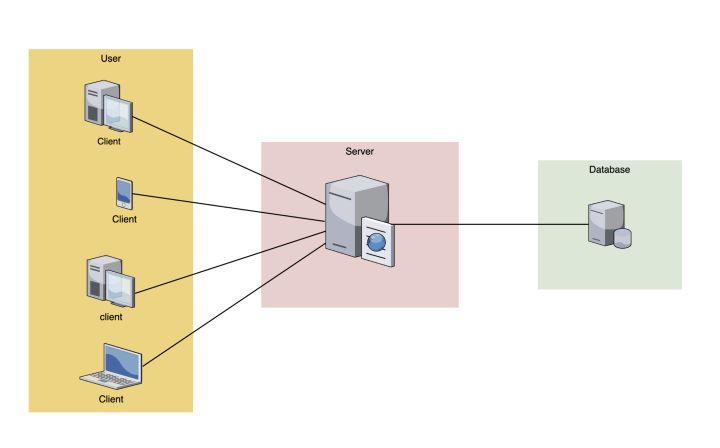
任何一个大型网站的架构都是随着用户量和数据量的不断增加不断迭代演进的结果。

在架构不断迭代演进的过程中会遇到很多问题，技术发展的本质是不断发现问题解决问题，解决问题又发现问题。

# 传统架构（单体应用）

在系统建立初期不会有特别多的客户，将所有的业务打包成一个应用放在tomacat容器中运行，与数据库共享一台服务器，这种架构一般称为单体架构。

在初期这种架构的效率非常高，根据用户的反馈可以快速迭代上线。但是随着用户数量增加，一台服务器的内存和CPU吃紧，很容易造成瓶颈。新的问题改怎么解决?

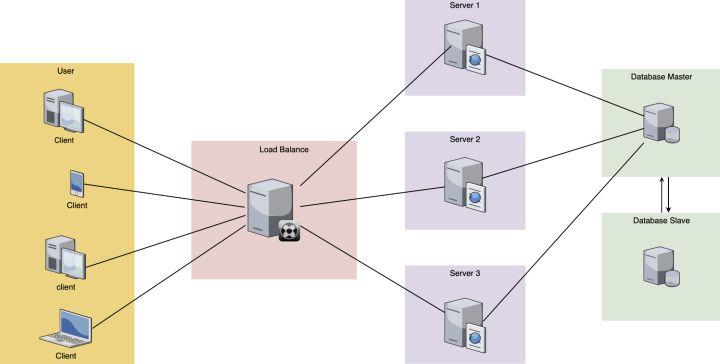


## 什么是传统架构

SSH或SSM架构,属于单点应用,把整个业务模块都放到一个项目中进行开发，分为MVC架构，拆分业务逻辑层、控制层、数据库访问层。一般只适合个人或者小团队开发;

## 传统架构的优点

1. 对于小型服务而言，传统的架构稳定且简单。
2. 后期，随着我们的用户数渐渐变多，单台服务器的压力扛不住的时候，我们就要用到负载均衡技术，增加多台服务器来抗压，后端的数据库也可以用主从的方式来增加并发量。



## 传统架构缺点：

1. 耦合度太高，一旦某个模块导致服务不可用，可能会影响整个项目;
2. 当多个开发人员开发同一个项目，可能出现代码冲突,业务不好分配。
3. 随着注册用户增加、网站应用的功能、规模在不断扩展,APP、微信、自助终端等访问渠道的增加，各种新业务，新需求不断涌入,项目工程无节制的变得臃肿庞大,项目维护成本变高。
4. 应用代码高耦合，拓展性和健壮性差。

## 单体架构-应用与数据分离

随着用户请求量增加，一台服务器的内存和CPU持续飙升，用户请求响应事件变慢。这时可以考虑将应用与数据库拆开,各自使用一台服务器。

但是其中一台服务器不能使用，用户所以的请求都会报错，应用无法使用。

## 集群部署

单实例很容易造成单点问题，比如遇到服务器故障或服务能力瓶颈。此时可以用集群来解决。

集群部署是指将应用部署在多个服务器或虚拟机上，用户通过负载均衡随机访问其中的一个实例，从而使多个实例的流量均衡，如果一个实例出现故障可以将其下线，其他实例不受影响仍然可以对外提供服务。

随着用户数量的快速增长，业务逻辑越来越复杂化，代码间耦合越来越严重。此时需要对架构重构，使用微服务架构解决。

# 分布式架构

## 什么是分布式架构

分布式架构是在传统架构上演变来的,将传统的项目以项目模块进行拆分成N多个子项目。  
  比如会员项目，订单项目，支付项目，每个项目中都有自己独立的数据库，独立的redis。

# SOA架构

SOA架构(Service Oriented Architecture)是基于分布式架构演变过来的，SOA架构代表面向服务架构（服务化），可理解为面向业务逻辑层开发，将共同的业务逻辑抽取出来形成一个服务，提供给其他服务接口进行调用，服务与服务之间使用rpc远程技术进行通信;

# 微服务架构

微服务架构是基于SOA架构演变而来的，比SOA架构上粒度更加精细，让专业的人做专业的事情（更加专注），目的是为了提供效率; 每个服务之间是互补影响的，每个服务必须独立部署（独立的数据库，独立的redis）,微服务架构更加体现轻量级，采用resful风格提供APi（即采用http+json格式），更加轻巧，更加适合于互联网公司敏捷开发（快速迭代产品）；

相关博客

<https://blog.csdn.net/mmgithub123/article/details/124744282>

# 微服务架构中的问题

## 服务与服务之间如何交换

### 问题

1.一个应用拆分为多个独立微服务。多个微服务可以使用Docker容器化进行多实例部署。

架构演化到这里遇到了一个难题，如果要查询用户所有的订单，用户服务可能会依赖订单服务，用户服务如何与订单服务交互呢？订单服务有多个实例该访问哪一个？

### 解决方案

1. **服务地址硬编码**
2. 服务的地址写死在数据库或者配置文件，通过访问DNS域名进行寻址路由。
3. 如果遇到大促活动需要对服务实例扩容，大促完需要对服务实例进行下线，运维人员要做大量的手工操作，非常容易误操作。

**2.服务注册和发现**

在单体架构中一个应用程序就是一个服务包，包内模块通过函数方式相互调用，模型足够简单，没有服务注册和发现一说。

在微服务中，将一个应用拆分为多个微服务，微服务部署在不同的服务器、不同的容器下，微服务间要相互调用。服务注册和发现成为了一种很好的解决方案。

相关博客：

<https://blog.51cto.com/u_15659694/5755282>