎毫无例外具备两个特点：

- 组成复杂系统的组件数量更多；
- 同时这些组件之间的关系也更加复杂。

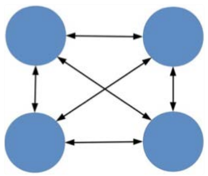
我以图形的方式来说明复杂性：

2个组件组成的系统：

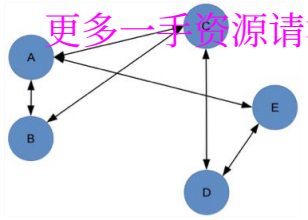
3个组件组成的系统：



4个组件组成的系统：



5个组件组成的系统：



结构上的复杂性存在的第一个问题是，**组件越多，就越有可能其中某个组件出现故障**，从而导致系统故障。这个概率可以算出来，假设组件的故障率是10%（有10%的时间不可用），那么有3个组件的系统可用性是（1-10%）×（1-10%）×（1-10%）= 72.9%，有5个组件的系统可用性是（1-10%）×（1-10%）×（1-10%）×（1-10%）×（1-10%）=59%，两者的可用性相差13%。

结构上的复杂性存在的第二个问题是，**某个组件改动，会影响关联的所有组件**，这些被影响的组件同样会继续递归影响更多的组件。还以上面图中5个组件组成的系统为例，组件A修改或者异常时，会影响组件B/C/E，D又会影响E。这个问题会影响整个系统的开发效率，因为一旦变更涉及外部系统，需要协调各方统一进行方案评估、资源协调、上线配合。

结构上的复杂性存在的第三个问题是，**定位一个复杂系统中的问题总是比简单系统更加困难**。首先是组件多，每个组件都有嫌疑，因此要逐一排查；其次组件间的关系复杂，有可能表现故障的组件并不是真正问题的根源。

2.逻辑的复杂性

意识到结构的复杂性后，我们的第一反应可能就是"降低组件数量"，毕竟组件数量越少，系统结构越简。最简单的结构当然就是整个系统只有一个组件，即系统本身，所有的功能和逻辑都在这一个组件中实现。

不幸的是，这样做是行不通的，原因在于除了结构的复杂性，还有逻辑的复杂性，即如果某个组件的逻辑太复杂，一样会带来各种问题。

逻辑复杂的组件，一个典型特征就是单个组件承担了太多的功能。以电商业务为例，常见的功能有：商品管理、商品搜索、商品展示、订单管理、用户管理、支付、发货、客服……把这些功能全部在一个组件中实现，就是典型的逻辑复杂性。

逻辑复杂几乎会导致软件工程的每个环节都有问题，假设现在淘宝将这些功能全部在单一的组件中实现，可以想象一下这个恐怖的场景：

- 系统会很庞大，可能是上百万、上千万的代码规模，"clone"一次代码要30分钟。
- 几十、上百人维护这一套代码，某个"菜鸟"不小心改了一行代码，导致整站崩溃。
- 需求像雪片般飞来，为了应对，开几十个代码分支，然后各种分支合并、各种分支覆盖。
- 产品、研发、测试、项目管理不停地开会讨论版本计划，协调资源，解决冲突。
- 版本太多，每天都要上线几十个版本，系统每隔1个小时重启一次。
- 线上运行出现故障，几十个人扑上去定位和处理，一间小黑屋都装不下所有人，整个办公区闹翻天。
- ……

不用多说，肯定谁都无法忍受这样的场景。

但是，为什么复杂的电路就意味更强大的功能，而复杂的架构却有很多问题呢？根本原因在于电路一旦设计好后进入生产，就不会再变，复杂性只是在设计时带来影响；而一个软件系统在投入使用后，后续还有源源不断的需求要实现，因此要不断地修改系统，复杂性在整个系统生命周期中都有很大影响。

功能复杂的组件，另外一个典型特征就是采用了复杂的算法。复杂算法导致的问题主要是难以理解，进而导致难以实现、难以修改，并且出了问题难以快速解决。

以ZooKeeper为例，ZooKeeper本身的功能主要就是选举，为了实现分布式下的选举，采用了ZAB协议，所以ZooKeeper功能虽然相对简单，但系统实现却比较复杂。相比之下，etcd就要简单一些，因为etcd采用的是Raft算法，相比ZAB协议，Raft算法更加容易理解，更加容易实现。

综合前面的分析，我们可以看到，无论是结构的复杂性，还是逻辑的复杂性，都会存在各种问题，所以架构设计时如果简单的方案和复杂的方案都可以满足需求，最好选择简单的方案。《UNIX编程艺术》总结的KISS（Keep It Simple, Stupid!）原则一样适应于架构设计。

演化原则

演化原则宣言："演化优于一步到位"。

软件架构从字面意思理解和建筑结构非常类似，事实上"架构"这个词就是建筑领域的专业名词，维基百科对"软件架构"的定义中有一段话描述了这种相似性：

> 从和目的、主题、材料和结构的联系上来说，软件架构可以和建筑物的架构相比拟。

例如，软件架构描述的是一个软件系统的结构，包括各个模块，以及这些模块的关系；建筑架构描述的是一幢建筑的结构，包括各个部件，以及这些部件如何有机地组成成一幢完美的建筑。

然而，字面意思上的相似性却掩盖了一个本质上的差异：建筑一旦完成（甚至一旦开建）就不可再变，而软件却需要根据业务的发展不断地变化！

- 古埃及的吉萨大金字塔，4000多年前完成的，到现在还是当初的架构。
- 中国的明长城，600多年前完成的，现在保存下来的长城还是当年的结构。
- 美国白宫，1800年建成，200年来进行了几次扩展，但整体结构并无变化，只是在旁边的空地扩建或者改造内部的布局。

对比一下，我们来看看软件架构。

Windows系统的发展历史：

更多一手资源请添加QQ/微信1182316662

（http://www.abcydia.com/upload/2016/01/4-1.jpg）

如果对比Windows 8的架构和Windows 1.0的架构，就会发现它们其实是两个不同的系统了！

Android的发展历史：

（http://www.dappworld.com/wp-content/uploads/2015/09/Android-History-Dappworld.jpg）

同样，Android 6.0和Android 1.6的差异也很大。

对于建筑来说，永恒是主题；而对于软件来说，变化才是主题。软件架构需要根据业务的发展而不断变化。设计Windows和Android的人都是顶尖的天才，即便如此，他们也不可能在1985年设计出Windows 8，不可能在2009年设计出Android 6.0。

如果没有把握“软件架构需要根据业务发展不断变化”这个本质，在做架构设计的时候就很容易陷入一个误区：试图一步到位设计一个软件架构，期望不管业务如何变化，架构都稳如磐石。

为了实现这样的目标，要么照搬业界大公司公开发表的方案；要么投入庞大的资源和时间来做各种各样的预测、分析、设计。无论哪种做法，后果都很明显：投入巨大，落地遥遥无期。更让人沮丧的是，就算跌跌撞撞拼死拼活终于落地，却发现很多预测和分析都是不靠谱的。

考虑到软件架构需要根据业务发展不断变化这个本质特点，软件架构设计其实更加类似于大自然“设计”一个生物，通过演化让生物适应环境，逐步变得更加强大：

- 首先，生物要适应当时的环境。
- 其次，生物需要不断地繁殖，将有利的基因传递下去，将不利的基因剔除或者修复。
- 第三，当环境变化时，生物要能够快速改变以适应环境变化；如果生物无法调整就被自然淘汰；新的生物会保留一部分原来被淘汰生物的基因。

软件架构设计同样是类似的过程：

- 首先，设计出来的架构要满足当时的业务需要。
- 其次，架构要不断地在实际应用过程中迭代，保留优秀的设计，修复有缺陷的设计，改正错误的设计，去掉无用的设计，使得架构逐渐完善。
- 第三，当业务发生变化时，架构要扩展、重构，甚至重写；代码也许会重写，但有价值的经验、教训、逻辑、设计等（类似生物体内的基因）却可以在新架构中延续。

架构师在进行架构设计时需要牢记这个原则，时刻提醒自己不要贪大求全，或者盲目照搬大公司的做法。应该认真分析当前业务的特点，明确业务面临的主要问题，设计合理的架构，快速落地以满足业务需要，然后在运行过程中不断完善架构，不断随着业务演化架构。

即使是大公司的团队，在设计一个新系统的架构时，也需要遵循演化的原则，而不应该认为团队人员多、资源多，不管什么系统上来就要一步到位，因为业务的发展和变化是很快的，不管多牛的团队，也不可能完美预测所有的业务发展和变化路径。

小结

今天我为你讲了面对“不确定性”时架构设计的三原则，分别是合适优于业界领先、简单优于复杂、演化优于一步到位，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧。我讲的这三条架构设计原则是否每次都要全部遵循？是否有优先级？谈谈你的理解，并说说为什么。

欢迎你把答案写到留言区，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！）

公号-Java大后端

2018-05-15

今日得到

架构即决策。架构需要面向业务需求，并在各种资源（人、财、物、时、事）约束条件下去做权衡、取舍。而决策就会存在不确定性。采用一些高屋建瓴的设计原则有助于去消除不确定，去逼近解决问题的最优解。

1 合适原则

架构无优劣，但存合适性。"汝之蜜糖，吾之砒霜"；架构一定要匹配企业所在的业务阶段；不要面向简历去设计架构，高大上的架构不等于适用；削足适履与打肿充胖都不符合合适原则；所谓合适，一定要匹配业务所处阶段，能够合理地将资源整合在一起并发挥出最大功效，并能够快速落地。

2 简单原则

"我没有时间写一封短信，所以只好写一封长信"。其实，简单比复杂更加困难。面对系统结构、业务逻辑和复杂性，我们可以编写出复杂的系统，但在软件领域，复杂代表的是"问题"。架构设计时如果简单的方案和复杂的方案都可以满足需求，最好选择简单的方案。但是，事实上，当软件系统变得太复杂后，就会有人换一个思路进行重构、升级，将它重新变得简单，这也是软件开发的大趋势。 简单原则是一个朴素且伟大的原则，Google的MapReduce系统就采用了分而治之的思想，而背后就是将复杂问题转化为简单问题的典型案例。

3 演化原则

大到人类社会、自然生物，小到一个细胞，似乎都遵循这一普世原则，软件架构也不例外。业务在发展、技术在创新、外部环境在变化，这一切都是在告诫架构师不要贪大求全，或者盲目照搬大公司的做法。应该认真分析当前业务的特点，明确业务面临的主要问题，设计合理的架构，快速落地以满足业务需要，然后在运行过程中不断完善架构，不断随着业务演化架构。怀胎需要十月，早一月或晚一月都很危险。

oddrock

2018-05-15

合适优于先进>演化优于一步到位>简单优于复杂

合适也就是适应当前需要是首位的，连当前需求都满足不了谈不到其他。
架构整体发展是要不断演进的，在这个大前提下，尽量追求简单，但也有该复杂的时候，就要复杂，比如生物从单细胞一直演化到如今，复杂是避免不了的，

作者回复

2018-05-15

标准答案����

石同享

2018-05-15

合适原则是不是可理解为：确定了一定的系统复杂度之后，能承受其包括性能，可用性，可拓展性，成本，安全方面的最小代价解(简单原则)，而演化原则是对上述系统的迭代优化。这样和之前的内容都可以联系起来了。

作者回复

2018-05-15

是的，我的思路就是这样的，三个原则是一体的，三个原则与架构设计的目的也是一脉相承的

narry

2018-05-15

个人感觉合适原则是最重要的，它决定了对简单原则和演化原则的判断，没有以合适为基础，很难判断简单是否能满足业务的需求，演化的起点在哪里

@漆~心endless

2018-05-17

架构设计三原则；
合适原则最适合；
简单原则不简单；
演化原则需推进；
如若脱离三原则；
老板生气你苦逼。

作者回复

2018-05-17

确认过眼神，你是油菜花的人！！ ❀❀❀❀❀❀❀

何磊

2018-05-15

三大原则：合适原则，简单原则，演进原则。
最重要的莫过于合适原则，如果不合适，无意欲削足适履，团队感觉难受，业务不稳定等，因此架构设计的第一原则一定是合适，合适当前的业务，合适当前的团队，合适当前的成本（时间与资本）
其二简单原则，这是一个相对原则，是在合适的基础上进行选择最简单方案，绝不能孤立，并且简单是自己业务的对比，比如：当前淘宝的架构每次迭代，他们选择一个简单方案，但他们的简单不意味着我们的简单。
关于演进原则，系统一定变化的、生长的，但是他们的起跑点肯定不同，比如大公司造的系统都是富二代，他会从微服务开始演进，十个人的小团队会从单体应用演进。
对于上面三个原则，演进原则其实我觉得考不考虑，他都存在，只要你的业务继续，不过好的架构有助于可扩展性，让后续演进更丝滑流畅，不过架构方面不能重来。

作者回复

2018-05-15

恰恰“演进原则”很多人不知道，总想一步到位过度设计

查理

2018-05-17

演进很重要，很多人都喜欢过度设计，简单的东西搞复杂。其实以后系统怎样变化很多过度的预测都不准，还不如让系统在一开始保持精简一点，根据需求慢慢演进

作者回复

2018-05-17

赞同，预测太长没有意义，也预测不准

木木

2018-05-15

合适优先级最高吧，反正一切脱离业务需求的架构设计都是耍流氓

Loy

2018-05-18

发现这三个选择放在爱情上，也完全没毛病

作者回复

2018-05-18

爱情怎么演化？����

L李亚光

2018-07-04

面对不确定性，架构师始终要做出一个选择，而三个原则遵循着解决问题的思路，提供了选择的依据。首先是合适，能够解决问题，其次是从合适中挑选简单的、能hold住的方案，最后，不要指望一个方案能解决所有问题，总会有弊端、不足，在碰到未来无法预料的情况时再做调整就是。变是永远不变的。

作者回复

2018-07-05

仅此一家，别无分店的三原则��

易燃易爆炸

2018-06-26

作者回复 架构师再牛气，首先得要有领导的信任�� 201...

哈哈哈，看到作者这句话，深感这才是第一重要的。个人感觉合适最重要，本着实事求是的态度，解决问题为导向，不求高大上，但是最各种体系都要熟悉，知道优缺点，了解具体试用场景，可能这是最好的状态吧。

作者回复

2018-06-28

不能说第一重要，但确实挺重要，第一重要的是架构师要真懂技术��

lufeng

2018-06-06

我们既要仰望星空（演进），又要脚踏实地（合适，简单），so 演进是一种sense，这个要求对业务有深入的理解和对技术趋势有独到的见解，合适和简单是两个维度，就是功能和质量要求达标的情况下尽量简单，因为越简单的东西越稳定、性能，可用性和可扩展性相当于复杂系统要好.

作者回复

2018-06-07

油菜花����先脚踏实地，再仰望星空

孙振超

2018-06-04

看完本篇文章，首先想到的就是奥卡姆剃刀原则，感觉本文中的三个原则和奥卡姆剃刀都有相关之处，只是每一个原则的侧重点不同，但根源都是为了表达最小化满足。

Liu

2018-05-22

架构设计的三原则，分别是合适优于业界领先、简单优于复杂、演化优于一步到位

星火燎原

2018-05-15

随着业务场景的演化合适也会变得不合适，演化，合适处于不断螺旋上升，也许这个不断变化中那个不变的就是“简单”

ZYCHD(子玉)

2018-05-15

合适原则，简单原则和演化原则是一体的，在合适的基础上简单，简单有利于后续系统因业务变化而产生的演化。很难说哪个原则最重要，三者都很重要。没有一尘不变的业务也就没有一尘不变的系统。

作者回复

2018-05-15

是一体的，并且这里的简单原则是指多个方案比较后挑选简单一些的，不是说方案本身一定要简单

孤星可

2018-05-15

简单意味着可读性更高 后续演化成本更低

绝恋飞鱼

2018-05-15

简单优于复杂，不要贪大求全，架构以满足当前需求慢慢迭代演进！不要一味追求好大上的技术！

anchor

2018-05-15

“软件架构设计其实更加类似于大自然”设计”一个生物，通过演化让生物适应环境，逐步变得更加强大”，这个比喻很形象。面试中经常问的一个问题是在你维护的产品中出现过哪些问题？是如何定位的？采用了什么方案后达到了什么效果？

作者回复

2018-05-15

这个面试方法叫做STAR面试法，situation，task，action，result

念一

2018-06-28

业务不断的变化，才有了架构的演变原则，而演变的过程，我们应该合适的选择，在选择的过程中，我们得遵循简单原则

健足登峰

2018-05-15

高可用，高性能，易扩展，低成本是架构师的追求。合适原则，简单原则，演化原则，是架构的尺子。

作者回复

2018-06-25

高可用这些是业务的需求，别变成架构师的追求，一旦追求就麻烦了��

Geek_59a17f

2018-06-01

合适优于业界领先、简单优于复杂、演化优于一步到位，希望对你有所帮助。

小龙

2018-06-01

写的很接地气

Geek_59a17f

2018-05-31

优秀程序员和架构师之间还有一个明显的鸿沟需要跨越，这个鸿沟就是“不确定性”。

Tony

2018-05-30

第一次了解架构三原则
初出茅庐时候有次和老大1-1面谈
直接甩给老大几个问题：为什么不用spring?为啥不用ibatis？为啥还在用SP?
看完这篇，现在回头想想老大给我的回复，姜还是老的辣.....
当时也是拍脑袋想到为何不用业界流行的框架重构我们的系统，其实没有最好的框架，只有合适的框架，只要能够简单的解决系统面临的业务复杂度，架构组优先会选择公司现有的框架
另外提到架构演绎，我的看法应该是偿还技术债务，公司目前推行的是敏捷开发，敏捷开发的价值就是快速可靠的持续交付，往往team实践时候优先考虑如何在现有框架基础上快速实现业务需求
长期以往一个组件的功能就非常复杂了，功能上容易牵一发而动全身，所以这时候不得不让team停顿下，解决现有的技术债务，从而让复杂的组件从功能上解耦

三月沙@wecatch

2018-05-29

架构的思想也非常适合不懂技术的互联网相关从业人员了来解软件体统背后的复杂性，从而对软件服务或产品有更深入的理解，有助于提高和程序员的沟通效率

小思绪

2018-05-26

合适原则>简单原则>演化原则。
技术实现业务需求时需要做各种权衡，首要任务是完成业务目标，所以并不是业界领先的就是最好的，要找到适合自己的解决方案。能方便扩展演化的架构一定会相对简单，所以个人觉得简单选择要优先考虑于演化选择。
三个原则并不完全是相互独立而是相辅相成的。

作者回复

2018-05-27

是的，三位一体

不再犹豫

2018-05-26

满足业务需求是核心原则，架构的演化也是在不满足业务或预料到将要不满足的前提下进行的，在演化设计过程中选择简单优于复杂。赶脚架构师应该有复杂问题简单化的能力，多复杂问题也要能说得清，讲得明！

作者回复

2018-05-27

架构师的核心能力：抽象，提炼，把握关键，预测未来，当然还有沟通（撕逼）��

小思绪

2018-05-26

看网友们的留言也能学到很多哈哈哈

itperson

2018-05-24

我觉得应该全部遵循 一并考虑 优先级一样
但是否应该根据预测做一定的超前架构设计 以避免业务爆发带来的架构问题 有可能没有时间来调整架构 现有架构已经扛不住了 这也是个值得思考的问题 一个度的问题

作者回复

2018-05-25

所以架构设计是技术与艺术的结合，原因就在于这个度的把握

黑客悟理

2018-05-24

合适，简单，演化，三位一体，这样去理解发现通了，确实一脉相承。

作者回复

2018-05-25

大道至简，这是从我最初的10来个原则中提炼出来的

黑客悟理

2018-05-24

演化原则，好理解，但是实际中怎么实现感觉好难啊。

作者回复

2018-05-25

一靠经验二靠直觉三看运气��把握不准就先遵循合适原则和简单原则

日光倾城

2018-05-24

我觉得合适是最重要，优先级最高的，其次才是尽量保持简单，最后根据业务变化进行优化，演化过程中依然要保持合适和简单原则

luop

2018-05-24

个人认为：架构设计就是中庸之道，全在一个“度”字，终极原则还是【合适原则】。

合适 = 可行 + 演化 + 简单；可行 > 演化 & 简单；可行是根本，演化和简单之间相互博弈。

作者回复

2018-05-24

还是很多大神有技术创新的

feifei

2018-05-22

原则不是一成不变的，原则也是如此，选择适合自己的场景最重要，我觉得优先级把是，简单，适合自己公司业务的系统，最后是演化

SHLOMA

2018-05-21

题外话，如何说服领导，合适太主观了。。。

作者回复

2018-05-22

人手不够，工期太长，成本太高，没有合适的人才……

yoummg

2018-05-20

合适，演化，简单，三个原则相辅相成。但最重要的还是合适，对等架构设计要解决的是业务复杂性的问题，业务包括功能，QPS等等，针对要解决的业务场景，制定现阶段最合适的架构设计，没毛病。理解了合适原则，演化和简单就是定义的实现过程。
这三个原则提炼的真是太精确了。膜拜。

作者回复

2018-05-20

仅此一家，别无分处，我本来列了10来个原则，最后决定精炼到最重要的3个

成功

2018-05-20

合适，简单，适应是三个维度的东西，架构设计时要根据项情况制定优先级，一切脱离生产的架构都是无效的

Jaime

2018-05-19

个人觉得简单最为重要，就像作者所说的软件是需要不断演进的。但不断演进的时候，保持系统的简单性非常重要。因为简单才更容易扩展，更容满足业务部门提出来的需求。简单里面还要保持各个组件的单一职责，这样容易定位问题，容易知道现在系统的瓶颈在哪里。而且保持了系统的简单性还可以在初创期间快速把系统落地，毕竟互联网时代有时就是在拼谁做的快，谁的系统比较稳定。技术是为了保障业务快速实现的。

作者回复

2018-05-20

先合适，有时候为了合适，不得不挑选复杂的方案

波波安

2018-05-19

具体情况具体分析
项目初期重点把握合适原则，保证项目的进度和上线。
当项目运行上线相对稳定了，则重点把握简单原则。不要把系统折腾的越来越复杂，难以维护
对于项目上线后需求变化很频繁的情况，我们我们应该着重考虑演化的原则。

苏本东

2018-05-19

目前在做一个某行业的国际站，还未上线。其中多语言文案需要其他同事翻译，于是做了一个excel表格，在微信群里飞来飞去。然后领导说同行业某某公司有自己的管理后台，说我们的做法low了，可我认为这是目前最快也是最合适的方法。后续做大了再演进呀。

作者回复

2018-05-19

看优先级，如果还有其他事情更优先就可以先用简单的方式

钰淇

2018-05-17

我们之前花了多年时间进行了一次全范围的架构升级，几乎重做了所有主力系统，这个过程中，我的体会其实跟三原则非常接近，尤其是演化原则，大公司的复杂系统永远不可能一步到位，甚至较长的工期本就会导致原有设计在最终实现时就已经落后，复杂系统的架构只能是演化迭代的，这其中即有技术因素也有业务因素，有工艺的复杂也有人的复杂；简单原则，实际上是对单一职责的追求，但是对大型系统而言，单一职责也会带来通讯的复杂，调用的复杂，大型系统清晰的逻辑分层并不容易实现，如果涉及多方协同施工，单一职责更难贯彻，尽管大家都认可，但是实现起来并不容易，合适原则非常正确，不过可惜的是，它经常不是甲方的原则。这三原则对于架构设计而言非常重要，但是保证它发挥作用的是架构师能够获得充分的保障履行职责。

作者回复

2018-05-18

架构师再牛气，首先得要有领导的信任

老王

2018-05-17

下一个版本要进行比较大的架构调整，哪些是优秀的部分需要保留，而又是什么原因促使这次调整呢？

作者回复

2018-05-17

根据业务和团队情况判断

清泉

2018-05-17

演化原则最重要，因为它最容易被人忽略。合适原则和简单原则多少还是能考虑到的。忽略了演化原则，就会导致不停的争论，不停的设计，而不能快速的落地

作者回复

2018-05-17

演化原则确实容易被人忽略，你说的这种场景我就经历过

吴建中

2018-05-17

合适对应着业务驱动，简单是基于成本，运维，扩展方面的考虑，演化是时间问题。再任何一个时间切面上，都要关注架构的合适性，简单性，而推动这个前进的动力是演化。

fiseasky

2018-05-17

老师能带我们做一个架构吗，一步一步的演变

作者回复

2018-05-17

没有业务和团队背景，是难以说清楚演进的；
如果讲业务和团队讲的太深，大部分读者又看不懂。
如果你们公司时间较长，并且业务也发生了很多变化，可以尝试分析一下自己公司技术的演进历史，也会有很多收获的

海洋

2018-05-16

我是这么排序的：合适>演进>简单

summer

2018-05-16

极好！
演化的第一条不就是适应（适合即可）么？演化加入了时间因素，拥抱变化，所以简单更能灵活应对将来的复杂。综上，演化已经包含了适合与简单。
作者很的一点是明确指出架构设计中，切勿自恋。

作者回复

2018-05-16

三位一体[illegible]

合民

2018-05-16

这个应该是看项目规模和所处阶段，我认为优先级是简单>合适>演化。对于互联网创业公司，要的是产品速度，收割用户，有时最开始我们并不能准确把握用户需求，所以简单最重要，比如django。在简单的前提下选择合适、通用、行业成熟的技术。如果是持续的、迭代的项目，下一步要考虑项目演进的方向，这个时候在明确需求的基础上在演化架构，我认为会更合适。

作者回复

2018-05-16

合适>简单>演化，因为首先还是要满足需求，适合业务

在路上

2018-05-16

我们之前用的消息队列是单点的，领导希望做成分布式的，求楼主分享一下这方面的经验。有过这方面经验的，求私聊。微信 YangYanhui8888

作者回复

2018-05-16

后面的案例就是消息队列

幸福时光

2018-05-16

软件系统唯一的不变就是变化，架构设计终极目标就是把适应未来变化的架构演进成本最小化。不过度设计，也不能没有设计。在适配当前业务场景的前提下，参考业界先进技术，找出系统设计的最优解。适应业务优先级第一，简单的架构方案不能理解为简单的系统设计，深入业务理解和技术剖析，才能输出适配的简单架构。大道至简，厚积才能薄发！

在路上

2018-05-16

我们原来对rabbitmq的用法是单点的，现在领导让研究分布式的用法，大牛或哪位同学有经验的私聊吧。微信YangYanhui8888

Mike Wang

2018-05-16

是否优于业界，简单与复杂的度，演化的进程快慢，都是个度的问题，审时度势的判断是重要的。

作者回复

2018-05-16

所以我在开篇词说架构设计是判断和选择

呵呵

2018-05-16

感觉业界很多中小公司都是一套模式，都没有多少所谓的架构选择的问题，以前是单体应用，现在整上dubbo 服务化,应用内部无外乎spring 变springboot,jdbc变hibernate,mabatis ，追新用上mongo 等

作者回复

2018-05-16

这也是选择，只是选的不好的话，可能后果也不会很严重

吴传卜

2018-05-15

产品策略→业务需求→技术架构，

合适的架构在首位
所以运用合适的技术架构，支持当前的业务帮助产品快速落地是技术与架构的第一要义

然后才是简单，迭代，最终还是要在产品与业务上落地，就好比想和做，知行是合一的，做产品和业务过程中即是在做架构，架构同时也是实现一个产品或业务

曹铮

2018-05-15

我觉得在适用性方面，如果综合各方面考量，团队的技术实力和财力都hold住的话，可以稍微拔高一点。理由不仅是为未来考虑，还有别的考虑。人的本性就是求新求变，控制好度可以激励团队尤其是年轻人的热情

作者回复

2018-05-16

可以这样做，但就是这个“稍微”难把握

wall

2018-05-15

太棒了，解决了我的疑惑！！

高歌在羊城

2018-05-15

我觉得主要是要合适，架构方案要考虑到团队素质与要解决的业务需求间的平衡。其实架构方案随着业务发展，不断演进优化，略微比业务发展速度快点就行。是否优于业界并不作为架构设计时的考量。

Even

2018-05-15

我曾经在知乎上看到过老师推荐的书，其中一本就是Unix 编程艺术，看来是要真的去拜读一下了。关于这三个原则，我觉得首先是合适原则，因为架构的目的是为了业务而设的，再就是简单原则，在合适的基础上再简化架构，第三选择是演化原则，架构的变化是因业务的变化而演化而来的。

作者回复

2018-05-15

《unix编程艺术》我看了3遍以上

A张邦卓

2018-05-15

面向业务的架构我觉得肯定是合适原则优先级最高

作者回复

2018-05-15

那如果是做中间件呢？例如让你开发一个消息队列系统。

大光头

2018-05-15

适应业务是最重要的，要不然架构就变成无水之鱼。其次就是简单原则，大道至简，变是永恒，那简单的架构维护和开发新需求也是最方便的。合适原则我觉得和简单类似，合适的也应该是简单的。

作者回复

2018-05-15

合适的不一定简单，例如蚂蚁的Oceanbase

汉斯·冯·拉特

2018-05-15

很有收获，期待后续内容

宇宙全栈

2018-05-15

Unix大法好

作者回复

2018-05-15

关键是要将unix大法应用到架构设计中，不是那么简单

Baker

2018-05-15

KISS，keep it stupid simply. 每个方法不能超过10行，每个类不能超过100行。����这就是unix为什么这么强壮的原因?

作者回复

2018-05-15

也不能以偏概全��

孙利江

2018-05-15

结构合理，满足需求，支持扩展

探索无止境

2018-05-15

"试图一步到位设计一个软件架构，期望不管业务如何变化，架构都稳如磐石"，这个误区我确实存在，下一篇文章是否就是以案例来说踩过的坑?

作者回复

2018-05-15

已经给了一个案例，你可以说说你的故事��

LONGER

2018-05-15

三大原则讲的很好，但这是从架构师的角度，如果领导有些需求违背了三大原则，领导所指的技术市面上还不成熟。该如何处理这种情况呢？是努力说服领导（显然不太可能，需求在那）？

作者回复

2018-05-15

这不是技术能解决的问题，能说服最好，不能说服就先遵循"简单原则"

winston

2018-05-15

我觉得第一条合适优于业界领先在架构时优先级可以调高，因为如果复杂的架构团队可以hold住，完全可以直接采用，在此基础上再慢慢演化。