**微服务学习笔记**

\*保持接口与实现的简单性

**原始分布式时代**

问题：远程和本地的速度差距

教训：功能能进行分布式，不意味着就应该进行分布式，不要强行追求。

**单体系统时代**

区分：大型的单体系统（差）&小型单体系统（优）

单体应用：由同一技术平台的不同组件构成的单层软件

纵向：分层架构

横向：模块

单体系统缺陷：拆分后的自治与隔离能力（可维护性）

改进：技术异构（允许程序出错，保证高可靠性）

**SOA时代**

烟囱式架构：没有互操作

微内核架构：有核心，具体业务系统以插件模块存在

事件驱动架构：子系统间建立事件队列管道

SOA：具体，系统（总结出一套自上而下的软件研发方法论），过于复杂

**微服务时代**

回归初心：透明，自由，以实践标准代替规范标准

微服务核心业务和技术特征：

围绕业务能力构建：康威定律

分散治理：开发团队有直接对服务运行质量负责的责任

通过服务来实现独立自治的组件：服务是进程外组件，远程调用

产品化思维：把软件视作一种持续改进、提升的过程

数据去中心化：同一数据实体在不同服务的视角里，抽象形态不同

强终端弱管道：需要额外通信能力，应该在自己的endpoint上解决

容错性设计：自动的机制进行快速故障检测（断路器）

演进式设计：承认服务会报废淘汰

基础设施自动化：减少工作复杂性

**后微服务时代**

虚拟化技术（边车代理模式）和容器化技术（kubernetes）

**无服务时代**

无服务：后端设施和函数

问题：按使用量计费控制消耗算力的规模，有冷启动时间

实现：使用无服务的云函数去实现微服务架构，将无服务作为技术层面的架构，将微服务视为应用层面的架构

**不可变基础设施**

**虚拟化容器**

目的：实现一次编译，到处运行

兼容性：ISA兼容，ABI兼容，环境兼容

虚拟化技术分类：指令集虚拟化，硬件抽象层虚拟化，操作系统层虚拟化，运行库层虚拟化，语言层虚拟化

没有很明白，下周学习docker怎么用应该能有更深的理解

**分布式的基石**

**分布式共识**

Paxos

Raft

分布式计算课程内容