

可扩展的架构设计

关于产品架构和 Web 后端架构

关于我

- 语言控：PHP、JavaScript(Node.js)、Java、Ruby、Python、Erlang、C
- 原理控：对系统内部的细节感兴趣
- 新技术发烧友：关注开源项目
- 关注前端架构：HTML5、Backbone.js、自动化部署
- 关注应用层框架：Drupal、Ruby on Rails、Django、Express.js
- 关注数据库：MySQL、Redis、MC、MongoDB
- 其他关注点：产品设计、UX、分布式、HA、自动化运维

分享内容大纲

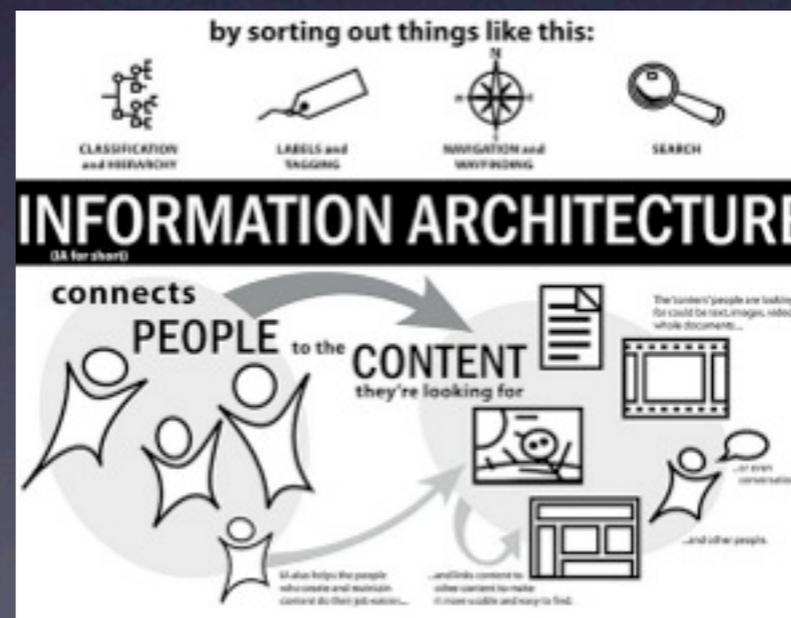
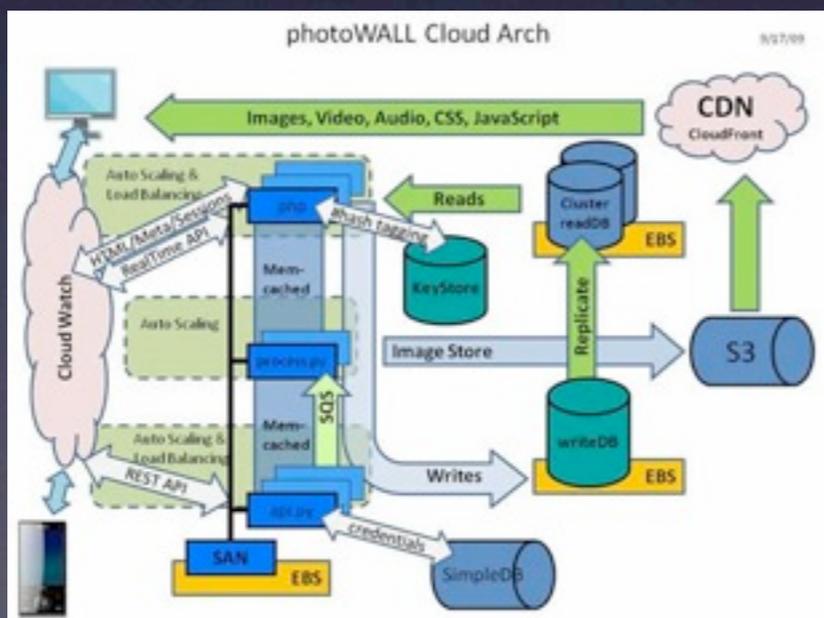
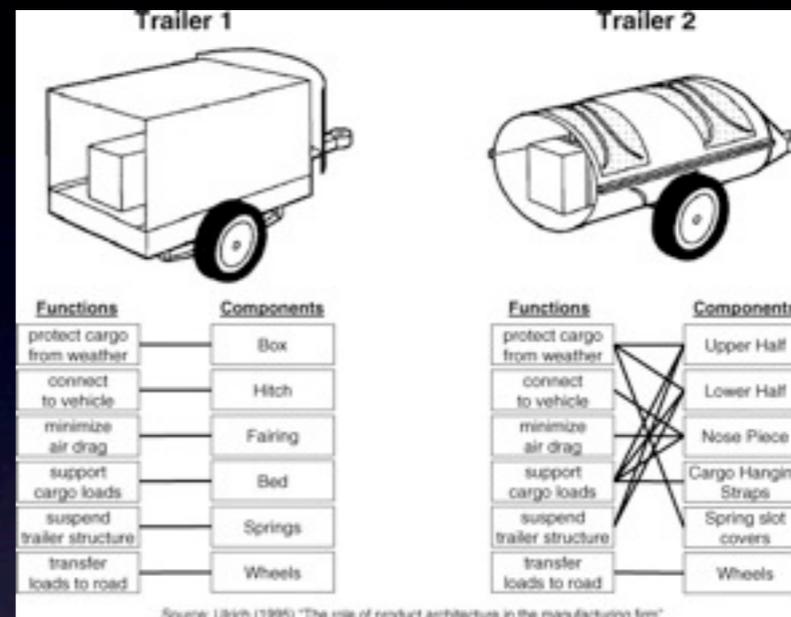
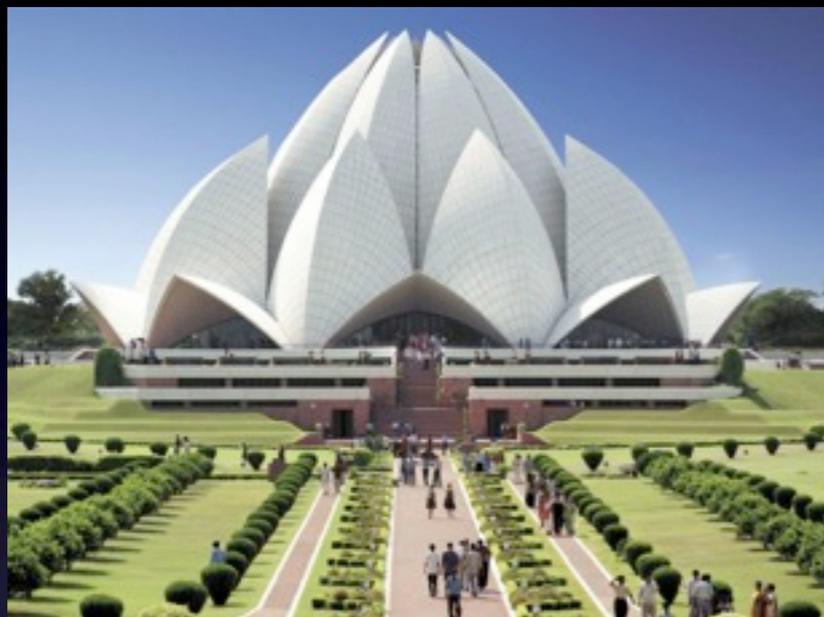
- 什么是架构，宏观和微观架构
- 架构的本质和存在的价值
- 架构实施和优化的普适方法
- 可扩展的产品架构
- 可扩展的 Web 后端架构
- 如何实现可扩展的 Web 后端架构
- 流行的 Web 架构模式和开源模块

什么是架构， 宏观和微观架构

人们常谈论到的架构

- 信息架构
- 产品架构
- Web 架构：前端架构、后端架构
- 团队架构
- 等等

人们常谈论到的架构



我理解的架构

架构是规划、选择、组织、组合、优化 (长期的 A/B testing、迭代的过程)

宏观架构：总体规划，比如，产品线的定义

微观架构：具体的实现方式

今天只讨论微观产品架构、Web 后端架构

大家都熟悉的架构活动

编程开发活动可以看做是一种微观架构设计

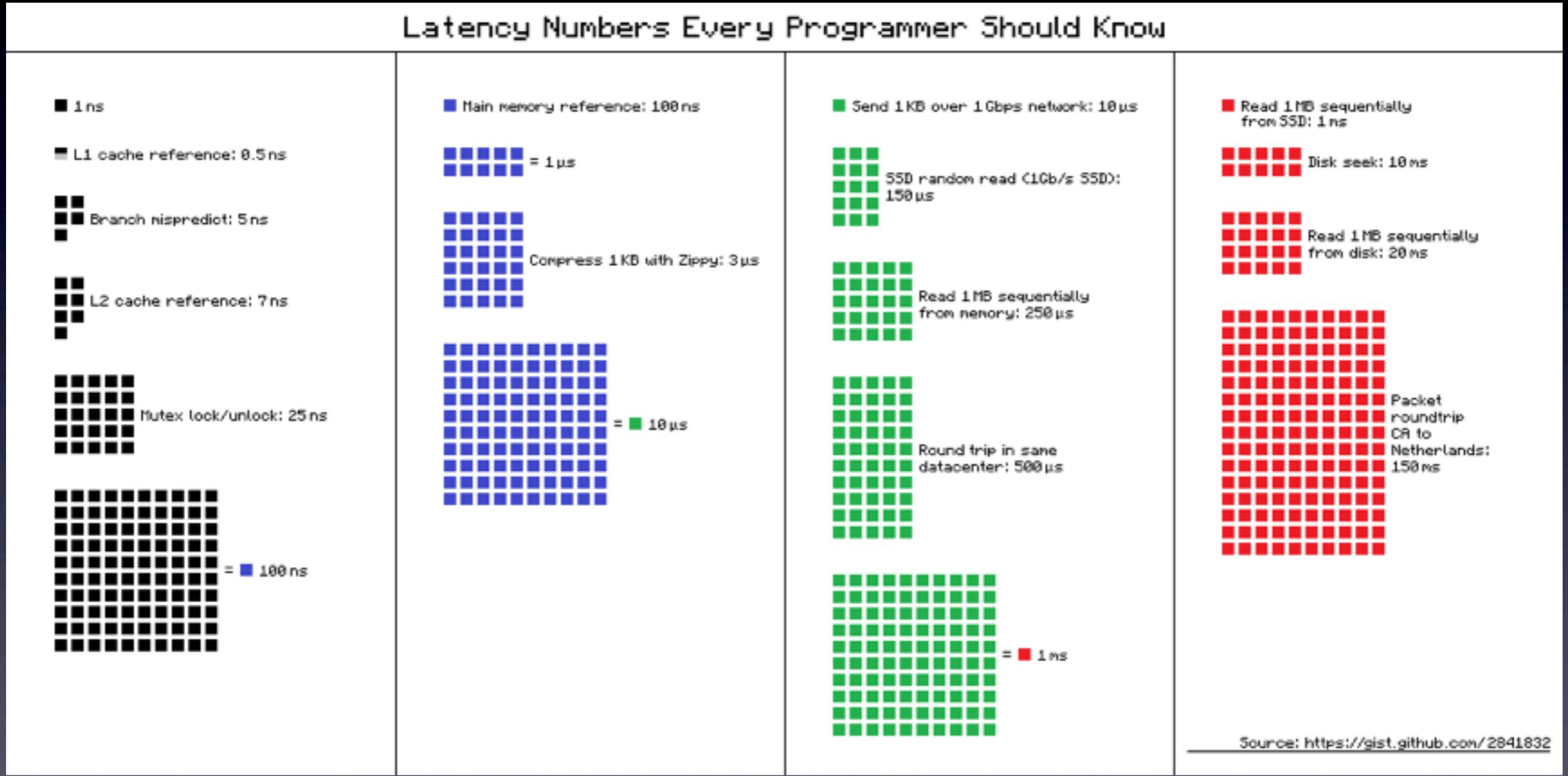
该使用内存还是文件系统还是网络, 需要权衡成本、执行性能、延时各个方面。

类、函数、对象、方法如何组织才能减少重复性工作、适应需求变化。

产品策划和设计活动也可以看做是一种架构设计

该使用什么样的布局、什么样的配色组合, 大量的信息如何合理得传递给用户, 主次的区分。

题外：每个程序员都应该知道的延迟数



架构的本质和存在的价值

架构的本质和存在价值

对于组织来说

- 1 降低长期成本(中长期的优化)
- 2 降低风险(可增减、可预期)
- 3 适应市场、业务、用户的增长

对于组织个体来说

- 1 人员活动更高效 (文档化, 摩擦更少、沟通耗费的时间更少、对业务理解更一致)
- 2 提高生产力
- 3 少加班、更多的自由时间学习新东西提高综合能力

架构实施和优化的普适方法

架构实施和优化的普适方法

架构是一个不断衡量 ROI、做选择、决策、预测、实施、测试，迭代试错的过程

可维护、并行、分层、分区、隔离、冗余、备份、重用、可测、可伸缩、可通信、监控、平衡、互补、异步、缓冲、缓存、故障检测、自恢复、重试、限流、分布式决策、虚拟化、自动化、自竞争

可扩展的产品架构

可扩展的微观产品架构

- 适应用户群体的增长(男人、女人、其他/自购、送人、代购/老人、小孩/Web、Mobile) (KISS)
- 视觉感受(页面横向宽度的限制，纵向是可扩展的) 分页是一种可扩展的例子、视觉组合 (Spaces)
- 使用体验(一致性的响应时间、一致性的查找目标的成本)
- 适应功能的持续添加和删减(要求更高) (结构尽量松耦合)
- 弹性伸缩，可增可减

可扩展的微观产品架构的一些例子和实践

- List 可增长 1-N
- Menu 可增长 1-N
- 松耦合和留空白，任何一个部件都应该是可有可无的，任何一个组件都应该有存在的原因（Cool不是原因）
- 简化意味着更强的适应性
- 衡量ROI，使用用户已经熟悉的控件和行为模式
- 激励反馈模式
- 用户群体的增长和划分
- 可以不断加减功能

可扩展的Web后端架构

可扩展的 Web 架构

弹性架构、自动根据业务量用户量增加减少机器、进程、各个组件。

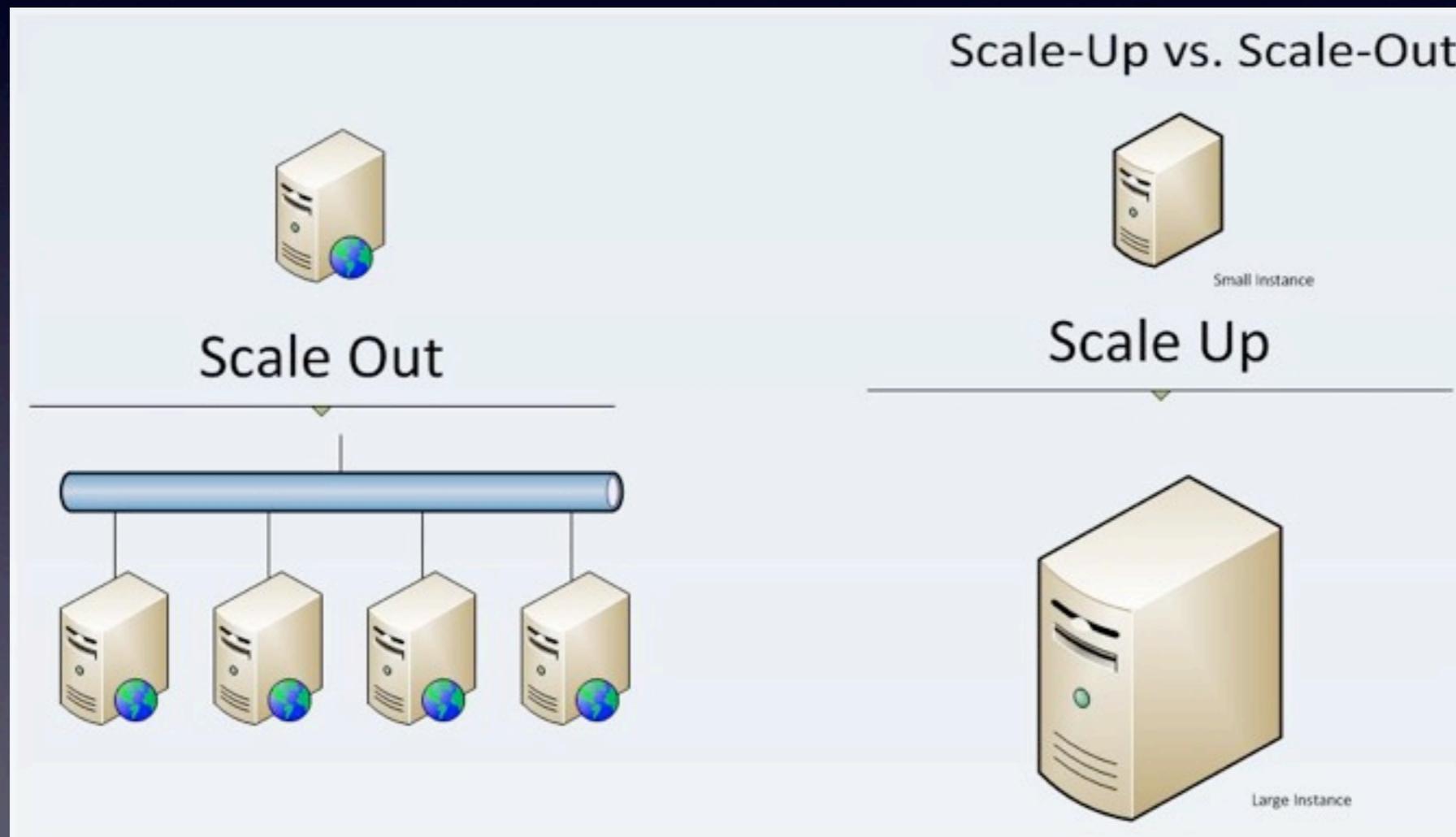
服务各司其职、不需要了解完整的业务、只需要了解接口、黑盒抽象、隔离忧虑。

- 运维很轻松: 自动化、进程当了会自动补充、机器当了会自动切换到备机、可以回头处理故障、有更充裕的时间
- 运营想促销就促销: 发红包不用考虑技术问题
- 产品的需求都能实现: (实时性的需求)、想要什么功能都能实现、局限更少
- 开发人员很 Happy: 更多的重用、少了很多重复性的工作、对自己的领域可以研究得更深入(职业发展)
- Boss 很 Happy: 网站不停机不当机 (每个部件故障或者消失、被隔离都不影响整体运作)

如何实现可扩展的 Web 后端架构

Web 架构扩展的 2 种方式

- 📌 Scaling up: Better machine, but expensive
- 📌 Scaling out: More cheap machines, open source, no licence fee



如何实现可扩展的 Web 架构 (1)

- 通过接口和协议沟通（不需要了解内部实现方式、只需要了解实现了什么）（降低通信消耗）
- 松耦合：意味着可以实现更高的性能（可以只优化瓶颈）切忌以某个组件为中心的架构，比如以数据库为中心的架构
- SOA 每个组件提供一种服务，服务直接通过接口和协议进行通信
- 服务保证 SLA 单一服务性能可测、整体支撑性能可测并且可预测（祈祷撑得住是一种高风险的做法）
- 同样服务有多个实例、之间互为冗余、互为备份（服务性能可测）
- 服务可以分级，集中优化瓶颈实现更高整体性能
- 服务可以重用、减少了工作量
- 服务更容易测试、运维、部署、分版本、迭代开发、测量性能、跟踪日志
- 可以区分实时性需求和不需要实时性的需求，异步处理非实时性需求

如何实现可扩展的 Web 架构 (2)

- 单一的服务更容易开发，不需要考虑太多牵扯的内容。开发人员的实现方式可以更深入、减少了工作量。
- 可以实现异构、使用任何语言、任何框架
- 服务之间可以隔离、当的也许只是一个不重要的服务，减少了整体损失。不重要的服务在特性情况下可以关闭。
- 服务部件可替换：性能不好、语言不爽、有了更好的开源产品
- 核心数据库承担尽量少的工作、并且做 Sharding (力气再大的牛所拉车的重量也是有限的，CPU 主频达到极限)
- 服务无状态和状态的中心化
- 分层次 (例子：Router App Cache MQ DB)
- 分区域 (例子：Sharding、业务的拆分)
- Naming service (虚拟化)
- 自动化、分布式、开源

流行的 Web 架构模式和开源模块

流行的 Web 架构模式

- Client-Server、Browser-Server
- Database-centric: (Most small website)
- ESB (Enterprise Service Bus)
- Event Driven and Async: (Most GUI、Most Proxy)
- Message Queue、Task Queue
- SOA: RESTful、SOAP、XMLRPC、Thrift
- P2P
- Share nothing (Stateless)
- Layered (分层)
- Map-Reduce
- 数据库的 Sharding、Router
- 数据库的 Master-Slave、Replica
- 大前端 (包括了服务端渲染和浏览器端渲染)

Web 架构中常见的开源模块

- ★ 分布式协同: Zookeeper
- ★ 负载均衡: LVS (Layer 4) , HAProxy (Layer 4、 7) , Nginx (Layer 7)
- ★ 虚拟化: LXC、 KVM、 Xen
- ★ HA: Keepalived、 Heartbeat
- ★ 分布式缓存: Memcache, Redis
- ★ 消息队列: 0MQ, Beanstalkd、 Gearman、 etc
- ★ 监控和自恢复工具: Supervised, Monit, Forever, etc
- ★ Storage RDMS: MySQL、 Postgres
- ★ Storage K-V DB: MySQL(handlersocket)、 Redis、 Hbase、 LevelDB、 Riak、 S3
- ★ Internal DNS: dnsmasq
- ★ 集中 Logging、 图表: Logstash、 Graphite
- ★ 压测工具: Tpcopy、 EM-Proxy、 Seige、 Tsung

Q & A