咕泡学院 VIP 课：分布式架构的演进过程

课程目标

1. 了解分布式架构中的相关概念
2. 初始分布式架构及意义
3. 分布式架构的发展过程和历史
4. 分布式架构的演进过程
5. 构建分布式架构最重要的因素

课程笔记

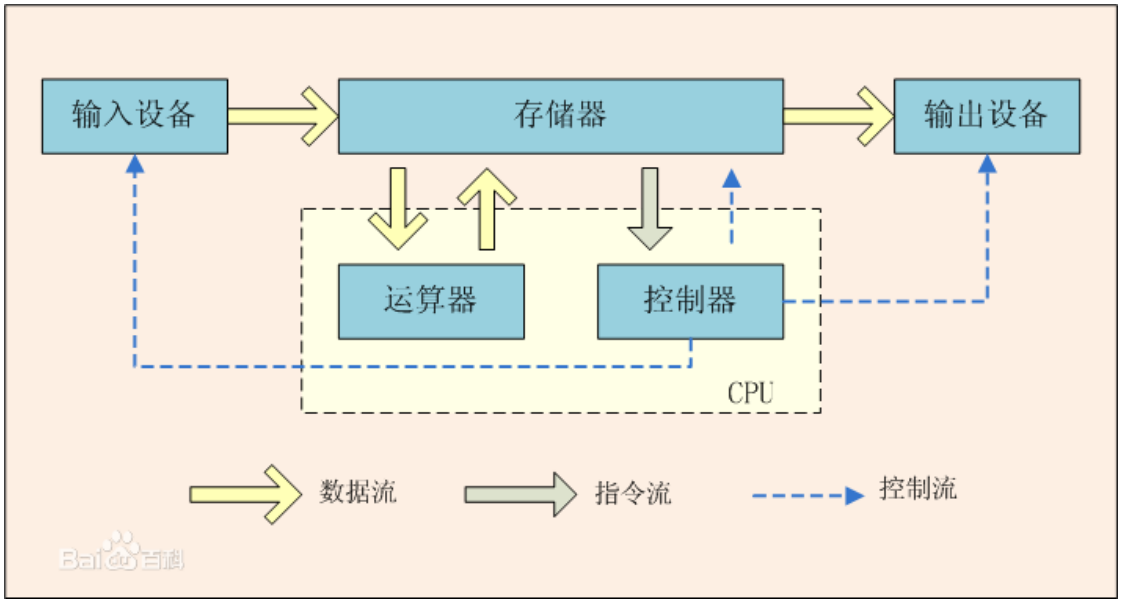
分布式架构的发展历史

1946 年情人节(2.14)，世界上第一台电子数字计算机诞生在美国宾夕法尼亚大学大学，它的名字是：ENIAC；这台计算机占

地 170 平米、重达 30 吨，每秒可进行 5000 次加法运算。

第一台电子计算机诞生以后，意味着一个日新月异的 IT 时代的到来。一方面单台计算机的性能每年都在提升：从最早的 8 位 CPU 到现在的 64 位 CPU；从早期的 MB 级内存到现在的 GB 级别内存;从慢速的机械存储到现在的固态 SSD 硬盘存储。

tips：冯诺依曼模型



ENIAC 之后，电子计算机便进入了 IBM 主导的大型机时代，IBM 大型机之父吉恩.阿姆达尔被认为是有史以来最伟大的计算机设计师之一。1964 年 4 月 7 日，在阿姆达尔的带领下，历时三年，耗费 50 亿美元，第一台 IBM 大型机 SYSTEM/360 诞生。这使得 IBM 在 20 实际 50~60 年代统治整个大型计算机工业，奠定了 IBM 计算机帝国的江山。

2.1 IBM 大型机曾支撑美国航天登月计划

2.2 IBM 主机一直服务于金融等核心行业的关键领域

由于高可靠性和超强的计算能力，即便在 X86 和云计算飞速发展的情况下,IBM 的大型机依然牢牢占据着一定的高端市场份额

20 世纪 80 年代，在大型机霸主的时代，计算机架构同时向两个方向发展

3.1 以 CISC (微处理器执行的计算机语言指令集) CPU 为架构的价格便宜的面向个人的 PC

3.2 以 RISC (精简指令集计算机) CPU 为架构的价格昂贵的面向企业的小型 UNIX 服务器

分布式架构发展的里程碑

大型主机的出现。凭借着大型机超强的计算和 I/O 处理能力、稳定性、安全性等，在很长一段时间内，大型机引领了计算机行业及商业计算领域的发展。而集中式的计算机系统架构也成为了主流。随着计算机的发展，这种架构越来越难以适应人们的需求，比如说

1. 由于大型主机的复杂性，导致培养一个能够熟练运维大型主机的人的成本很高
2. 大型主机很贵，一般只有土豪(政府、金融、电信)才能用得起
3. 单点问题，一台大型主机出现故障，那么整个系统将处于不可用状态。而对于大型机的使用群体来说，这种不可用导致的损失是非常大的

4. 科技在进步，技术在进步。PC 机性能不断提升，很多企业放弃大型机改用小型机及普通 PC 来搭建系统架构

阿里巴巴在 2009 年发起了一项"去 IOE"运动

IOE 指的是 IBM 小型机、Oracle 数据库、EMC 的高端存储
2009 年"去 IOE"战略透露，到 2013 年 5 月 17 日最后一台 IBM 小型机在支付宝下线。

为什么要去 IOE?

阿里巴巴过去一直采用的是 Oracle 数据库，并利用小型机和高端存储设备提供高性能的数据处理和存储服务。随着业务的不断发展，数据量和业务量呈爆发性增长，传统的集中式 Oracle 数据库架构在扩展性方面遭遇瓶颈。
传统的商业数据库软件(Oracle,DB2)，多以集中式架构为主，这些传统数据库软件的最大特点就是将所有的数据都集中在一个数据库中，依靠大型高端设备来提供高处理能力和扩展性。集中式数据库的扩展性主要采用向上扩展(Scale up)的方式，通过增加 CPU，内存，磁盘等方式提高处理能力。这种集中式数据库的架构，使得数据库成为了整个系统的瓶颈，已经越来越不适应海量数据对计算能力的巨大需求

分布式系统的意义

1. 升级单机处理能力的性价比越来越低

   单机的处理能力主要依靠 CPU、内存、磁盘。通过更换硬件做垂直扩展的方式来提升性能，成本会越来越高。
2. 单机处理能力存在瓶颈

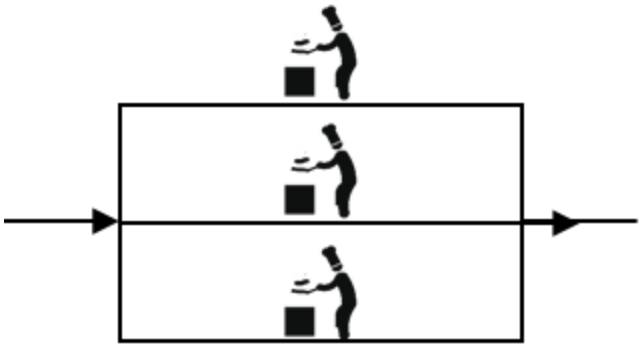
   单机处理能力存在瓶颈，CPU、内存都会有自己的性能瓶颈，也就是说就算你是土豪不惜成本去提升硬件，但是硬件的发展速度和性能是有限制的。
3. 稳定性和可用性这两个指标很难达到

   单机系统存在可用性和稳定性的问题，这两个指标又是我们必须要去解决的

分布式架构的常见概念

集群

小饭店原来只有一个厨师，切菜洗菜备料炒菜全干。后来客人多了，厨房一个厨师忙不过来，又请了个厨师，两个厨师都能炒一样的菜，这两个厨师的关系是集群



分布式

为了让厨师专心炒菜，把菜做到极致，又请了个配菜师负责切菜，备菜，备料，厨师和配菜师的关系是分布式，一个配菜师也忙不过来了，又请了个配菜师，两个配菜师关系是集群

节点

节点是指一个可以独立按照分布式协议完成一组逻辑的程序个体。在具体的项目中，一个节点表示的是一个操作系统上的进程。

副本机制

副本(replica/copy)指在分布式系统中为数据或服务提供的冗余。
数据副本指在不同的节点上持久化同一份数据，当出现某一个节点的数据丢失时，可以从副本上读取到数据。数据副本是分布式系统中解决数据丢失问题的唯一手段。
服务副本表示多个节点提供相同的服务，通过主从关系来实现服务的高可用方案

中间件

中间件位于操作系统提供的服务之外，又不属于应用，他是位于应用和系统层之间为开发者方便的处理通信、输入输出的一类软件，能够让用户关心自己应用的部分。

架构的发展过程

一个成熟的大型网站系统架构并不是一开始就设计的非常完美，也不是一开始就具备高性能、高可用、安全性等特性，而是随着用户量的增加、业务功能的扩展逐步完善演变过来的。在这个过程中，开发模式、技术架构等都会发生非常大的变化。而针对不同业务特征的系统，会有各自的侧重点，比如像淘宝这类的网站，要解决的是海量商品搜索、下单、支付等问题；

像腾讯，要解决的是数亿级别用户的实时消息传输；百度所要解决的是海量数据的搜索。每一个种类的业务都有自己不同的系统架构。我们简单模拟一个架构演变过程。

我们以 javaweb 为例，来搭建一个简单的电商系统，从这个系统中来看系统的演变历史；要注意的是，接下来的演示模型，关注的是数据量、访问量提升，网站结构发生的变化， 而不是具体关注业务功能点。其次，这个过程是为了让大家更好的了解网站演进过程中的一些问题和应对策略。

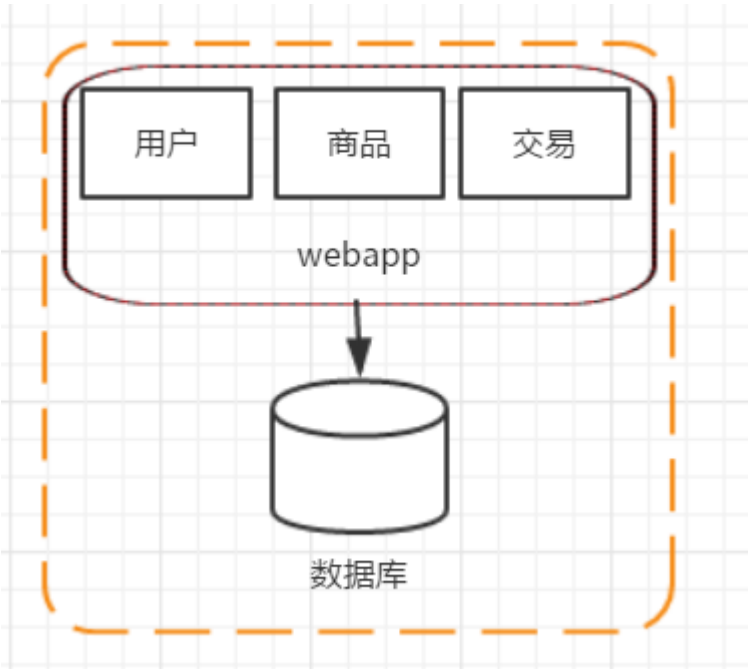
假如我们系统具备以下功能：

用户模块：用户注册和管理

商品模块：商品展示和管理

交易模块：创建交易及支付结算

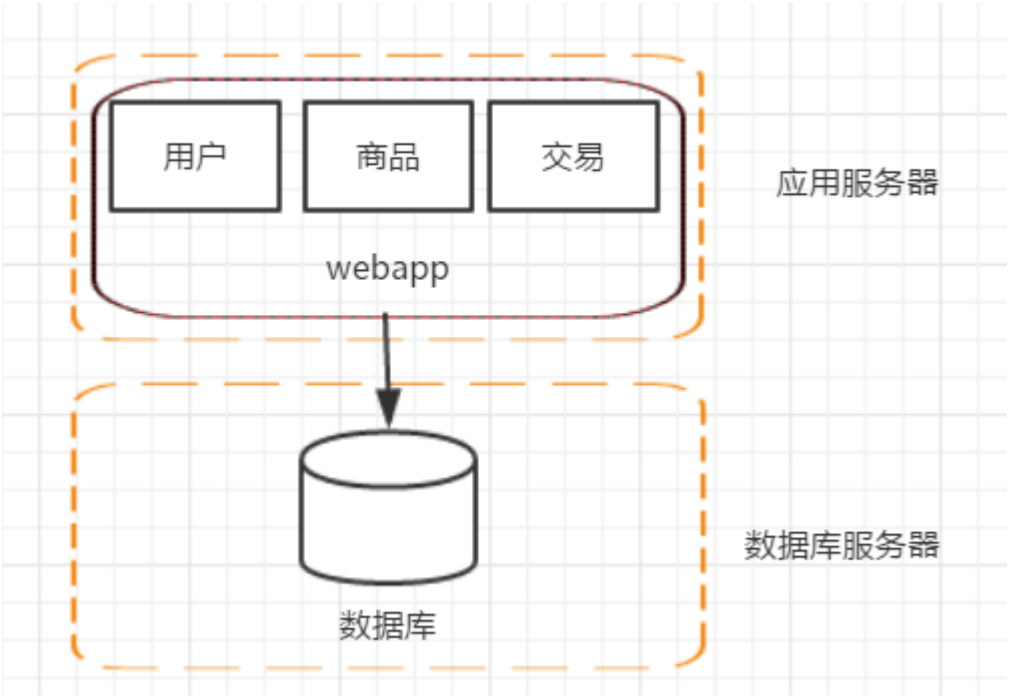
阶段一，单应用架构



网站的初期也可以认为是互联网发展的早起，我们经常会在单机上跑我们所有的程序和软件。
把所有软件和应用都部署在一台机器上，这样就完成一个简单系统的搭建，这个时候的讲究的是效率

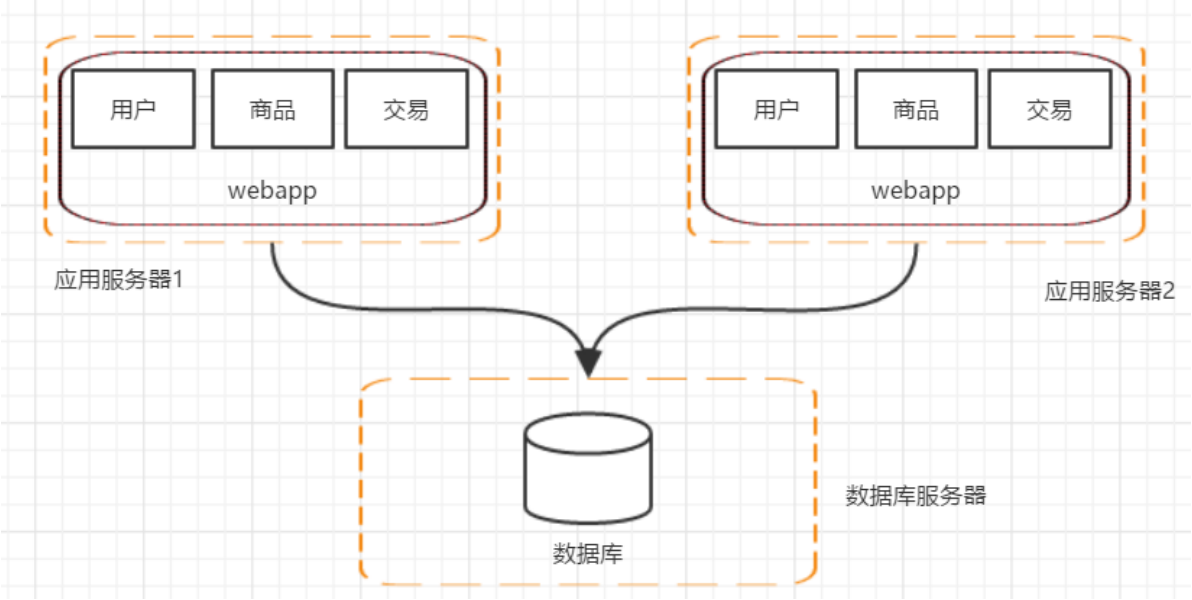
阶段二，应用服务器和数据库服务器分离

随着网站的上线，访问量逐步上升，服务器的负载慢慢提高，在服务器还没有超载的时候，我们应该做好规划，提升网站的负载能力。假如代码层面的优化已经没办法继续提高，在不提高单台机器的性能，增加机器是一个比较好的方式，投入产出比非常高。这个阶段增加机器的主要目的是讲 web 服务器和数据库服务器拆分，这样不仅提高了单机的负载能力，也提高了容灾能力



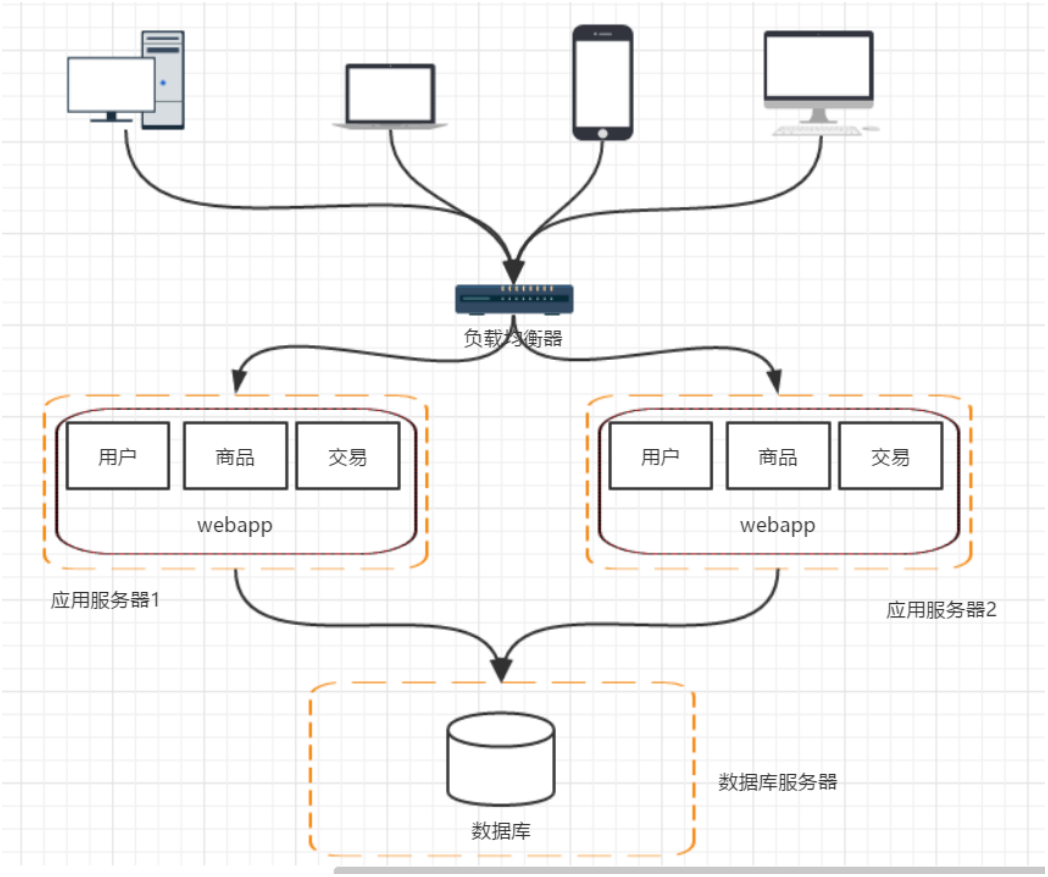
阶段三，应用服务器集群-应用服务器负载告警，如何让应用服务器走向集群

随着访问量的继续增加，单台应用服务器已经无法满足需求。在假设数据库服务器还没有遇到性能问题的时候，我们可以增加应用服务器，通过应用服务器集群将用户请求分流到各个服务器中，从而继续提升负载能力。此时多台应用服务器之间没有直接的交互，他们都是依赖数据库各自对外提供服务



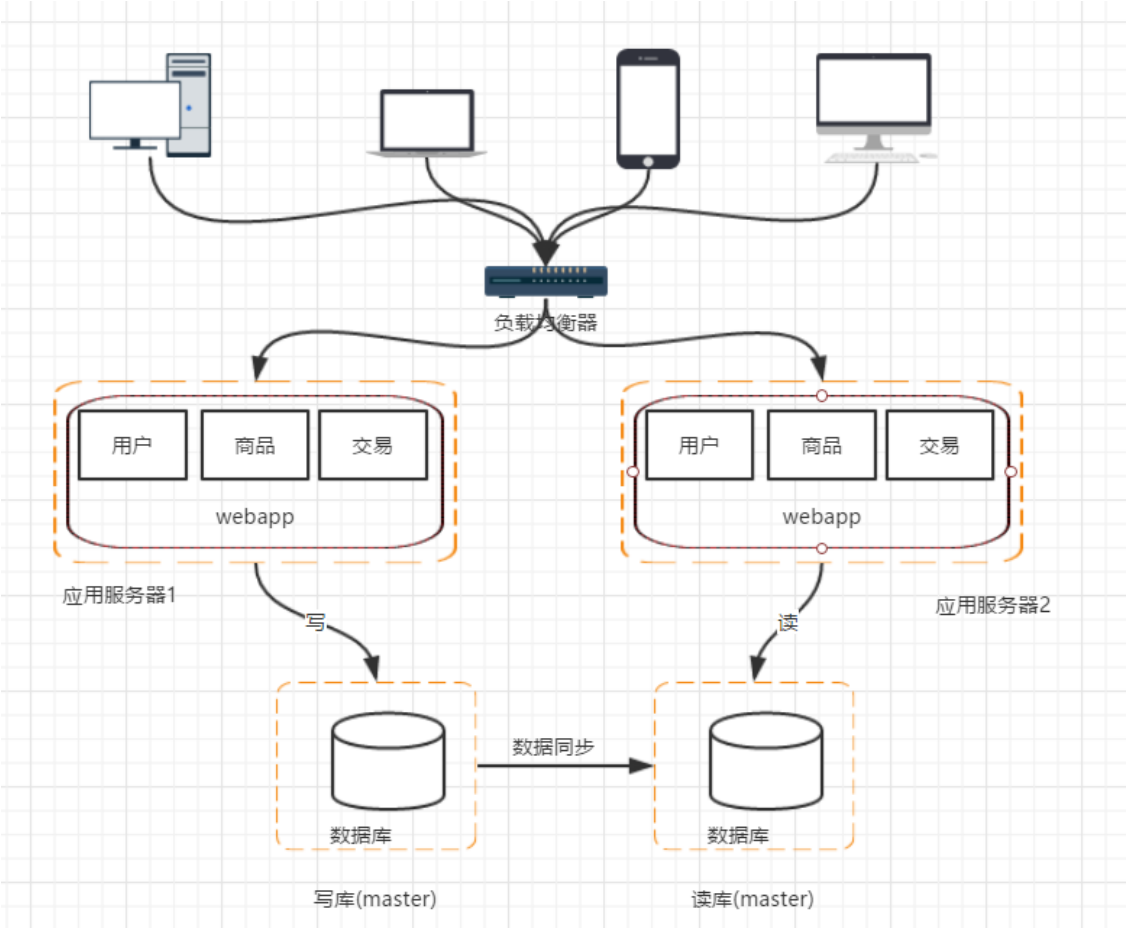
架构发展到这个阶段，各种问题也会慢慢呈现

1. 用户请求由谁来转发到具体的应用服务器
2. 用户如果每次访问到的服务器不一样，那么如何维护session



阶段四，数据库压力变大，数据库读写分离

架构演变到这里，并不是终点。上面我们把应用层的性能拉上来了，但是数据库的负载也在慢慢增大，那么怎么去提高数据库层面的负载呢？有了前面的思路以后，自然会想到增加服务器。但是假如我们单纯的把数据库一分为二，然后对于后续数据库的请求，分别负载到两台数据库服务器上，那么一定会造成数据库不统一的问题。所以我们一般先考虑读写分离的方式



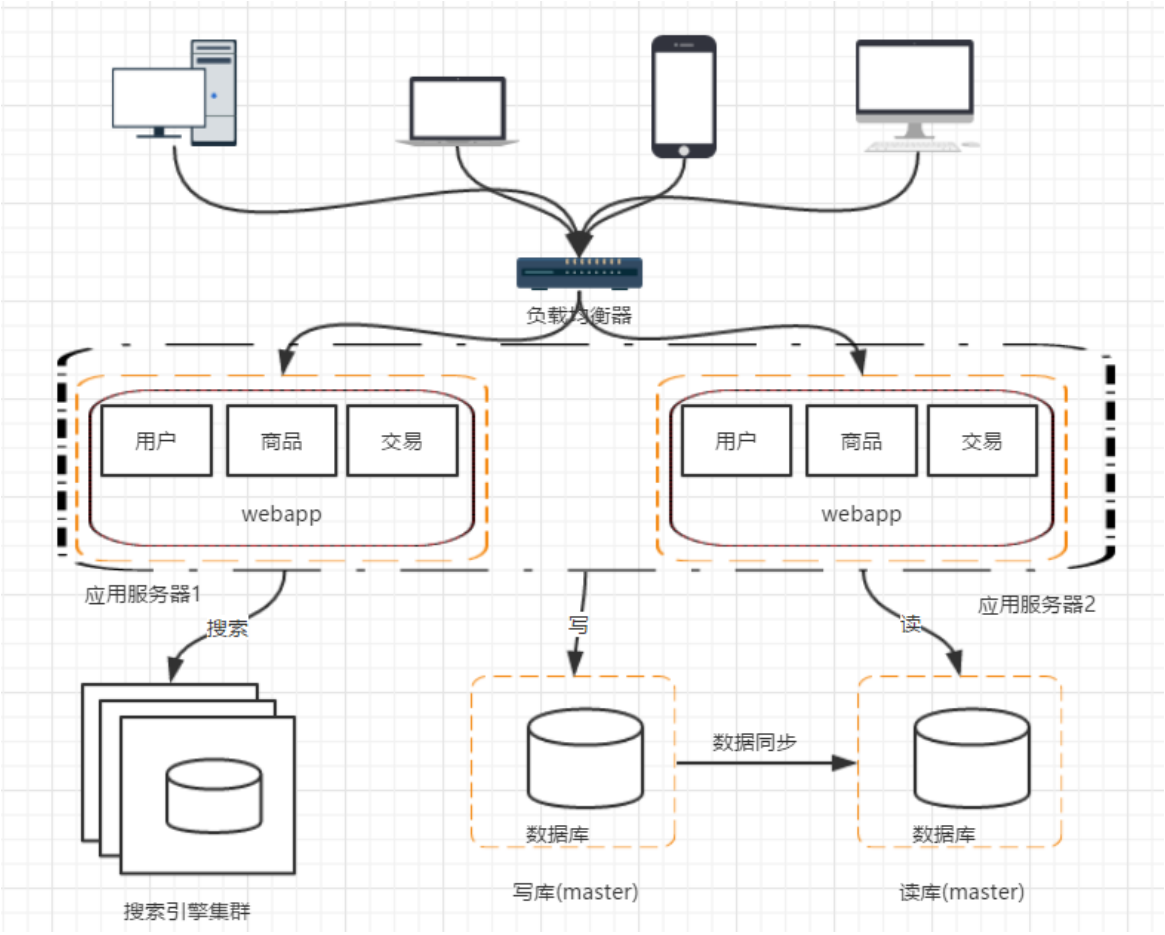
这个架构的变化会带来几个问题

1. 主从数据库之间的数据同步 ； 可以使用 mysql 自带的 master-slave 方式实现主从复制
2. 对应数据源的选择 ； 采用第三方数据库中间件，例如 mycat

阶段五，使用搜索引擎缓解读库的压力

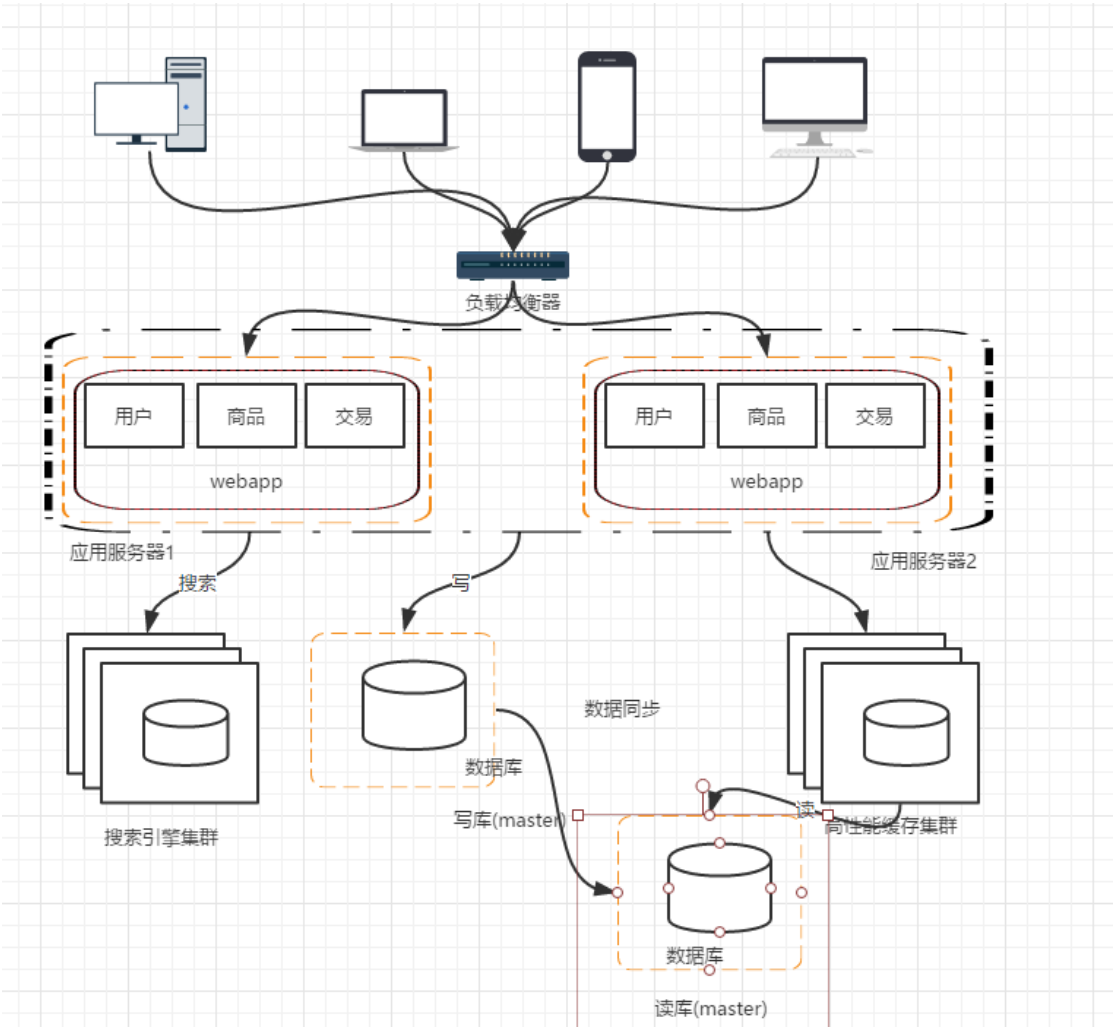
数据库做读库的话，尝尝对模糊查找效率不是特别好，像电商类的网站，搜索是非常核心的功能，即便是做了读写分离，这个问题也不能有效解决。那么这个时候就需要引入搜索引擎了

使用搜索引擎能够大大提高我们的查询速度，但是同时也会带来一些附加的问题，比如维护索引的构建。



阶段六，引入缓存机制缓解数据库的压力

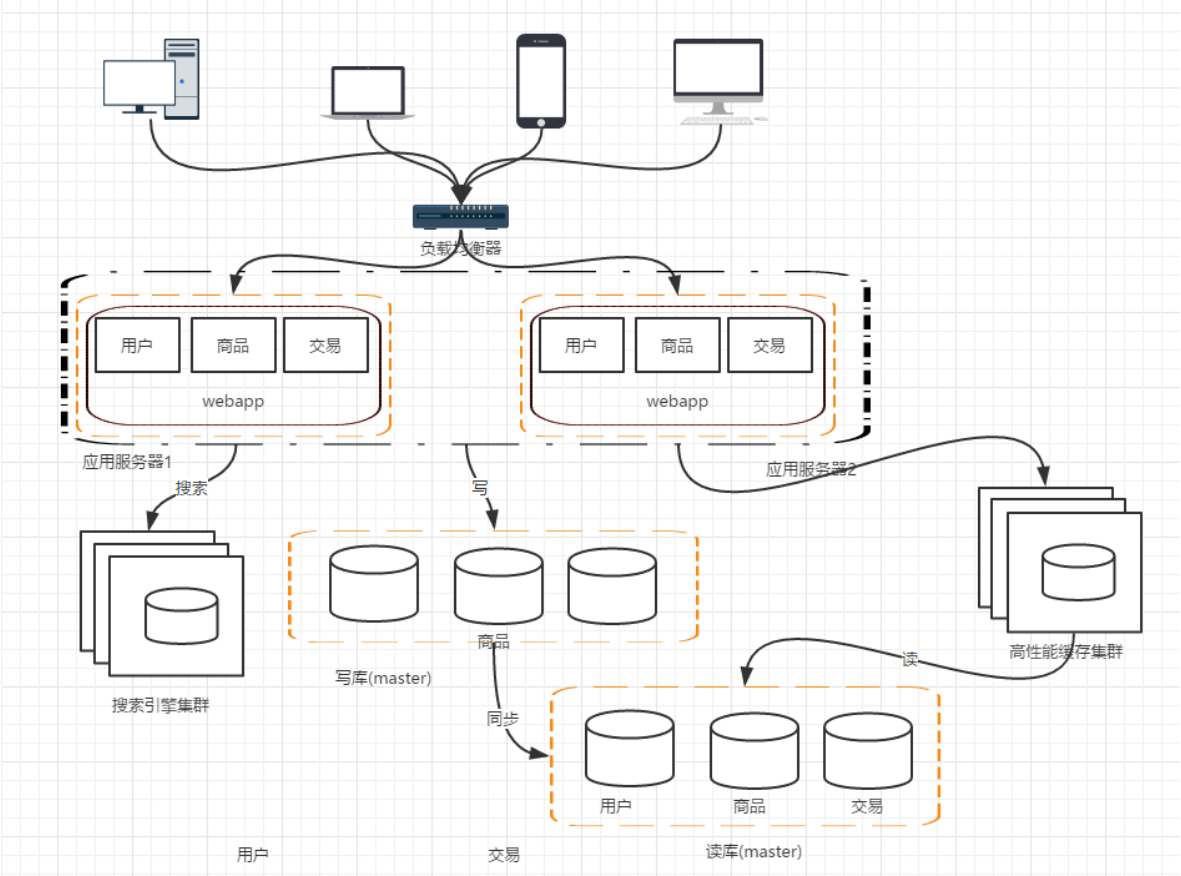
随着访问量的持续增加，逐渐出现许多用户访问统一部分内容的情况，对于这些热点数据，没必要每次都从数据库去读取，我们可以使用缓存技术，比如 memcache、redis 来作为我们应用层的缓存；另外在某些场景下，比如我们对用户的某些 IP 的访问频率做限制，那这个放内存中又不合适，放数据库又太麻烦，这个时候可以使用 Nosql 的方式比如 mongDB 来代替传统的关系型数据库



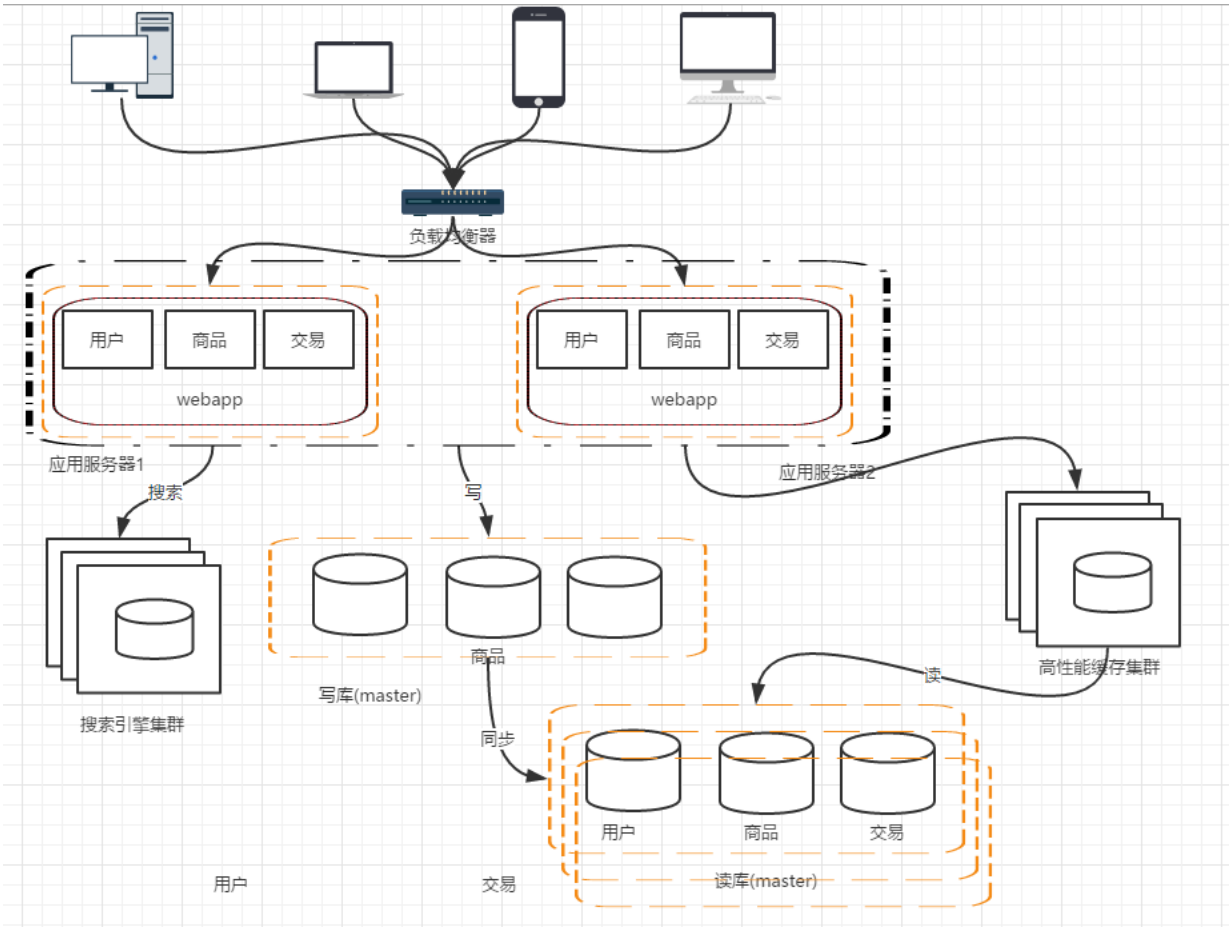
阶段七，数据库的水平/垂直拆分

我们的网站演进的变化过程，交易、商品、用户的数据都还在同一个数据库中，尽管采取了增加缓存，读写分离的方式，但是随着数据库的压力持续增加，数据库的瓶颈仍然是个最大的问题。因此我们可以考虑对数据的垂直拆分和水平拆分

垂直拆分：把数据库中不同业务数据拆分到不同的数据库

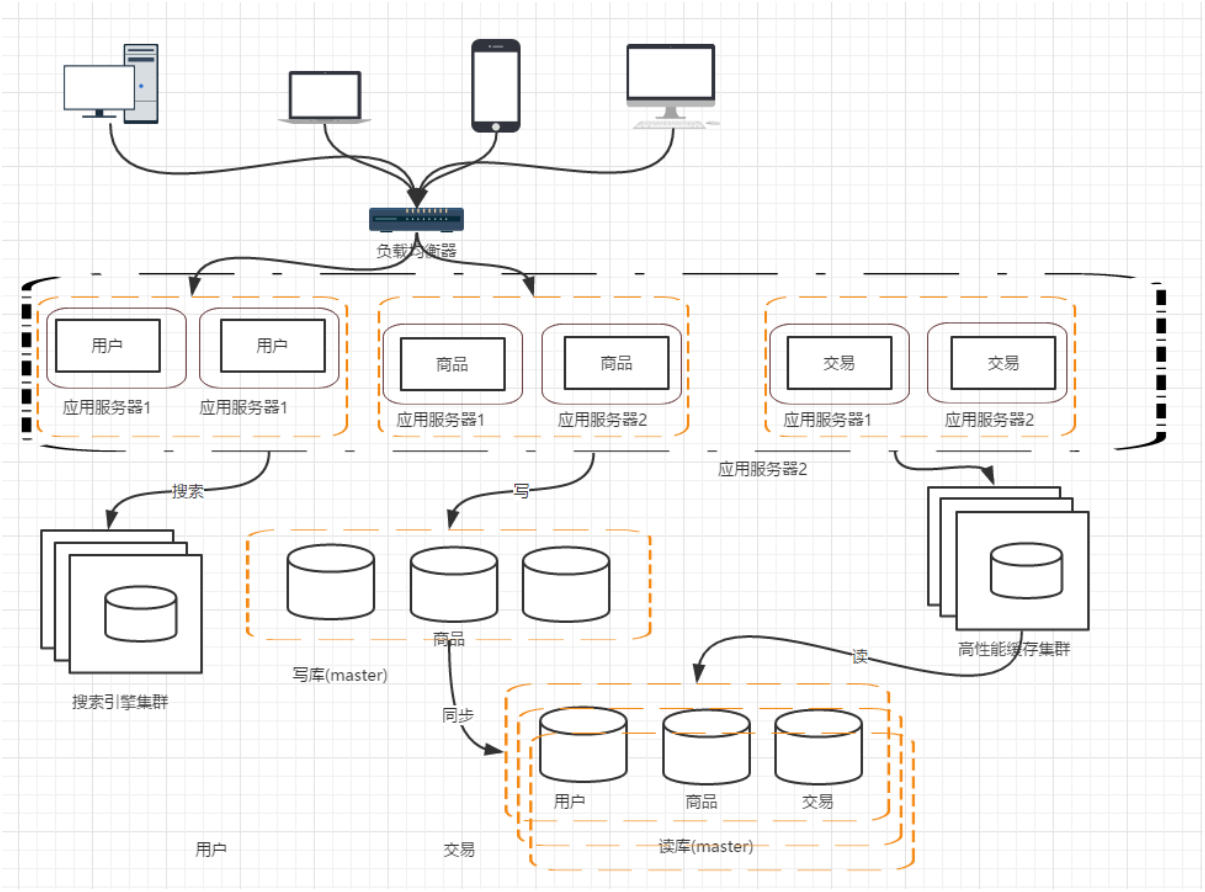


水平拆分：把同一个表中的数据拆分到两个甚至跟多的数据库中，水平拆分的原因是某些业务数据量已经达到了单个数据库的瓶颈，这时可以采取讲表拆分到多个数据库中

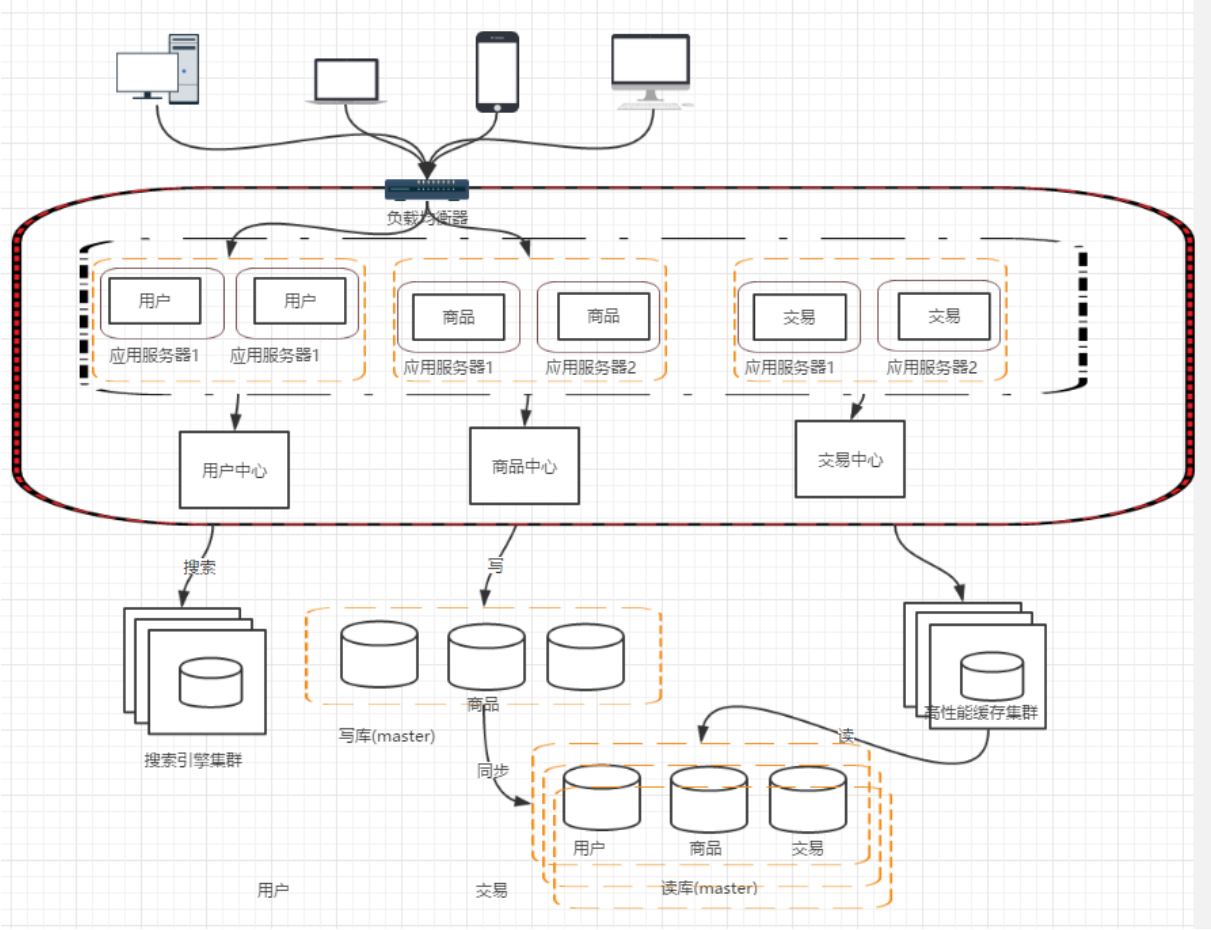


阶段八，应用的拆分

随着业务的发展，业务越来越多，应用的压力越来越大。工程规模也越来越庞大。这个时候就可以考虑讲应用拆分，按照领域模型讲我们的用户、商品、交易拆分成多个子系统



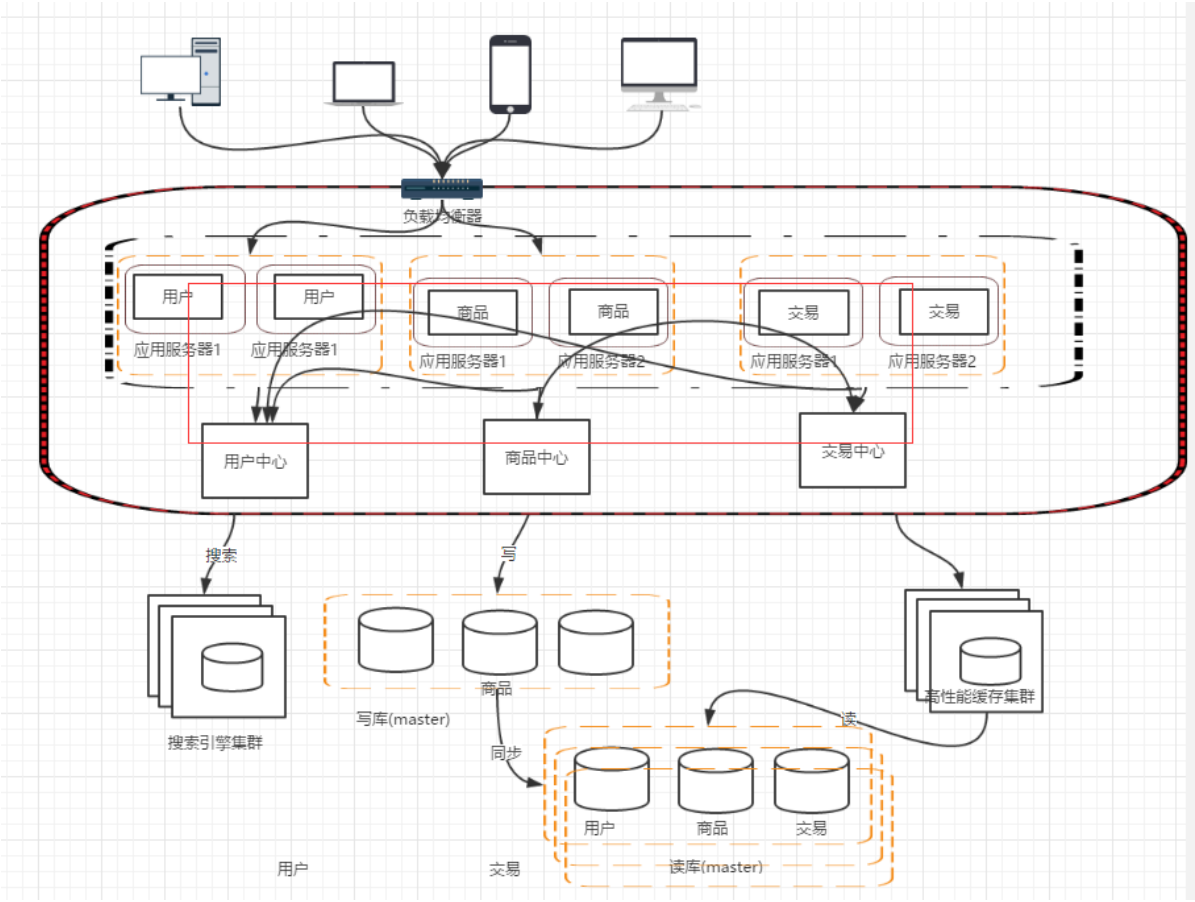
这样拆分以后，可能会有一些相同的代码，比如用户操作，在商品和交易都需要查询，所以会导致每个系统都会有用户查询访问相关操作。这些相同的操作一定是要抽象出来，否则就会是一个坑。所以通过走服务化路线的方式来解决



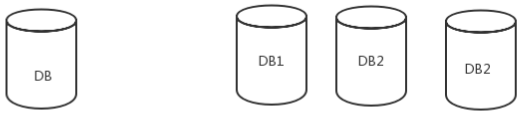
那么服务拆分以后，各个服务之间如何进行远程通信呢?

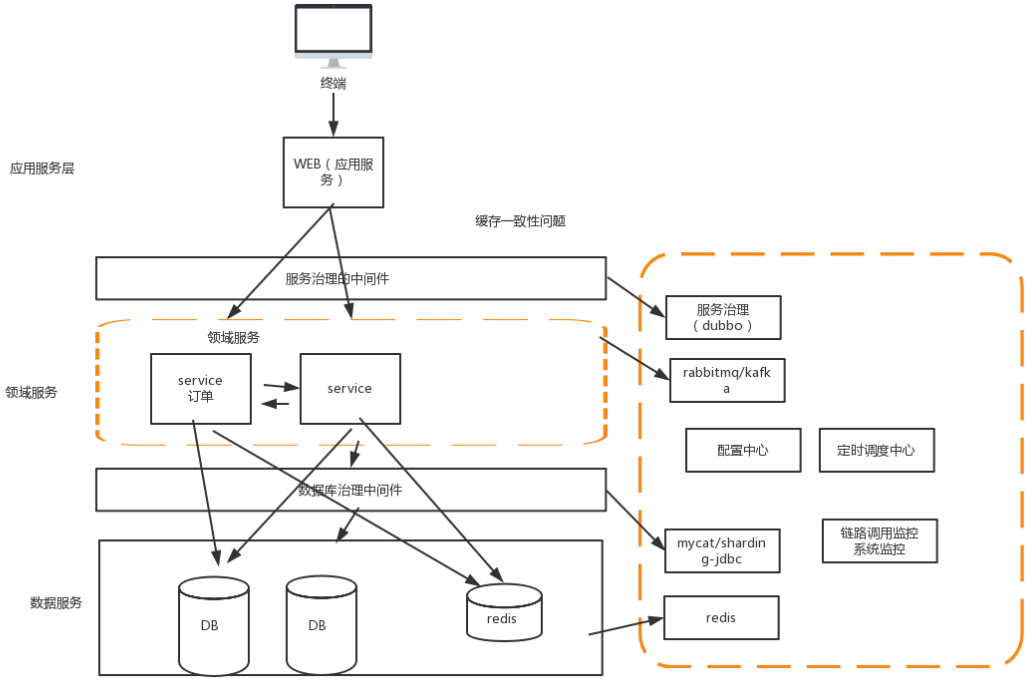
通过 RPC 技术，比较典型的有：webservice、hessian、http、RMI 等等

前期通过这些技术能够很好的解决各个服务之间通信问题，but，互联网的发展是持续的，所以架构的演变和优化还在持续。


负载均衡器
用户
用户
应用服务器1
应用服务器1
商品
商品
应用服务器1
应用服务器2
交易
交易
应用服务器1
应用服务器2
用户中心
商品中心
交易中心
搜索
写
搜索引擎集群
写库(master)
商品
高性能缓存集群
读
同步
用户
商品
交易
读库(master)
用户
交易

总结梳理





分布式领域中冯诺依曼模型的变化

前面我们讲过经典理论-冯.诺依曼体系，计算机硬件由运算器、控制器、存储器、输入设备、输出设备五大部分组成。不管架构怎么变化，计算机仍没有跳出该体系的范畴；

输入设备的变化

在分布式系统架构中，输入设备可以分两类，第一类是互相连接的多个节点，在接收其他节点传来的信息作为该节点的输入；另一种

就是传统意义上的人机交互的输入设备了

输出设备的变化

输出和输入类似，也有两种，一种是系统中的节点向其他节点传输信息时，该节点可以看作是输出设备；另一种就是传统意义上的人际交互的输出设备，比如用户的终端

控制器的变化

在单机中，控制器指的是 CPU 中的控制器，在分布式系统中，控制器主要的作用是协调或控制节点之间的动作和行为；比如硬件负载均衡器；LVS 软负载；规则服务器等

运算器

在分布式系统中，运算器是由多个节点来组成的。运用多个节点的计算能力来协同完成整体的计算任务

存储器

在分布式系统中，我们需要把承担存储功能的多个节点组织在一起，组成一个整体的存储器；比如数据库、redis（key-value 存储）

分布式系统的难点

毫无疑问，分布式系统对于集中式系统而言，在实现上会更加

复杂。分布式系统将会是更难理解、设计、构建 和管理的，同时意味着应用程序的根源问题更难发现。

三态

在集中式架构中，我们调用一个接口返回的结果只有两种， 成功或者失败，但是在分布式领域中，会出现“超时”这个状态。

分布式事务

这是一个老生常谈的问题，我们都知道事务就是一些列操作的原子性保证，在单机的情况下，我们能够依靠本机的数据库连接和组件轻易做到事务的控制，但是分布式情况下，业务原子性操作很可能是跨服务的，这样就导致了分布式事务，例如 A 和 B 操作分别是不同服务下的同一个事务操作内的操作，A 调用 B，A 如果可以清楚的知道 B 是否成功提交从而控制自身的提交还是回滚操作，但是在分布式系统中调用会出现一个新状态就是超时，就是 A 无法知道 B 是成功还是失败，这个时候 A 是提交本地事务还是回滚呢？其实这是一个很难的问题，如果强行保证事务一致性，可以采取分布式锁，但是那样会增加系统复杂度而且会增大系统的开销，而且事务跨越的服务越多，消耗的资源越大，性能越低，所以最好的解决方案就是避免分布式事务。

还有一种解决方案就是重试机制，但是重试如果不是查询接口，

必然涉及到数据库的变更，如果第一次调用成功但是没返回成功结果，那调用方第二次调用对调用方来说依然是重试，但是对于被调用方来说是重复调用，例如 A 向 B 转账，A-100,B + 100，这样会导致 A 扣了 100，而 B 增加 200。这样的结果不是我们期望的，因此需在要写入的接口做幂等设计。多次调用和单次调用是一样的效果。通常可以设置一个唯一键，在写入的时候查询是否已经存在，避免重复写入。但是幂等设计的一个前提就是服务是高可用，否则无论怎么重试都不能调用返回一个明确的结果调用方会一直等待，虽然可以限制重试的次数，但是这已经进入了异常状态了，甚至到了极端情况还是需要人肉补偿处理。其实根据 CAP 和 BASE 理论，不可能在高可用分布式情况下做到一致性，一般都是最终一致性保证。

负载均衡

每个服务单独部署，为了达到高可用，每个服务至少是两台机器，因为互联网公司一般使用可靠性不是特别高的普通机器，长期运行宕机概率很高，所以两台机器能够大大降低服务不可用的可能性，这正大型项目会采用十几台甚至上百台来部署一个服务，这不仅是保证服务的高可用，更是提升服务的 QPS，但是这样又带来一个问题，一个请求过来到底路由到哪台机器？路由算法很多，有 DNS 路由，如果 session 在本机，还会根据用户 id 或则 cookie 等信息路由到固定的机器，当然现在应用

服务器为了扩展的方便都会设计为无状态的，session 会保存到专有的 session 服务器，所以不会涉及到拿不到 session 问题。那路由规则是随机获取么？这是一个方法，但是据我所知，实际情况肯定比这个复杂，在一定范围内随机，但是在大的范围也会分为很多个域，例如如果为了保证异地多活的多机房，夸机房调用的开销太大，肯定会优先选择同机房的服务，这个要参考具体的机器分布来考虑。

一致性

数据被分散或者复制到不同的机器上，如何保证各台主机之间的数据的一致性将成为一个难点。

故障的独立性

分布式系统由多个节点组成，整个分布式系统完全出问题的概率是存在的，但是在时间中出现更多的是某个节点出问题，其他节点都没问题。这种情况下我们实现分布式系统时需要考虑得更加全面些