

可运维的架构

陈皓

个人简介

- › 18年工作经历，超大型分布式系统基础架构研发和设计
- › 擅长领域：金融、电子商务、云计算、大数据
- › 职业背景
 - › 阿里巴巴资深架构师（阿里云、天猫、淘宝）
 - › 亚马逊高级研发经理（AWS、全球购、商品需求预测）
 - › 汤森路透资深架构师（实时金融数据处理基础架构）
- › 目前创业MegaEase，致力于为企业提供技术架构管理产品
 - › 支撑高可用高并发高性能的分布式系统架构
 - › 为40+公司提供过软件技术服务



Weibo: @左耳朵耗子
Twitter: @haoel
Blog: <http://coolshell.cn/>

大纲

- › 运维的工作
- › 公司发展阶段
- › 典型软件架构演进
- › 运维的本质
- › 分布式软件架构
- › 运维的理念



运维的工作

运维在干什么



DevOps



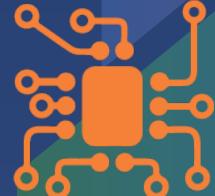
倒腾机器



故障分析



服务治理



资源管理



异地多活



各种监控



高性能高并发



高可用



配置管理

运维的几个痛点

- › 面对的是配置而不是服务
 - › 想想现在的中间件 – nginx 和 zookeeper
 - › 都不是服务化的，任何静态配置或动态配置的改变都需要通过改conf文件
- › 面对的是缺胳膊少腿以及完全不统一的架构
 - › 想想需要人肉部署、配置、启动、验证……的情况
 - › 想想各种不同格式的日志、配置、环境
- › 面对的是质量差的代码以及高速发展的业务
 - › 想想故障的定位和故障的处理
 - › 想想需要对性能和可用性的保障



企业发展和软件挑战

企业的不同的时期



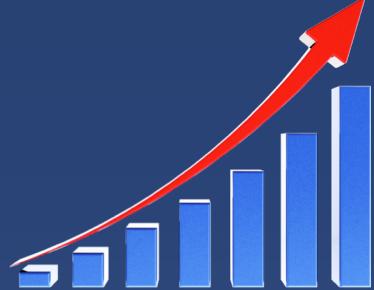
想法



实现/试错



获取用户



业务扩张



竞争

功能性的开发

从0到1，把想法实现

性能+稳定性
性能和稳定性
功能快速迭代

高级技术架构 + 生态圈 + 技术创新

性能和稳定性
软件生产流水线
大量的自动化工具

应用服务化
开放平台API
数据分析处理平台

各个阶段的运维内容

- › 公司的初期：All-in-One的系统
 - › 这个阶段基本上不需要什么运维，目前的云平台可以很好的解决。
- › 公司的中期：系统的分离
 - › 架构方面：
 - › 第一阶段：接入层的负载均衡，应用服务器的水平扩展，数据库的读写分离，缓存系统的加入
 - › 第二阶段：业务的隔离，导致接入层需要进行服务路由和服务发现，业务的分离以及服务间的相互调用，可能会引入消息中间件
 - › 运维方面：
 - › 扩容机器，系统的稳定性方面的监控（需要APM），故障的快速定位和处理（日志关联分析和报警）。
 - › 规范化运维的雏形开始出现，但重点主要集中在性能和稳定性方面。



公司的第三阶段 - 业务扩张期

- › 组织架构 - 开始走向独立的BU
 - › 公司内业务团队越来越多，机器越来越多，流程也越来越多，应用也越来越多
- › 架构方面 - 开始走向SOA
 - › 接入层开始需要负载均衡、请求路由、服务发现、权限控制、流量控制……
 - › 出现大量的中间件：
 - › 业务方面：统一用户中心，开放平台、支付财务中心、商家中心、商品中心、搜索引擎……
 - › 技术方面：高可用的消息中心件、Pub/Sub、高可用的缓存、配置中心、数据库访问层、服务治理、数据平台、……
- › 运维方面 - 开始走向云平台和高级的软件工程
 - › 基础设施（计算、存储、网络）的管理、调度和自动化……
 - › 应用的SLA管理和健康度监控分析，包括全方位的日志关联和报警，故障的快速恢复和定位……
 - › 灾备、双活、高可用、弹性伸缩、故障迁移、服务调度……
 - › 持续集成和部署……



软件研发的现状和问题

软件的复杂度持续不断地提升

- › 业务需求复杂度
- › 用户数支持复杂度
- › 部署运维规模复杂度

软件开发迭代周期和频率越来越快

- › 开发、测试周期
- › 交付周期
- › 解决问题的周期

软件的运行和质量要求的越来越高

- › 扩展性
- › 稳定性、可用性
- › 用户体验

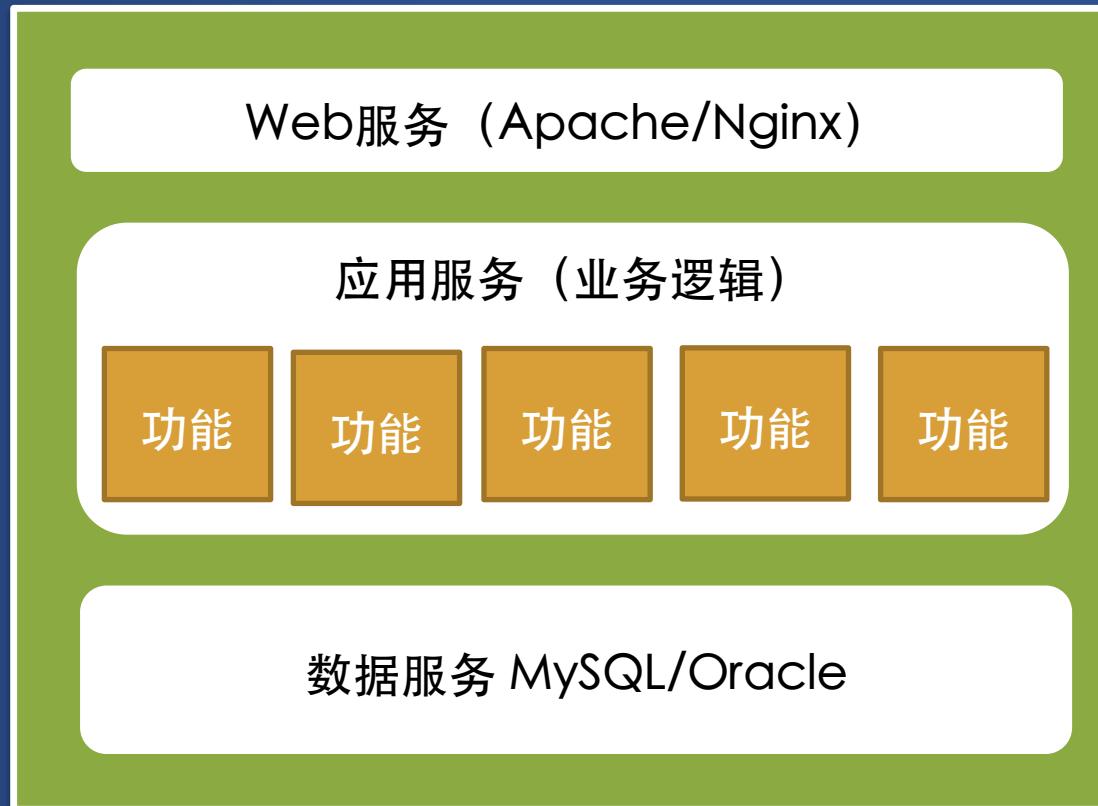


软件架构演进一瞥



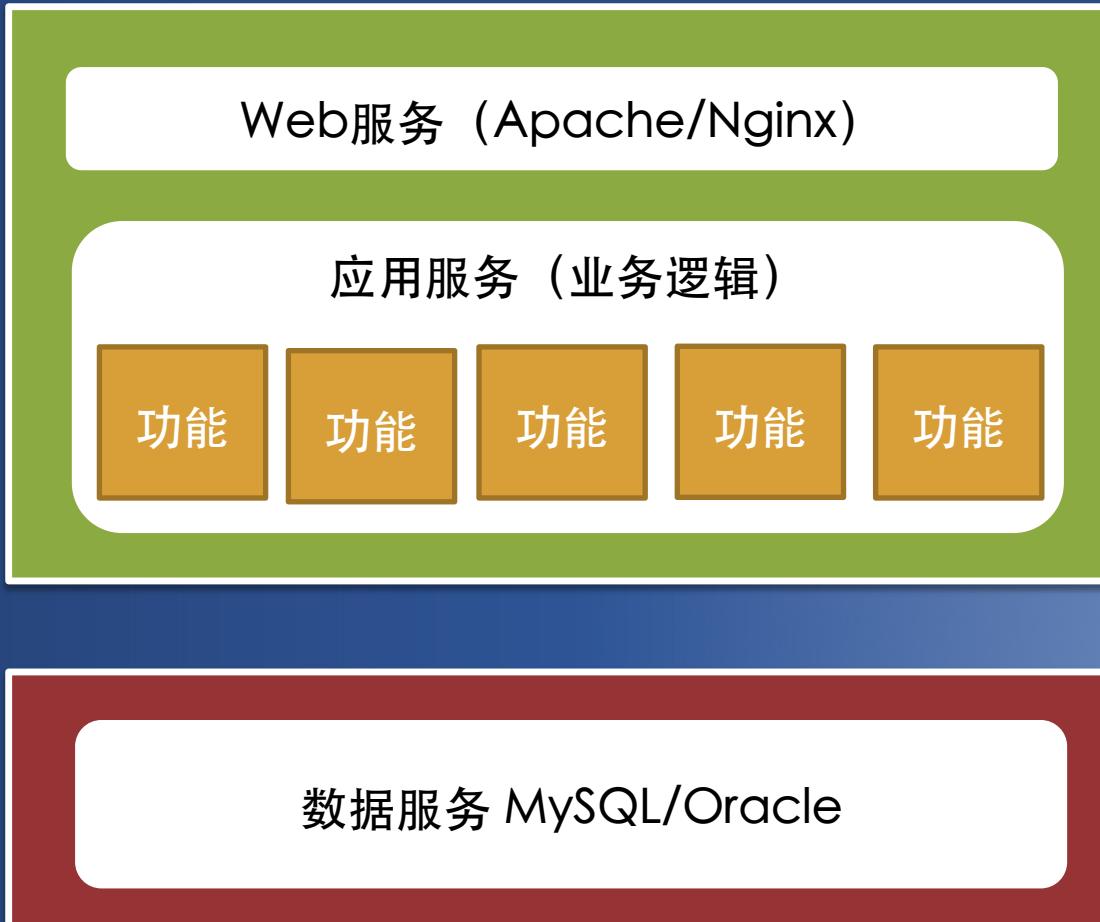
网站初始 – All in One

- › 经典的三层架构（单机，全同步）
- › 经典的MVC，ORM
- › 经典的技术栈：LAMP, RoR, Django, Spring/Hibernate ...



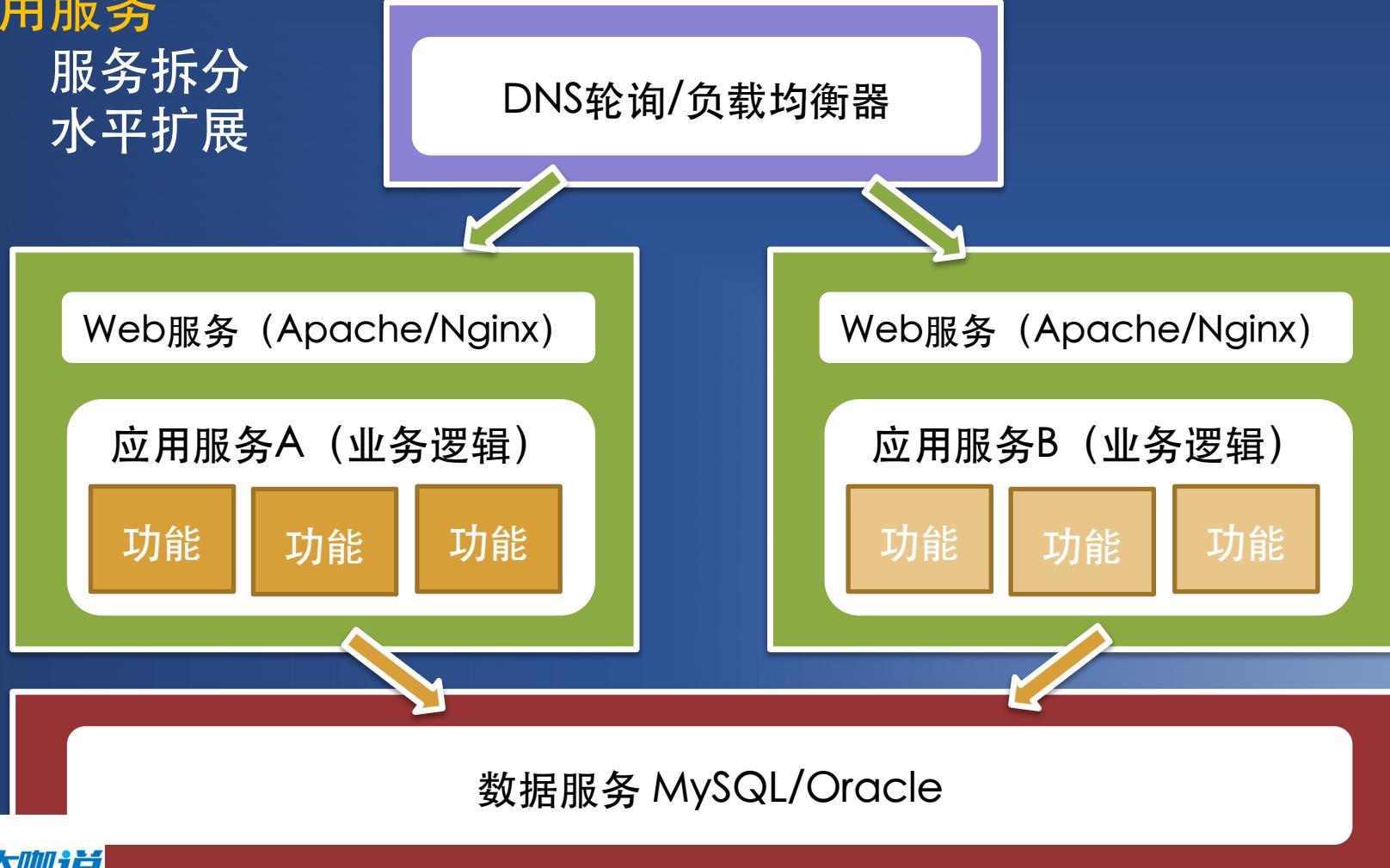
单机告警

- 数据服务和应用服务物理上分离



应用服务告警

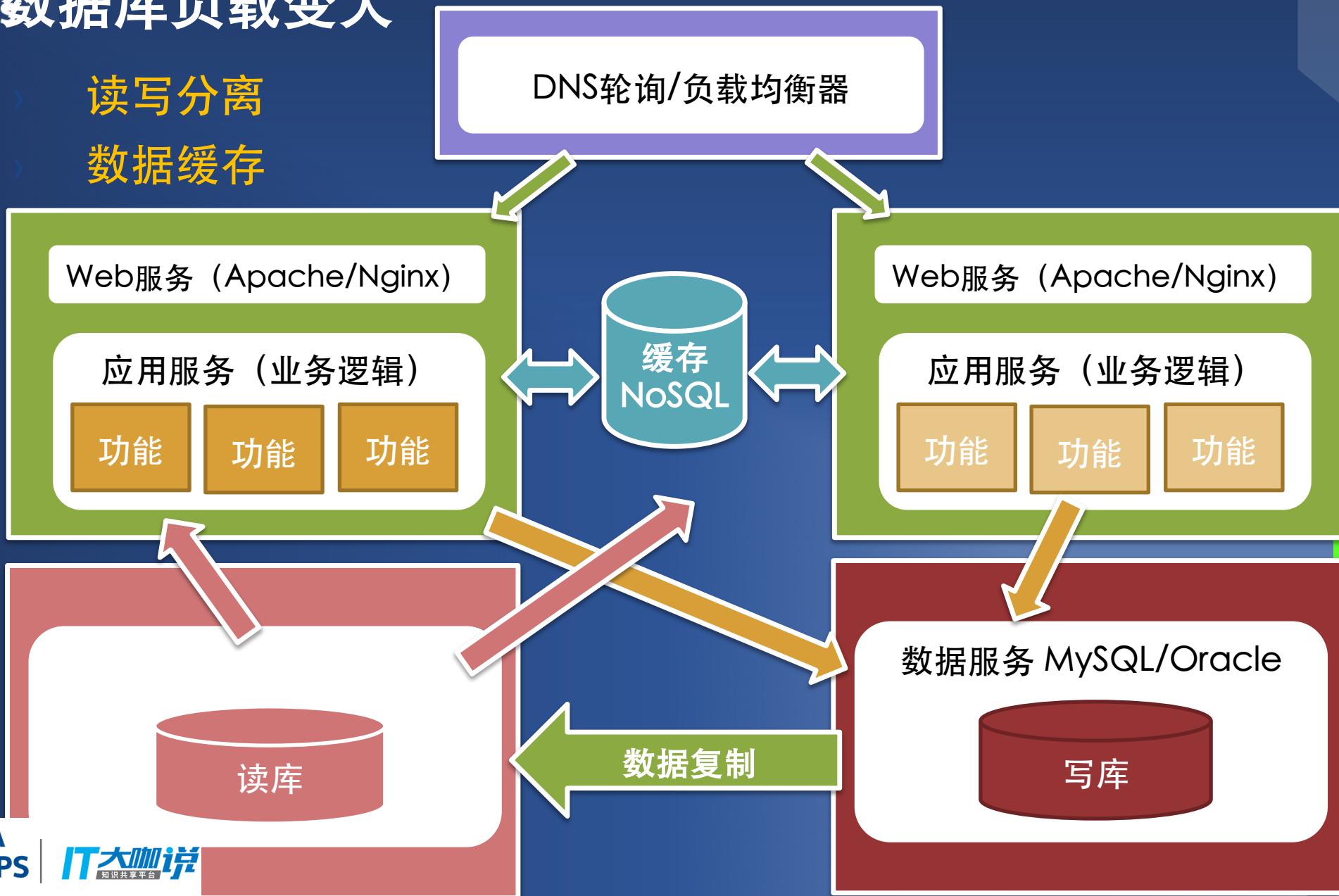
- > 应用服务
- > 服务拆分
- > 水平扩展



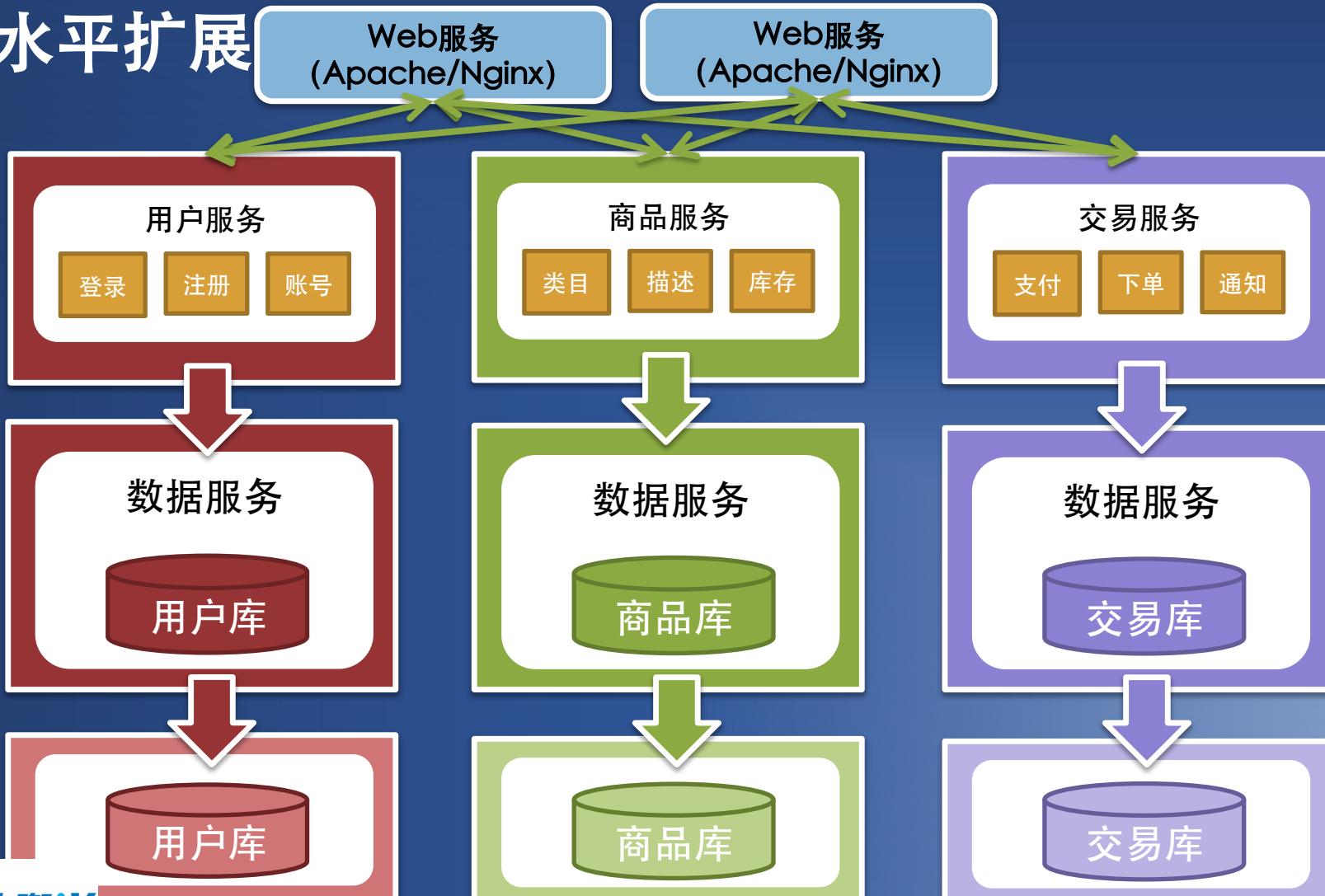
数据库负载变大

读写分离

数据缓存



继续水平扩展



运维的本质

软件工程的三个核心



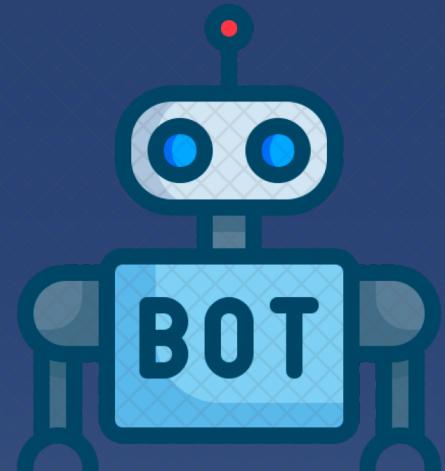
服务SLA

高可用的系统
自动化的运维



能力和资源重用

软件模块的重用
运行环境和资源的重用



过程自动化

软件生产流水线
软件运维自动化



Christopher Church
@layer_3

Simplified chart I'm using to explain the nines of availability.

| Availability | Downtime per year | Cost to implement |
|--------------|-------------------|---------------------|
| 99.9 | 8.76 hrs | \$\$\$\$\$ |
| 99.99 | 52.6 min | \$\$\$\$\$\$\$ |
| 99.999 | 5.25 min | \$\$\$\$\$\$\$\$ |
| 99.9999 | 31.5 sec | you can't afford it |

9:47 AM · 18 Jul 18

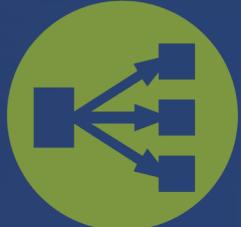


提高性能



加缓存

缓存系统
缓存分区
缓存更新
缓存命中



负载均衡

网关系统
负载均衡
服务路由
服务发现



异步调用

异步系统
消息队列
消息持久
异步事务



数据镜像

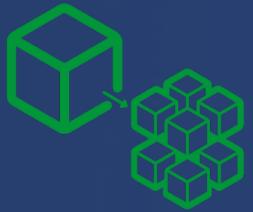
数据镜像
数据同步
读写分离
数据一致性



数据分区

数据分区
分区策略
数据访问层
数据一致性

提高稳定性



服务拆分



服务冗余



限流降级



高可用架构



高可用运维

服务治理

服务调用

服务依赖

服务隔离

服务调度

弹性伸缩

故障迁移

服务发现

限流降级

异步队列

降级控制

服务熔断

高可用架构

多租户系统

灾备多活

高可用服务

运维系统

全栈监控

DevOps

自动化运维

软件和能力重用



业务抽象



单一简化



解耦合



平台化



标准化

软件抽象

模型抽象

流程抽象

数据抽象

简化系统

KISS原则

单一职责

高内聚

去耦合

微服务化

标准协议

反转控制

PaaS平台

中件间

中台系统

API接口

系统标准

协议方法

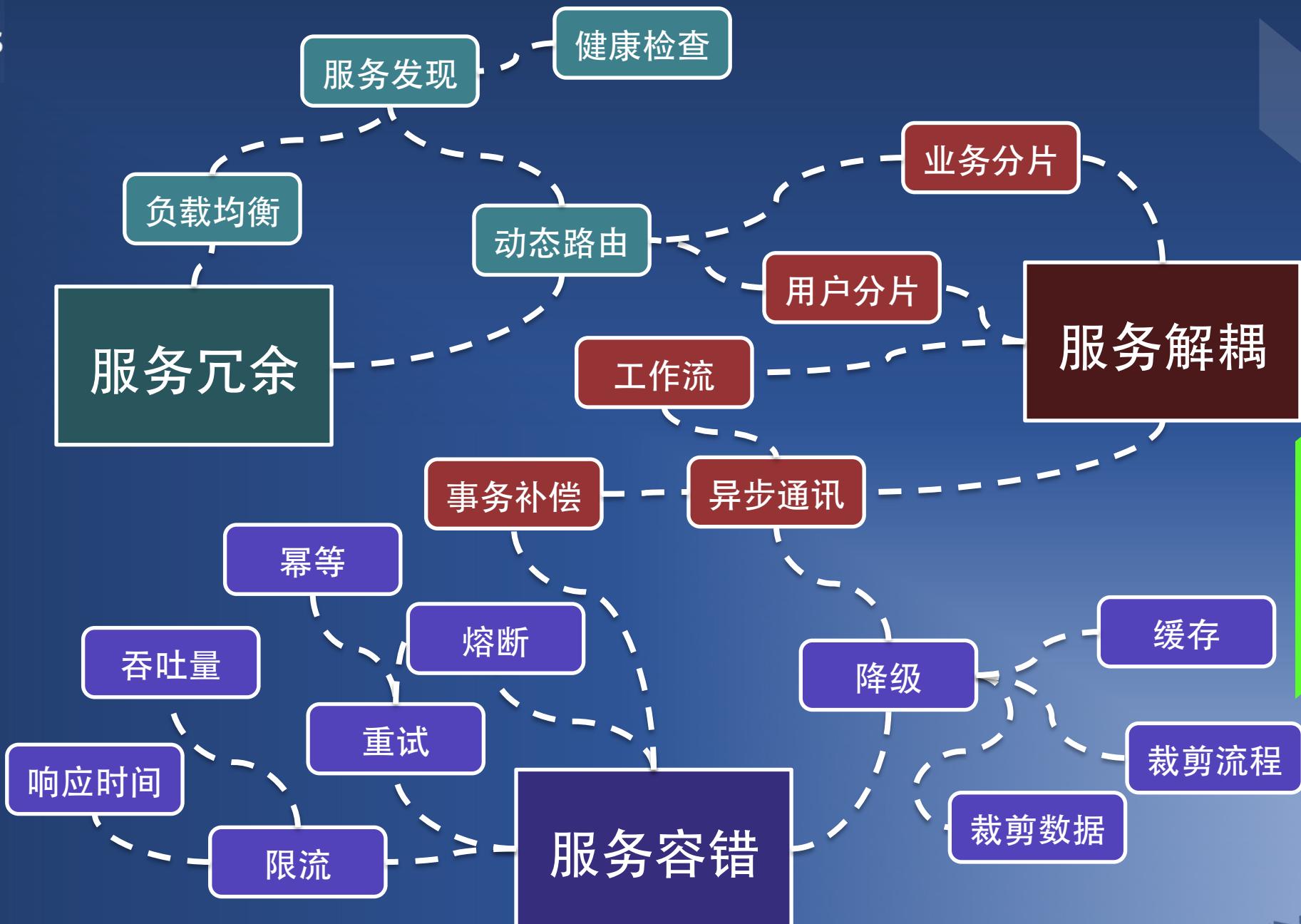
规则约定

取舍规则

我们要做多少事？

- › **高性能处理**
 - › 缓存、弹性伸缩、异步处理、数据复制……
- › **关键业务保护**
 - › 高可用、故障隔离、业务降级……
- › **流量控制**
 - › 负载均衡、服务路由、熔断、降级……
- › **整体架构监控**
 - › 三层系统监控（应用层、中间件层、基础层）
- › **DevOps**
 - › 环境构建、持续集成、持续部署
- › **架构管理**
 - › 架构版本、生命周期管理，服务管理……
- › **自动化运维**
 - › 自动伸缩、故障迁移、配置管理，状态管理……
- › **基础资源调度管理**
 - › 计算、存储、网络资源调度和管理

容错设计



如何面对如此 纷乱的技术

分布式系统的问题

| | 传统单体架构 | 分布式服务化架构 |
|-------|-------------|-------------|
| 新功能开发 | 新功能开发需要时间 | 容易开发和实现 |
| 部署 | 不经常且容易部署 | 经常发布，部署复杂 |
| 隔离性 | 故障影响范围大 | 故障影响范围小 |
| 系统性能 | 响应时间快，吞吐量小 | 响应时间慢，吞量大 |
| 系统运维 | 运维简单 | 运维复杂 |
| 新人上手 | 学习曲线大（应用逻辑） | 学习曲线大（架构逻辑） |
| 技术 | 技术单一且封闭 | 技术多样且开放 |
| 测试 | 简单 | 复杂 |
| 系统扩展性 | 扩展性很差 | 扩展性很好 |
| 系统管理 | 重点在于开发成本 | 重点在于服务治理和调度 |

分布式系统的本质

应用整体监控

- 基础层监控
OS、主机、网络…
- 中间件层监控
消息队列、缓存、数据库、应用容器、网关、RPC框架、JVM…
- 应用层监控
API请求、吞吐量、响应时间、错误码、SQL语句、调用链路、函数调用栈、业务指标…

资源/服务调度

- 计算资源调度
CPU, 内存、磁盘、网络…
- 服务调度
服务编排、服务复本、服务容量伸缩、故障服务迁移、服务生命周期管理…
- 架构调度
多租户、架构版本管理、架构部署、运行、更新、销毁管理、多租户管理、灰度发布…

状态/数据调度

- 数据可用性
多副本保存
- 数据一致性
读写一致性策略
- 数据分布式
数据索引、分片

流量调度

- 服务治理
服务发现、服务路由、服务降级、服务熔断、服务保护…
- 流量控制
负载均衡、流量分配、流量控制、异地灾备…
- 流量管理
协议转换、请求校验、数据缓存、数据计算…

状态/数据调度

| | Backups | M/S | MM | 2PC | Paxos |
|--------------|---------|-----------|----------|------------|--------|
| Consistency | Weak | | Eventual | | Strong |
| Transactions | No | Full | Local | | Full |
| Latency | | Low | | | High |
| Throughput | | High | | Low | Medium |
| Data loss | Lots | Some | | | None |
| Failover | Down | Read only | | Read/write | |

Google App Engine的co-founder Ryan Barrett
2009年的google i/o上的演讲《[Transaction Across DataCenter](#)》
(视频: <http://www.youtube.com/watch?v=srOgpXECblk>)

微服务架构



Spring Cloud & Kubernetes

| Capability | Spring Cloud with Kubernetes |
|------------------------------|--|
| DevOps Experience | Self service, multi-environment capabilities |
| Auto Scaling & Self Healing | Pod/Cluster Autoscaler, HealthIndicator, Scheduler |
| Resilience & Fault Tolerance | HealthIndicator, Hystrix, HealthCheck, Process Check |
| Distributed Tracing | Zipkin |
| Centralized Metrics | Heapster, Prometheus, Grafana |
| Centralized Logging | EFK |
| Job Management | Spring Batch, Scheduled Job |
| Load Balancing | Ribbon, Service |
| Service Discovery | Service |
| Configuration Management | Externalized Configurations, ConfigMap, Secret |
| Service Logic | Apache Camel, Spring Framework |
| Application Packaging | Spring Boot maven plugin |
| Deployment & Scheduling | Deployment strategy, A/B, Canary, Scheduler strategy |
| Process Isolation | Docker, Pods |
| Environment Management | Namespaces, Authorizations |
| Resource Management | CPU and memory limits, Namespace resource quotas |
| IaaS | GCE, Azure, CenturyLink, VMware, Openstack |

运维的理念

运维的Principles

- › 制订大的基调有利于统一思想
- › 几个我喜欢的Principles

› 简化和自动化

简单不是简陋。简单的东西易于维护，也容易执行。简化，除了使用比较简单的架构，同时也是对软件的一中抽象，让软件模块的复用率更高，复杂的架构通常来说都是不利于运维的。因此，这需要技术研发团队在设计自己架构的同时要有比较强的能力。

› 提高效率和SLA

自动化不单单是大势所趋，而且还是必需的。举个简单的例子，当用户请求量过大时，出现负载过载时，没有一个自动化（或半自动化）的东西对系统做限流和降级，那么运维人员只能是“干瞪眼”。

SLA的级别是对服务的质量提升，服务质量的提升需要的是从服务提供的API的角度来思考问题，也就是说，所有的运维内容都应该往API的服务质量上关联。

› Cloud Native

所谓Cloud-Native也就是说，如果应用层没有“云化”的架构，那么仅靠基础层的“云化”来运维，其实是不靠谱的。只有应用层和基础层的“云化”，才能达到真正高效和可靠的运维。虽然许多云提供商不断地宣传它们的服务是多么可靠，但真正的范式转变在于：正常运行时间的职责已经转移到应用程序的层面，取决于应用程序对基础设施的感知以及控制能力。对于要成为“cloud native”的应用来说，这一点是最基本的。”。

Tenets

The first AWS tenet is: ***Operational excellence and security are the most important things we do.*** Keep this tenet in mind when balancing the multiple trade-offs that you are practically promised to need to make as an AWS service owner.

Operational Tenets

[edit]

- **Operational excellence and security are the most important things we do.** You saw that one coming, right?
- **Customer retention and customer acquisition are equally important.** Focus on both simultaneously.
- **Measure twice, cut once.** COEs come at a cost. In terms of engineer efficiency and customer obsession, it is cheaper and easier to do it right the first time.
- **Prevent fires instead of fighting them.** We stay ahead of customer-impacting issues through comprehensive monitoring, full automation and relentless follow up.
- **Simple automation enables scale.** We are intolerant of repetitive manual tasks and provide robust tools to enable our services to scale.
- **Don't let perfect be the enemy of good.** We deploy solutions quickly, incrementally, and as soon as they can improve the situation today.
- **Walk before you run.** Automation is required to operate at scale, but solid process is the enabler to automation.

With this new handy context, let's begin to look at common operational areas and discuss how specifically to turn these tenets into happy customers and productive engineers.

相信技术还是相信管理？

- › 在面对很多问题的解决方案上，通常来说，有两种方式，一种是用技术的手段来解决，一种是用管理的手段来解决。用技术的手段通常会逼着我们开发更多更牛的技术设施，而使用管理手段的公司，一般会使用严格的流程和规章制度。
- › 也许，对于运维来说，流程或技术都可以解决一些问题，这关键看我们有多大的雄心，和能找到什么样的人。
- › 当然，我们并不需要把技术和管理隔裂开来，我们可以两者都用。但我还是希望能看到能对这两者有一个优先级，比如：尽可能的使用技术解决问题。
- › 老实说，这一条非常考验我们是否想成为一个技术公司。而对于运维来说，也是一样。

几个观点

- › 好的系统不是运维出来的，而是设计开发出来的
- › 运维不是管理机器和应用，而是开发和架构
- › Eat Your Own Dog Food
- › 标准化、自动化、规模化

- › 架构没搞好，做不好运维，所以——运维就是在做架构！
- › 简化和自动化是关键——简单的自动化才能做到易扩展和运维！
- › Cloud Native —— 云计算的本质就是虚拟化，虚拟化的本质就是为了调度和管理自动化！
- › 人管代码，代码管机器！
- › 相信技术，少一些人肉和流程，多一些自动化！

左耳朵耗子

全年独家专栏

你将获得

- 主流语言编程范式详解
- 分布式系统关键技术
- 性能调优攻略
- 微服务实战经验



更多细节请关注
极客时间《左耳听风》专栏



谢谢