

Apache架构师的30条设计原则！

Srinath 方志朋 8月5日

点击上方“方志朋”，选择“设为星标”
做积极的人，而不是积极废人



作者：Srinath

来源：ImportSource

本文作者叫 Srinath，是一位科学家，软件架构师，也是一名在分布式系统上工作的程序员。他是 Apache Axis2 项目的联合创始人，也是 Apache Software 基金会的成员。他是 WSO2 流处理器（wso2.com/analytics）的联席架构师。Srinath 撰写了两本关于 MapReduce 和许多技术文章的书。他获得了博士学位。来自美国印第安纳大学。

Srinath 通过不懈的努力最终总结出了30条架构原则，他主张架构师的角色应该由开发团队本身去扮演，而不是专门有个架构师团队或部门。Srinath 认为架构师应该扮演的角色是一个引导者，讨论发起

者，花草修建者，而不是定义者和构建者。Srinath 为了解决团队内部的架构纷争和抉择，制定了以下30条原则，这些原则被成员们广泛认可，也成为了新手架构师的学习途径。

基本原则

原则1： KISS(Keep it simple, stupid) 和保持每件事情都尽可能的简单。用最简单的解决方案来解决问题。

原则2： YAGNI(You aren't gonna need it)-不要去搞一些不需要的东西，需要的时候再搞吧。

(小编点评：speculative development的例子可谓俯拾皆是。程序员们对自己说：“我肯定以后会需要这项额外的功能，所以现在就提前把它实现了吧”。其实这是最考验功力的地方，不能闭门YY需要的功能，架构上又要洞察趋势。)

原则3： 爬，走，跑。换句话说就是先保证跑通，然后再优化变得更好，然后继续优化让其变得伟大。迭代着去做事情，敏捷开发的思路。对于每个功能点，创建里程碑（最大两周），然后去迭代。

(小编点评：快速反馈，一个“拍脑袋的里程碑”也好过没有里程碑...)

原则4： 创建稳定、高质量的产品的唯一方法就是自动化测试。所有的都可以自动化，当你设计时，不妨想想这一点。

(小编点评：一切自动化也要考虑ROI，比如对于特别易变的页面层...)

原则5： 时刻要想投入产出比（ROI）。就是划得来不。

原则6： 了解你的用户，然后基于此来平衡你需要做哪些事情。不要花了几个月时间做了一个devops用户界面，最后你发现那些人只喜欢命令行。此原则是原则5的一个具体表现。

原则7： 设计和测试一个功能得尽可能的独立。当你做设计时，应该想想这一条。从长远来看这能给你解决很多问题，否则你的功能只能等待系统其他所有的功能都就绪了才能测试，这显然很不好。有了这个原则，你的版本将会更加的顺畅。

原则8： 不要搞花哨的。我们都喜欢高端炫酷的设计。最后我们搞了很多功能和解决方案到我们的架构中，然后这些东西根本不会被用到。

(小编点评：老板喜欢ppt?)

功能选择

原则9：不可能预测到用户将会如何使用我们的产品。所以要拥抱MVP（Minimal Viable Product），最小可运行版本。这个观点主要思想就是你挑几个很少的使用场景，然后把它搞出来，然后发布上线让用户使用，然后基于体验 and 用户反馈再决定下一步要做什么。

原则10：尽可能的做较少的功能。当有疑问的时候，就不要去做，甚至干掉。很多功能从来不会被使用。最多留个扩展点就够了。

(小编点评：产品经理可能是听不进去的，最好采取数据度量说话...)

原则11：等到有人提出再说（除非是影响核心流程，否则就等到需要的时候再去做）。

原则12：有时候你要有勇气和客户说不。这时候你需要找到一个更好的解决方案来去解决。记住亨利福特曾经说过的：“如果我问人们他们需要什么，他们会说我需要一匹速度更快的马”。记住：你是那个专家，你要去引导和领导。要去做正确的事情，而不是流行的事情。最终用户会感谢你为他们提供了汽车。

服务端设计和并发

原则13：要知道一个server是如何运行的，从硬件到操作系统，直到编程语言。优化IO调用的数量是你通往最好架构的首选之路。

原则14：要了解Amdahl同步定律。在线程之间共享可变数据会让你的程序变慢。只在必要的时候才去使用并发的数据结构，只在必须使用同步（synchronization）的时候才去使用同步。如果要用锁，也要确保尽可能少的时间去hold住锁。如果要在加锁后做一些事情，要确保自己在锁内会做哪些事情。

原则15：如果你的设计是一个无阻塞且事件驱动的架构，那么千万不要阻塞线程或者在这些线程中做一些IO操作，如果你做了，你的系统会慢的像骡子一样。

分布式系统

原则16：无状态的系统的是可扩展的和直接的。任何时候都要考虑这一点，不要搞个不可扩展的，有状态的东东出来，这是起码的。

原则17：保证消息只被传递一次，不管失败，这很难，除非你要在客户端和服务端都做控制。试着让你的系统更轻便（使用原则18）。你要知道大部分的承诺exactly-once-delivery的系统都是做了精简的。

原则18：实现一个操作尽可能的幂等。这样的话就比较好恢复，而且你还处于至少一次传递（at least once delivery）的状态。

原则19：知道CAP理论。可扩展的事务（分布式事务）是很难的。如果可能的话，尽可能的使用补偿机制。RDBMS事务是无法扩展的。

（小编点评：new SQL了解一下。。。）

原则20：分布式一致性无法扩展，也无法进行组通信，也无法进行集群范围内的可靠通信。理想情况下最大的节点限制为8个节点。

原则21：在分布式系统中，你永远无法避免延迟和失败。

（小编点评：嗯，对，面向fail 设计。但是你的考虑你的用户，你的服务提供SLA。是真的需要7*24*365吗？）

用户体验

原则22：要了解你的用户和清楚他们的目标。他们是新手、专家还是偶然的用户？他们了解计算机科学的程度。极客喜欢扩展点，开发者喜欢示例和脚本，而普通人则喜欢UI。

原则23：最好的产品是不需要产品手册的。

原则24：当你无法在两个选择中做决定的时候，请不要直接把这个通过提供配置选项的方式传递给用户。这样只能让用户更加的发懵。如果连你这个专家都无法选择的情况下，交给一个比你了解的还少的人这样合适吗？最好的做法的是每次都找到一个可行的选项；次好的做法是自动的给出选项，第三好的做法是增加一个配置参数，然后设置一个合理的默认值。

原则25：总是要为配置设置一个合理的默认值。

原则26：设计不良的配置会造成一些困扰。应该总是为配置提供一些示例值。

原则27：配置值必须是用户能够理解和直接填写的。比如：不能让用户填写最大缓存条目的数量，而是应该让用户填写可被用于缓存的最大内存。

原则28：如果输入了未知的配置要抛出错误。永远不要悄悄的忽略。悄悄的忽略配置错误往往是找bug花了数小时的罪魁祸首。

艰难的问题

原则29：梦想着新的编程语言就会变得简单和明了，但往往要想真正掌握会很难。不要轻易的去换编程语言。

(小编点评：“技术极客”是听不进去的，不如把“个人修炼”和“项目采用”分开看待...)

原则30：复杂的拖拉拽的界面是艰难的，不要去尝试这样的效果，除非你准备好了10人年的团队。

(小编点评：我一直不太相信整体性的代码生成，比如MDA，或者拖拉拽建模代替写代码...如果说有成功的，或者是在比较狭小的领域)

最后，说一个我的感受。在一个理想的世界里，一个平台应该是有多个正交组件组成-每个组件都负责一个方面（比如，security, messaging, registry, mdidation, analytics）。好像一个系统构建成这样才是完美的。但不幸的是，现实中我们很难达到这样的状态。因为在项目初始状态时，很多事情是不确定的，你无法做到这样的独立性，现在我更倾向于在开始的时候适当的重复是必要的，当你尝试铲除他们的时候，你会发现引入了新的复杂性，分布本身就意味着复杂。有时候治愈的过程要比疾病本身更加的糟糕。

(小编点评：不同阶段采用不同的做法，照抄往往会东施效颦)

总结

作为一个架构师，应该像园丁一般，更多的是修剪花草，除草而不是去定义和构建，你应该策划而不是指挥，你应该去修剪而不是去定义，应该是讨论而不是贴标签。虽然在短期内可能会觉得也没什么，但

从长远看，指导团队找到自己的方式会带来好处。如果你稍不留神，就很容易让架构成为一个空洞的词汇。比如设计者会说他的架构是错误的，但不知道为什么是错误的。一个避免这种情况的好办法就是有一个原则列表，这个原则列表是被广泛接受的，这个列表是人们讨论问题的锚点，也是新手架构师学习的路径。

END

热门内容：

- 面试时写不出排序算法？看这篇就够了
- 网易云音乐的消息队列改造之路
- Elasticsearch性能优化实战指南
- 几种常用 JSON 库性能比较
- 11 个 Linux 终端命令，没用过的快去试试吧！
- 也许是东半球直接底气的分库分表实践了
- 史上最详细 Linux 用户与用户组知识
- [咱们从头到尾说一次 Java 垃圾回收](#)
- [ELK教程1：ElasticSearch集群的部署](#)
- [ELK教程2：Kibana的安装](#)
- [ELK教程3：logstash的部署、SpringBoot整合ELK+Filebeat](#)

方志朋的专栏

专注于Java、SpringBoot、
SpringCloud、微服务、Docker、
Kubernetes、持续集成等领域



▲长按图片识别二维码关注

喜欢

就点个["在看"](#)呗^_^