# 系统架构演变

## 演变

随着互联网的发展，网站应用的规模不断扩大。需求的激增，带来的是技术上的压力。系统架构也因此不断的演进、升级、迭代。从单一应用，到垂直拆分，到分布式服务，到SOA，以及现在火热的微服务架构，还有在Google带领下来势汹涌的Service Mesh。我们到底是该乘坐微服务的船只驶向远方，还是偏安一隅得过且过？

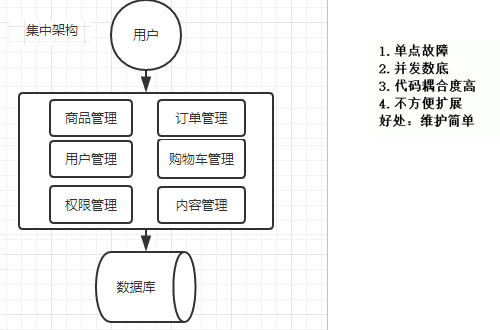
其实生活不止眼前的苟且，还有诗和远方。所以我们今天就回顾历史，看一看系统架构演变的历程；把握现在，学习现在最火的技术架构；展望未来，争取成为一名优秀的Java工程师。

# 集中式架构

## 概述

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是影响项目开发的关键。

## 架构图



## 存在的问题：

代码耦合，开发维护困难

无法针对不同模块进行针对性优化

无法水平扩展

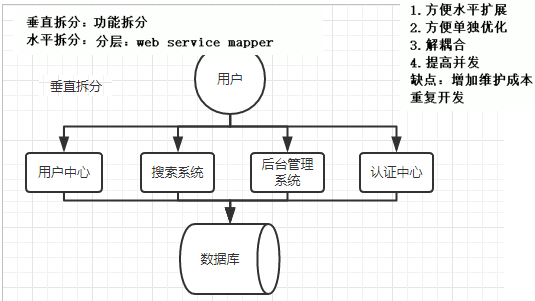
单点容错率低，并发能力差

# 垂直拆分

## 概述

当访问量逐渐增大，单一应用无法满足需求，此时为了应对更高的并发和业务需求，我们根据业务功能对系统进行拆分：

## 架构图



## 优缺点

优点：

- 系统拆分实现了流量分担，解决了并发问题

- 可以针对不同模块进行优化

- 方便水平扩展，负载均衡，容错率提高

缺点：

- 系统间相互独立，会有很多重复开发工作，影响开发效率

（eg: 比如在用户模块会操作用户user，然后在订单模块也会操作用户模块，此时userManger重复开发了，当然不止这一处）

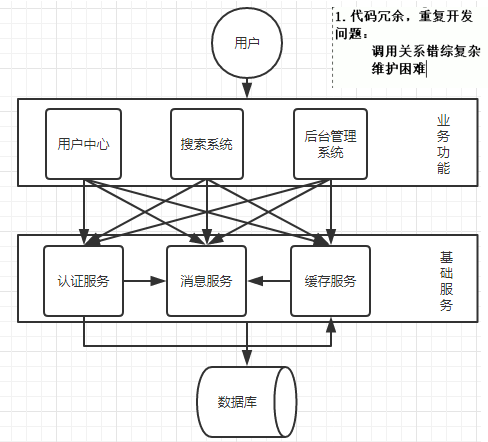
# 分布式服务

## 概述

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式调用是关键。

## 架构图

这里是将功能分类为前台服务，和后台服务，不是什么前端后台



## 优缺点

优点：

- 将基础服务进行了抽取，系统间相互调用，提高了代码复用和开发效率

缺点：

- 系统间耦合度变高，调用关系错综复杂，难以维护

# 流动计算架构（SOA）

## 概述

SOA ：面向服务的架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键

## 以前出现了什么问题？

- 服务越来越多，需要管理每个服务的地址

- 调用关系错综复杂，难以理清依赖关系

- 服务过多，服务状态难以管理，无法根据服务情况动态管理

## 服务治理要做什么？

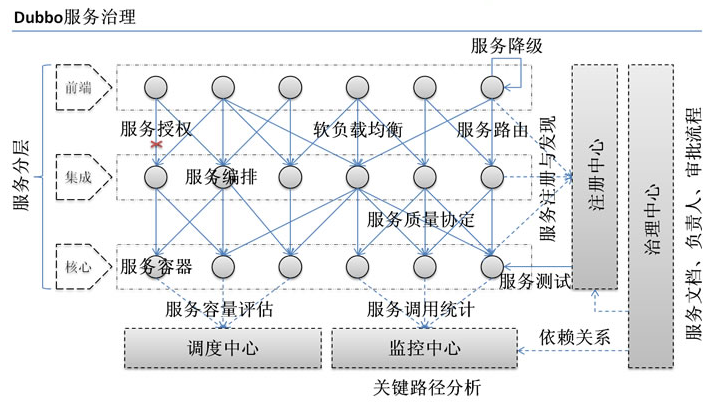
- 服务注册中心，实现服务自动注册和发现，无需人为记录服务地址

- 服务自动订阅，服务列表自动推送，服务调用透明化，无需关心依赖关系

- 动态监控服务状态监控报告，人为控制服务状态

## 架构图

这里是将功能分类为前台服务，和后台服务，不是什么前端后台



## 优缺点

缺点：

- 服务间会有依赖关系，一旦某个环节出错会影响较大

- 服务关系复杂，运维、测试部署困难，不符合DevOps思想

# 微服务

## 概念

前面说的SOA，英文翻译过来是面向服务。微服务，似乎也是服务，都是对系统进行拆分。因此两者非常容易混淆，但其实却有一些差别：

## 微服务的特点：

1.单一职责：微服务中每一个服务都对应唯一的业务能力，做到单一职责

2.微：微服务的服务拆分粒度很小，例如一个用户管理就可以作为一个服务。每个服务虽小，但“五脏俱全”。

3.面向服务：面向服务是说每个服务都要对外暴露Rest风格服务接口API。并不关心服务的技术实现，做到与平台和语言无关，也不限定用什么技术实现，只要提供Rest的接口即可。

4.自治：自治是说服务间互相独立，互不干扰

- 团队独立：每个服务都是一个独立的开发团队，人数不能过多。

- 技术独立：因为是面向服务，提供Rest接口，使用什么技术没有别人干涉

- 前后端分离：采用前后端分离开发，提供统一Rest接口，后端不用再为PC、移动段开发不同接口

- 数据库分离：每个服务都使用自己的数据源

- 部署独立，服务间虽然有调用，但要做到服务重启不影响其它服务。有利于持续集成和持续交付。每个服务都是独立的组件，可复用，可替换，降低耦合，易维护

## 架构图

