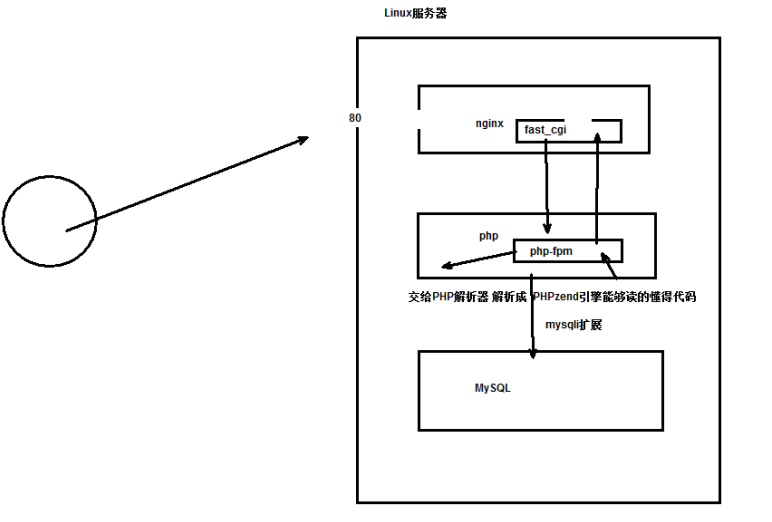
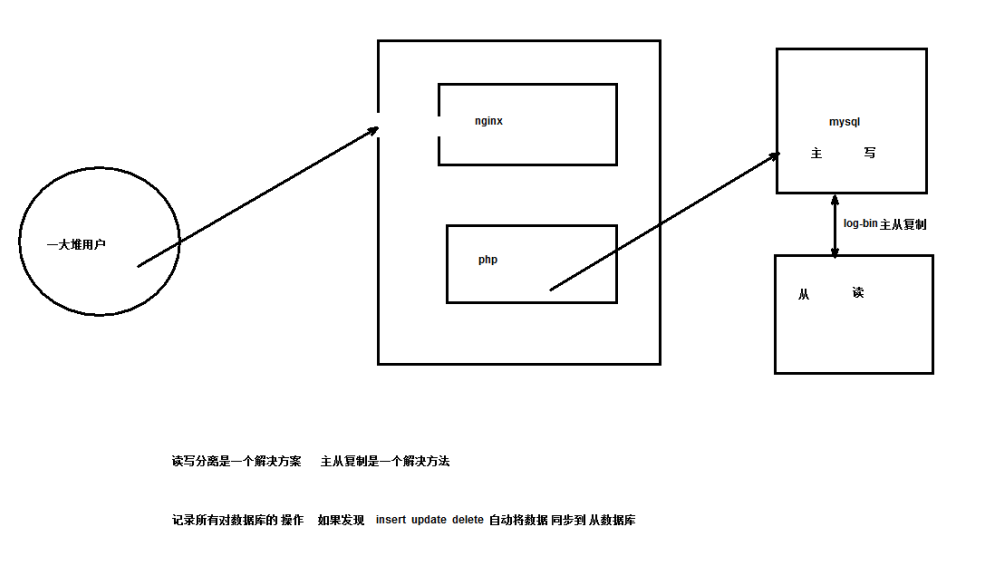
**单机架构**



**读写分离、主从复制**



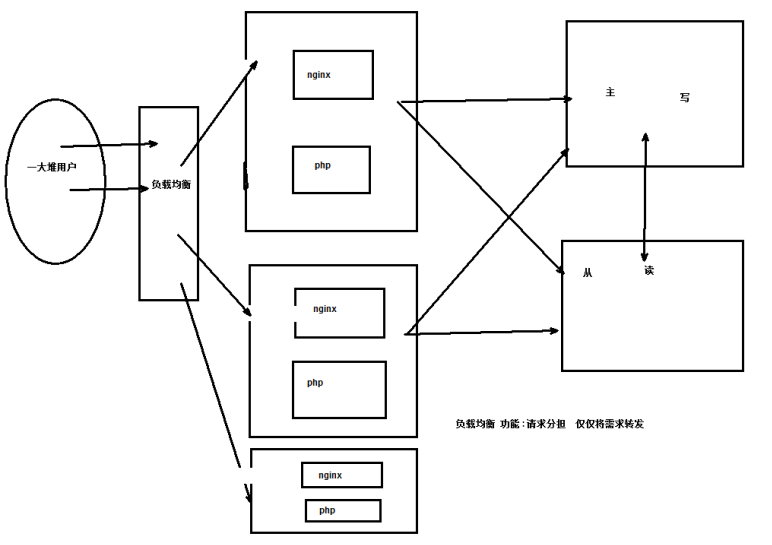
单机架构有一个木桶原理，如果一旦mysql压力增大的话，其他整个机器的性能都会下降，为了解决这个问题，一个核心的原则就是把它们分开，可以把mysql独立出来，与php和nginx分开，这样主的数据库压力就变小了。在实际的生产环境中，由单台Mysql作为独立的数据库是完全不能满足实际需求的，无论是在安全性，高可用性以及高并发等各个方面。这时候我们就可以通过采用读写分离的方案来分担数据库的压力。读写分离是靠主从复制来实现的。即当一个数据写入到主服务器后，主服务器会将写入信息写入到binlog（二进制日志）里，binlog日志会记录所有对数据库的操作，自动将数据同步到从服务器里。从服务器根据主服务器传来的binlog，生成relay-log(中继日志），然后mysql服务器再用relay-log的信息将数据写入到数据库。

因为数据库读的duo，写操作较少，所以把两者分开，一个是主一个是从，主的是写操作，从的是读操作，所以在对数据库进行写的时候也是相当于备份。这就是主从复制，读写分离

当主写上后，从是怎么读的那？？？？？

是通过binlog/log-bin日志，主库开启bin-log日志，它会进入所有的数据库的操作，当遇到inset、uodate 、delete时，会自动将数据写入日志中；当线程请求主库 的binlog，并将得到的binlog日志写到relay log（中继日志） 文件中；从库会读取relay log文件中的日志，并解析成具体操作，来实现主从的操作一致，而最终数据一致；

**负载均衡**



当我们的用户量很大或者请求数很多时，并发量会很高，我们的服务器会承受不了那么大的压力，这时候可以采用负载均衡请求分担。Web负载均衡（Load Balancing），简单地说就是给我们的服务器集群分配“工作任务”，而采用恰当的分配方式，对于保护处于后端的Web服务器来说，非常重要。

功能：主要是请求分担，它仅仅是将需求转发

（集群和分布式）

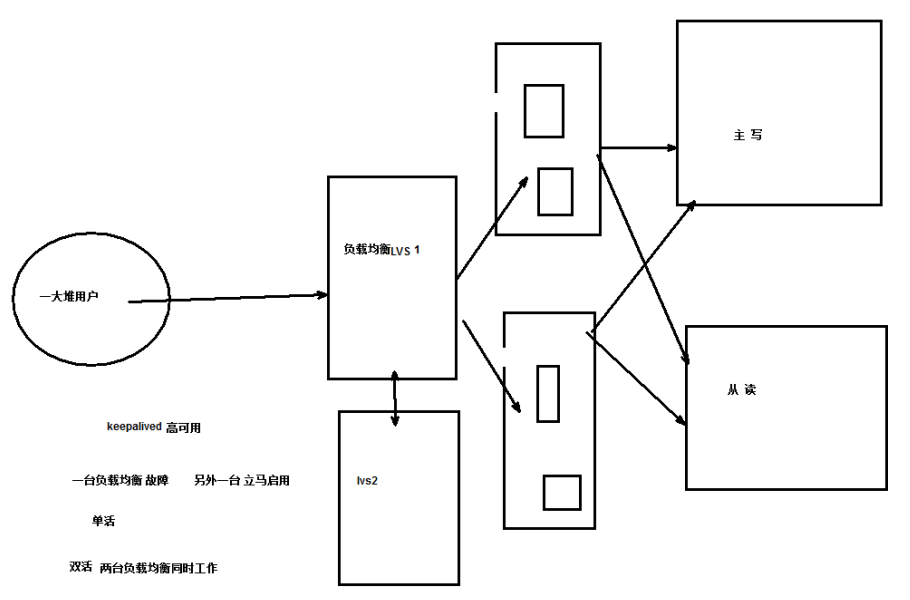
当用户量一下子爆发的时候，一台服务器容易崩溃，所以多设置几台，这就有了集群

集群 : 就是一个物理形态 一大堆机器放在那里 就是集群

分布式 : 工作的方式 将任务放在 多个 隔离物理节点 上进行

​ 分布式中各个节点互不通信 统一受 调控中心调度 （指定路由、负载均衡 、发现并剔除故障设备 、扩 容 ）

**高可用架构**

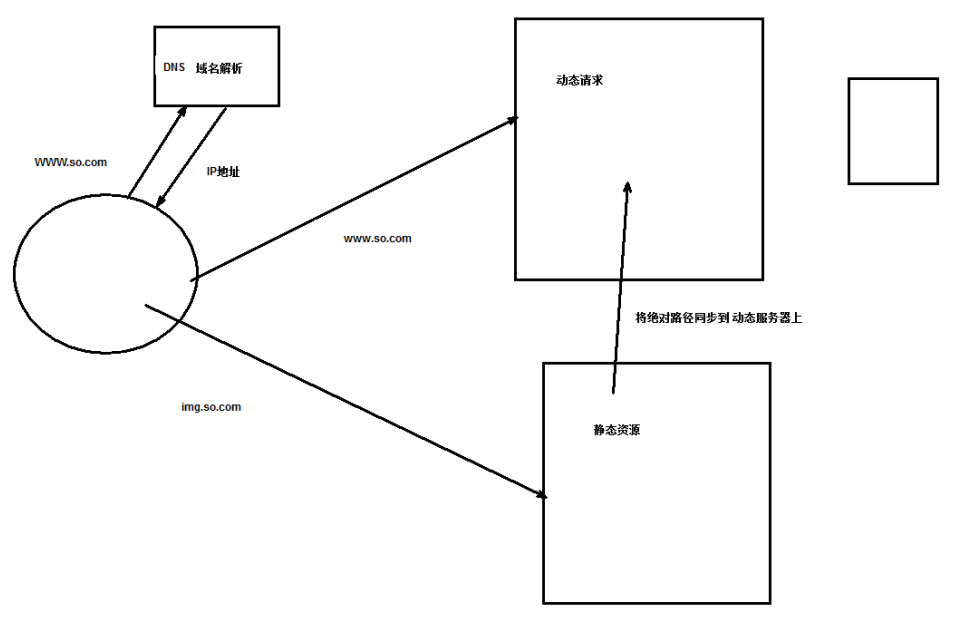


高可用指的是通过尽量缩短因日常维护操作和突发的系统崩溃所导致的停机时间，以提高系统和应用的可用性。

单活：首先是一台负载均衡承担角色，一旦出现故障，另外一台立马启用

双活：两台负载均衡同时工作，同时承担角色（现在比较流行）

**动静分离**



动静分离：html和php代码分离，html可以放在cdn上，然后计算或拿数据用ajax解决。        //**两步走战略**

**动静分离**：说白了，两步走战略，将主要的访问压力放到cdn上，主web服务器相当于延迟加载，即页面需要数据时，通过ajax去读取。

**优势**：分担负载，减轻主web服务器的压力，适用于大负载。

　　　　　html放置cdn，同时还可以通过配置缓存要客户浏览器中，类似于css、js那样。极大减轻压力。

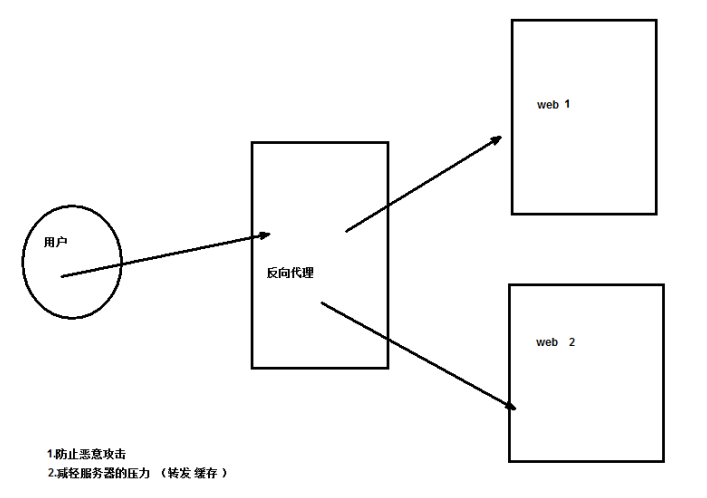
**劣势**：网络环境不佳时，ajax回应很慢，导致页面出现空白，出错处理会不好看。

　　　　　编程方面，url之类参数都需要写死，参数修改麻烦，

　　　　　增加了复杂度。

首先用户发送域名（www.so.com）到DNS域名解析，域名返回一个ip地址，用户根据返回的ip进行访问，访问的时候，静态资源（html、css、js、image）耗费很大的http请求，为了减少服务器的压力，可以让用户把一些静态资源的请求发送到动态资源服务器上，上传的时候用location进行匹配，一旦匹配到有static或upload就表示是上传文件或是静态文件，就走静态资源的服务器，把一些图片的绝对路径或二级域名发送到动态请求资源的服务器上，（或者将存储图片的目录挂载在动态请求的服务器上，但请求还是在原来的服务器上，不影响域名解析，仅仅是把文件放在了处理动态请求的服务器上，但http请求仍然是在处理静态资源的服务器上），这样就实现了动静分离。（OSS存储技术实现了动静分离）

**反向代理**



代理 : 正向代理和 反向代理

正向代理 一般不让你直达目标主机，先到代理服务器上 客户端到服务端 (例如:黑客,一般不直达目标主机)

反向代理 nginx在服务端 从服务端到客户端

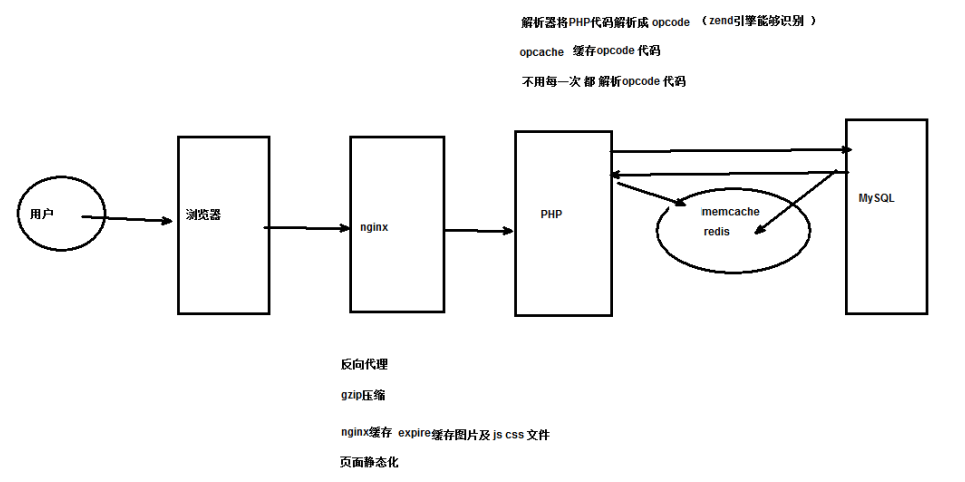
反向代理的作用 : 转发用户的请求

原理:

用户发送请求后先到反向代理，其实反向代理就是负载均衡的一种。反 向代理会将用户的请求转发到某一台服务器上，先将服务器返回的结果存起来，等到下一个用户再次发出同样的请求时直接将结果返回给用户，这样就无形中减轻了服务器的压力。为了防止恶意攻击，反向代理还可以先把用户的请求转发出去，如果遇到跨站脚本攻击（在url有拼接script，script可以执行）等恶意攻击，可以把请求转发出去，转发到别的地方或反馈给用户404或500等错误，一旦遇到不确定的请求就转发出去。一旦mysql注入的话，使用反向代理，可以阻挡80%的mysql攻击

作用 : 转发用户的请求

**缓存架构**



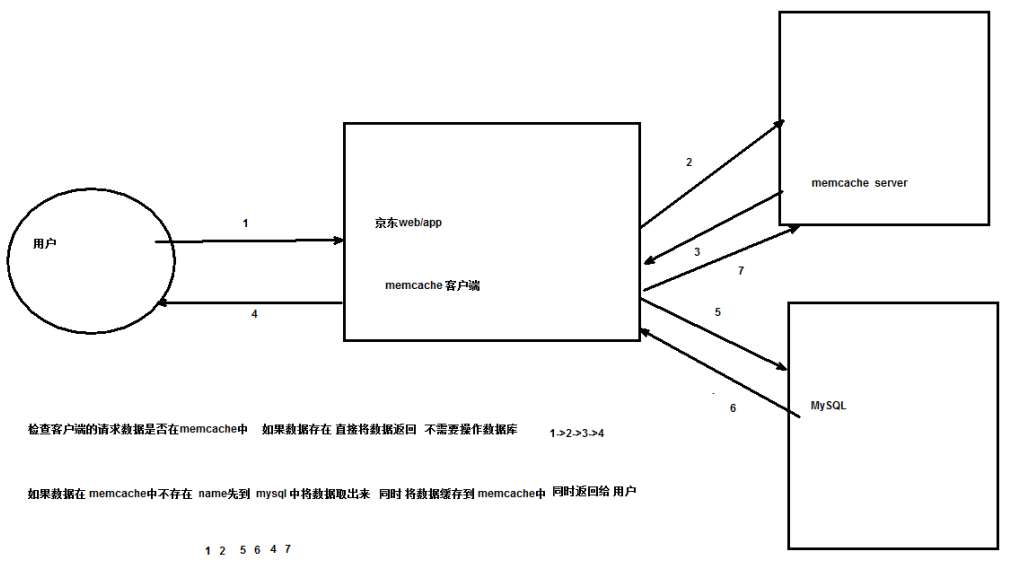
原理：首先用户访问浏览器，浏览器可以先进行缓存，可以缓存我们的图片（第一次打开网页很慢，第二次打开变快，这就是浏览器的缓存）；

接下来是nginx的缓存，nginx有哪些缓存，例如：反向代理架构，gzip压缩，expire缓存图片以及js、css文件(比如可以缓存一天24小时，24小时过去图片就失效)，页面静态化（用户访问网站发送的是动态请求，静态化的话就不用再请求数据库了，我们可以直接把静态页面返回给用户，这样服务器没什么压力。页面静态化有助于seo爬虫）；

php缓存：运行原理：首先nginx中fast\_cgi发送请求，php中php-fpm接收，然后php-fpm将请求交给解析器，解析器解析成PHPzend引擎能够识别的opcode代码。opcache将opcode代码缓存下来，这样就不用每次都解析opcode代码。

php过来请求先看memcache或者redis中有没有，没有的话就到mysql,然后mysql把数据给php的同时，把数据给memcache或者redis一份，这样下次php再请求就直接到memcache或者redis中。

**Memcache缓存**



内存分布式缓存系统 在内存中维护一个 巨大的hash表 内存中可以存图像 文件 、视频 将数据库中的数据缓存到内存 中 以 减轻 数据库的压力

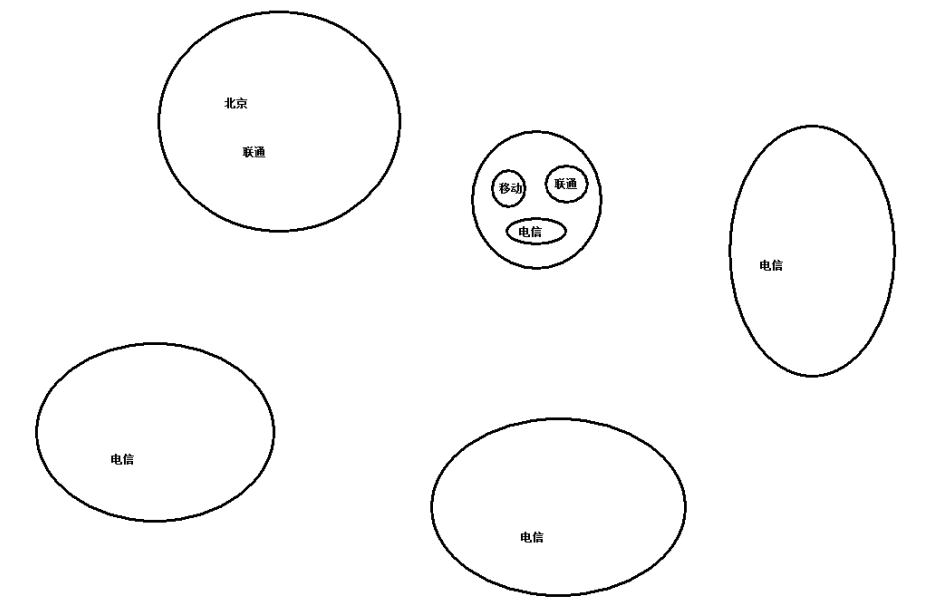
原理：首先用户先发送请求，检查客户端的请求数据是否存在memcache服务器端，如果数据存在，直接将数据返回，不需要操作数据库。（1=>2>3>4）

如果数据在memcache中不存在，就先到mysql中将数据取出来，返回给客户端，发送给用户一份，同时将数据缓存到memcache中。（1=>2=>5=>6>=>4、7）

memcache是多线程工作，因为内存是多用户多任务，它操作的是内存。

进程就是将我们这个程序或者服务在内存中或者CPU中开辟一个空间，线程就是在空间中工作。

**运营商储备策略**



类似于高可用架构，有备份，一个有故障，另一个立马顶上

**单点登录(SSO):**

https://baike.baidu.com/item/%E5%8D%95%E7%82%B9%E7%99%BB%E5%BD%95/4940767?fr=aladdin

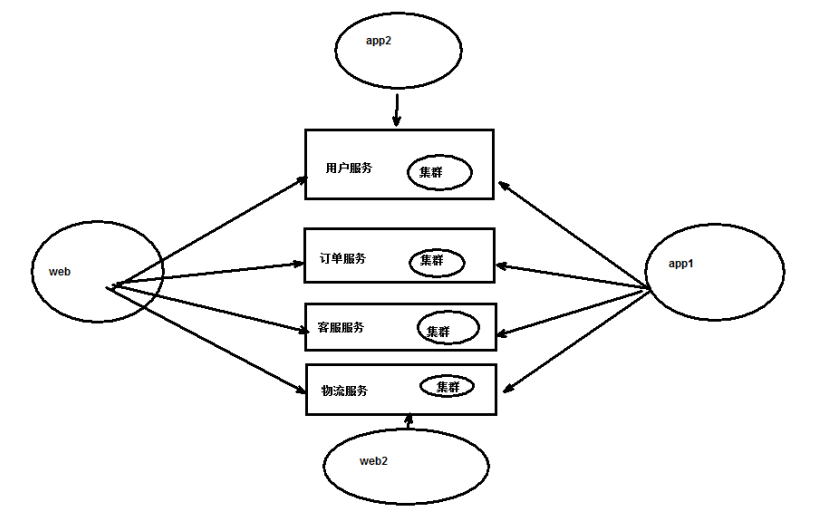
用户只需要登录一次就可以访问所有相互信任的应用系统。(cookie共享)

要实现SSO，需要以下主要的功能：

所有应用系统共享一个身份认证系统。  
　　统一的认证系统是SSO的前提之一。认证系统的主要功能是将用户的登录信息和用户信息库相比较，对用户进行登录认证；认证成功后，认证系统应该生成统一的认证标志（ticket），返还给用户。另外，认证系统还应该对ticket进行效验，判断其有效性。

所有应用系统能够识别和提取ticket信息  
　　要实现SSO的功能，让用户只登录一次，就必须让应用系统能够识别已经登录过的用户。应用系统应该能对ticket进行识别和提取，通过与认证系统的通讯，能自动判断当前用户是否登录过，从而完成单点登录的功能。

**业务分离架构**



（app1,app2,web1,web2四个产品，可以共享用户，订单等）

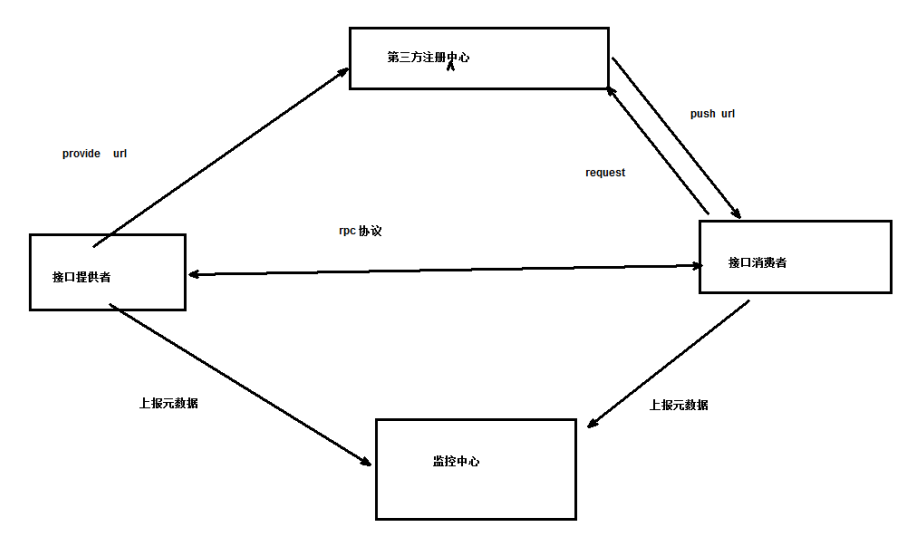
我们的业务很庞大，越做越庞大，但是每个业务在各个产品上都有可能用到，所以这个时候我们需要做一种操作，就是将业务独立出来，那怎么样让业务独立，我们可以以接口的方式，一个接口是服务者，一个是消费者，用户服务这些就是服务的提供者、生产者，app1这些就是消费者。对于微信开发，那就open API（开放接口），就是说你这个产品开放出去别人可以用，可以定制，还有第三方登录也是open API的一种。

对于每一个服务都有一个强大的集群支撑着。现在的数据库并不是直接操作，而是通过一个接口，只能看到成功返回1，失败返回0，这叫中间件。相当于一个临时数据库，如果测试没有问题再送到线上。

淘宝搜索就把业务给独立出来了，包括淘选购。

业务分离涉及到一个问题:服务化

**服务治理**



服务化和服务治理严格遵循了一个概念：高内聚低耦合

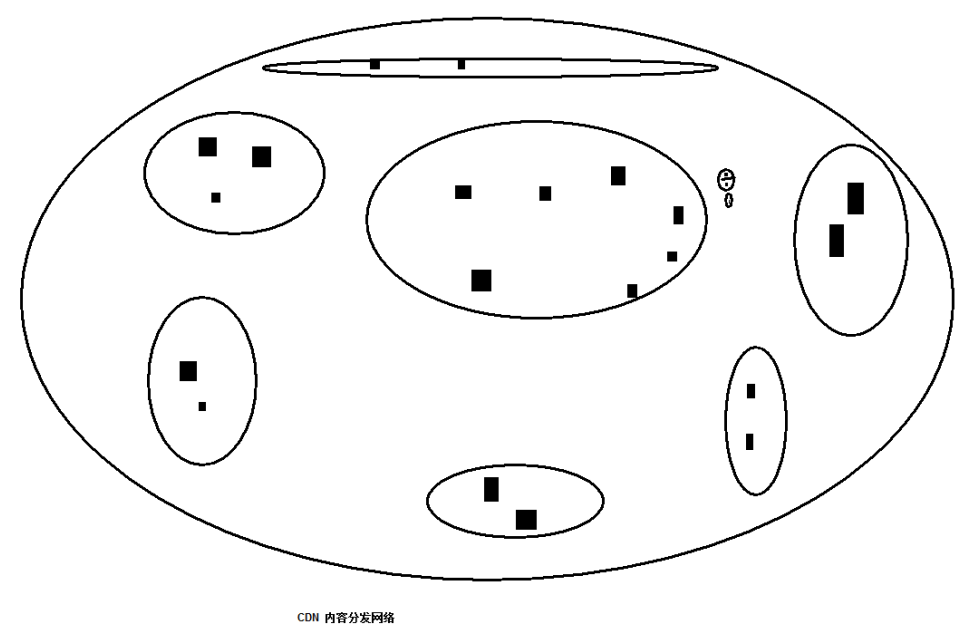
高内聚:一个产品内部之间联系非常紧密；

低耦合：功能和功能之间联系越少越好，越独立越好。

一个是接口的生产者，一个是接口的消费者，第三方注册中心

如果我修改了，接口提供者会向第三方注册中心发送接口地址（provide url）,当接口消费者会往第三方发送请求，request请求这个接口，第三方会实时监控，如果第三方有更新，会立马推送给接口消费者（pull url），现在走的协议都是基于tcp、udp的http协议，http有一个不足之处，在php这一块存储的时间比较短，为了解决这个问题可以自己单独建立一个通道，一旦消费者与提供者成功，就建立一个通道，单独使用rcp协议；；；还要有一个监控中心，接口消费者与提供者上报源数据，监控中心实时监控接口提供者是否存活，也可以把结果告诉给第三方注册中心，如果消费者需要请求或者出现堵塞，第三方提供新的或者提供者提供新的url给消费者。

**CDN(Content Delivery Network内容分发网络)**



将内容缓存在离用户最近的一个网络节点，这个网络节点就是服务器，当用户访问的时候就会快一点(电缆：网线越长，访问越慢)。

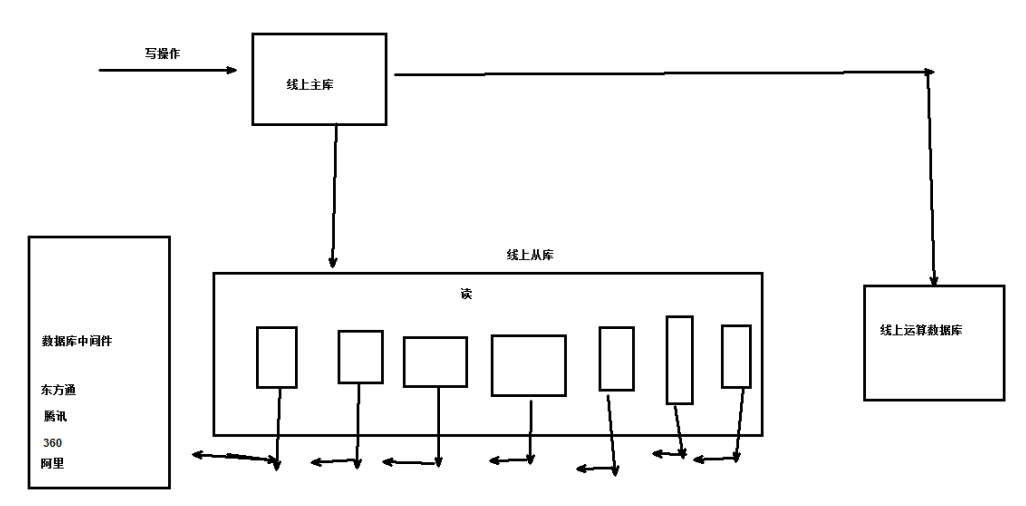
静态文件或者经常访问的文件可以用CDN加速，比如抢购的时候为了提高打开速度可以使用cdn加速，因为不同地区的人都要抢，为了公平起见，网速尽可能要保持一样。秒杀的时候首页一般是要cdn加速，可以把图片地址发给cdn运营商进行cdn加速，加速之后还要回源，可以按照回源策略的地址在网上运行，用户请求的时候请求这个回源的地址。

CDN是怎么收费的？它一般是按照流量收费。

Cdn回源：

常规的CDN都是回源的。即：当有用户访问某一个URL的时候，如果被解析到的那个CDN节点没有缓存响应的内容，或者是缓存已经到期，就会回源站去获取。如果没有人访问，那么CDN节点不会主动去源站拿的。

**分布式数据库**

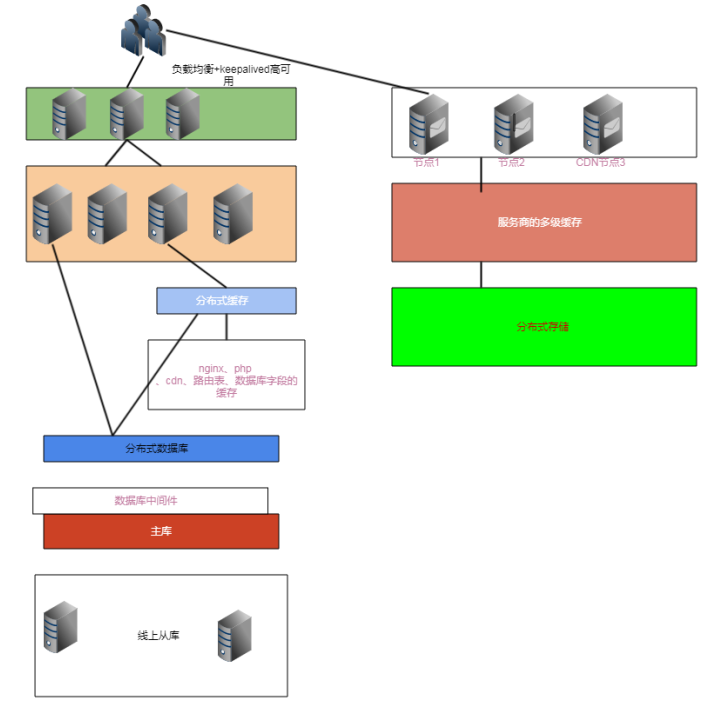


数据库有线上主库、线上从库和线上运算数据库

用户请求过来之后，在写操作的时候走线上主库，读的时候是线上从库。从库可以是个集群，因为读压力很大，读的时候数据很多。我们把主库要同步过来，还可以有一个专门处理运算的库线上运算数据库。从库是个集群，所以我们要做监控，将数据统一上报到数据库中间件。

数据库中间件可以解决的问题：数据库可以拆分，可以做主从复制，这个就是不用在数据库上面操作，所有的操作交给数据库中间件。阿里、腾讯、百度都有数据库中间件，它会提供相应的服务，你只需要按照它的服务模块付费就可以了。它有什么服务：东方通、腾讯、360、阿里。就比如分库分表模块、主从复制模块等，就好比解锁一个功能，功能都有，付费之后就可以使用。配置相当简单，因为它在实时监控线上从库里的集群，所以只要把IP地址给它就好，哪里有故障可以及时发现。

**千万级PV架构**

动态架构：

几台负载均衡+keepalived高可用，然后是一个web集群，由负载均衡和反向代理往外分担，分担完了之后该数据库出场了。

用户发出请求，首先找分布式数据库，在这之前， web集群与分布式数据库之间有分布式缓存（缓存可以有几种：nginx缓存、路由表缓存、文件缓存、memcached、radis缓存等，php缓存中除了opcode缓存，还有就是在自己封装model类的的时候，有一个数据库表字段的缓存，缓存表字段的好处是请求的时候不用每次都读取表结构），用户发出请求后，会先找分布式缓存的存储，如果缓存中没有，如果没有才到数据库请求，在返回的时候也会向缓存里面存储一份，以便下次请求的时候读取。

数据库这还有一个东西叫数据库中间件，一般它承担的角色有：分库分表分机器、监控等。线上从库还可以做计算，平时可以负责计算；在这么多的分布式集群中如果有空闲的，功能上很单一的，可以做计划任务服务器。

静态架构：

首先有个cdn加速，我们将我们经常访问到的静态文件缓存到cdn上进行加速，有cdn节点1，节点2等。cdn后有一个叫服务商的多级缓存，然后就是自己的分布式存储，不同的服务器存放不同的文件。比如:一台服务器只存储上传文件，另外一台存放我们网站中必须的图片，js，css。

这个千万级PV架构，数据库的数量，以及集群的数量决定了能承受多大的，一般这个架构能承受1千万的pv，但是还是要算一算，算QPS（我们当时预估的时候我们保守估计是一千万PV一天，峰值是一千万PV。比如8台服务器，线上从库用了四台，缓存memcache一台，或者说redis三台，还有一个存储备份的服务器，web负载均衡等。 视频一般是8000-10000个资源，用户的话大概就是1：10或者1:15 活跃用户）