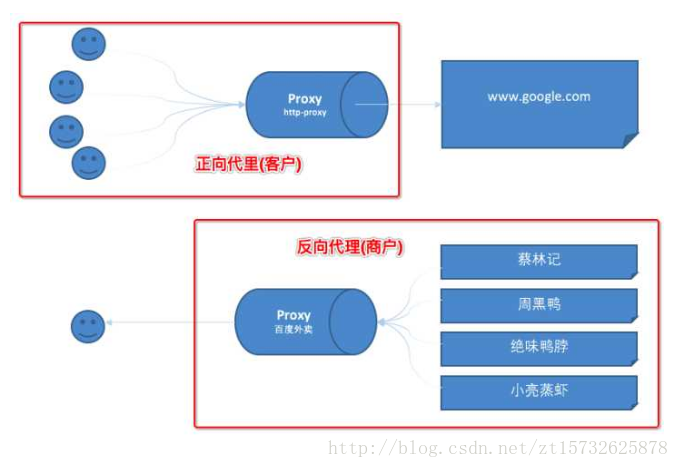
# 高可用

## Nginx

*Nginx* (engine x) 是一个高性能的[HTTP](https://baike.baidu.com/item/HTTP)和[反向代理](https://baike.baidu.com/item/%E5%8F%8D%E5%90%91%E4%BB%A3%E7%90%86/7793488)服务

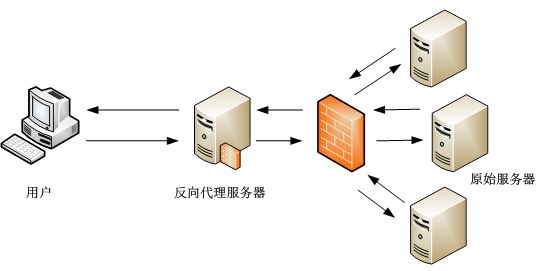
正向代理与反向代理的区别



nginx的作用

1. 保护内网安全   
    2. 负载均衡   
    3. 缓存，减少服务器的压力

用户请求流程图



负载均衡策略

|  |  |
| --- | --- |
| 轮询 | 默认方式 |
| weight | 权重方式 |
| ip\_hash | 依据ip分配方式 |
| least\_conn | 最少连接方式 |
| fair（第三方） | 响应时间方式 |
| url\_hash（第三方） | 依据URL分配方式 |

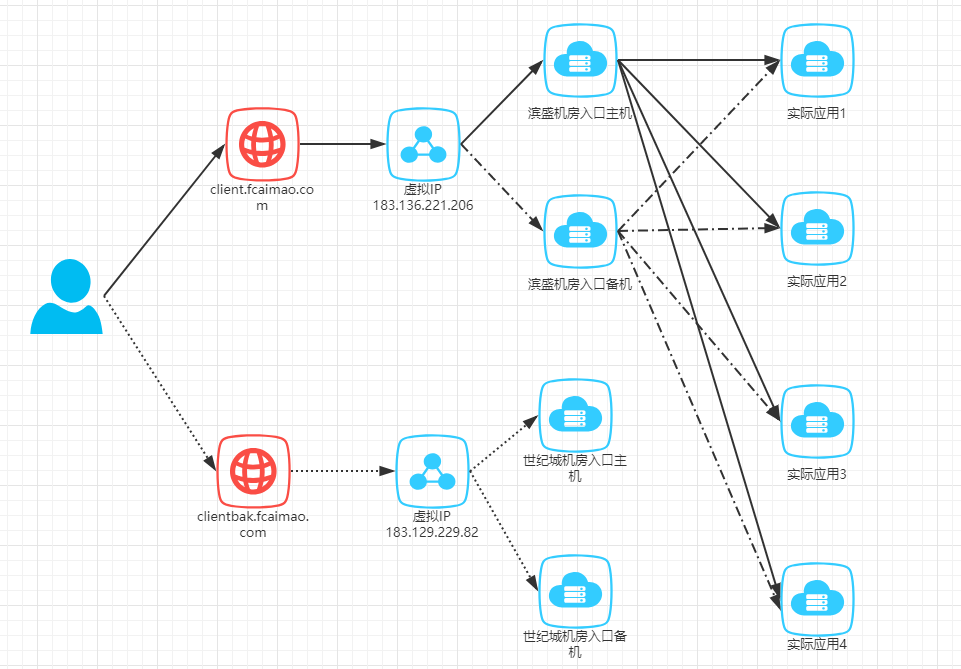
## Keeplived

nginx只能做到单应用停机不影响整体服务，

如何做到单服务器闳机不影响业务

我们使用keepalived来实现双机热备

## 现在的结构图



# 高并发

## **同步和异步**

同步就是一件事，一件事情一件事的做。  
 异步就是，做一件事情，不引响做其他事情。

例如：吃饭和说话，只能一件事一件事的来，因为只有一张嘴。  
                但吃饭和听音乐是异步的，因为，听音乐并不引响我们吃饭。

在补充一下(脏数据和不可重复读的相关概念):

       脏数据

　　脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这

个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是脏数据(Dirty Data)，依据脏数据所做的操作可能是不正确的。

 　　不可重复读

　　不可重复读是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读

## 锁

### java：同步关键字synchronized

数据库锁

### 分布式锁

redis

zookeeper

# 高性能

## 本地缓存

### ehcache

老牌， Hibernate默认的缓存

### Caffeine

新兴 Spring5， springboot2

单机服务可以考虑使用本地缓存

## 分布式缓存

### memcached

适合多读少写，大数据量的情况：文章内容、用户信息等

据说性能高一些，差别不大

### redis

memcached能做的都能做到，但是维护成本比memcached高

提供了5种类型

#### String 字符串

普通情况会考虑使用memceched，除非需要安全的计数

#### Hash 散列

主要也是为了计数，将统一类型的计数放到一个散列中

#### List 列表

可以用来做分布式队列

抢红包活动就是先在redis初始化一个该活动的list

行情的实施交易（近100条）

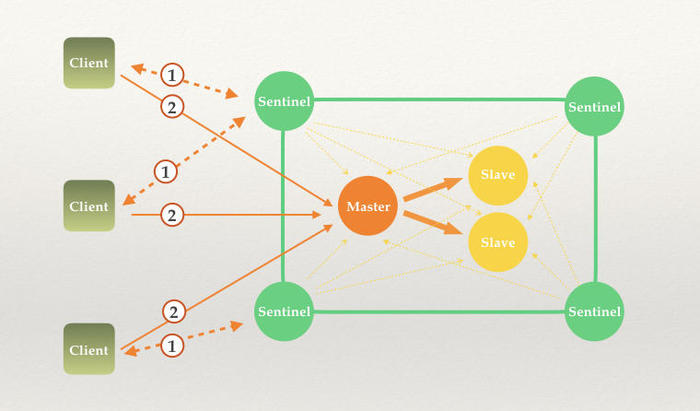
#### Set 集合

存储好友关系，将一个好友的所有关注用户存在一个set，Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能，对上面的所有集合操作，你还可以使用不同的命令选择将结果返回给客户端还是存集到一个新的集合中。

除此之外 redis 还支持 发布订阅(pub/sub)

#### 分布式模式

1. Redis Sentinel



sentinel 类似于仲裁会

负责判断master是否可用、选举授权新的master

1. Redis Cluster

Redis Cluster是Redis的分布式解决方案，在Redis 3.0版本正式推出的，有效解决了Redis分布式方面的需求。



# 其他

## Zookeeper

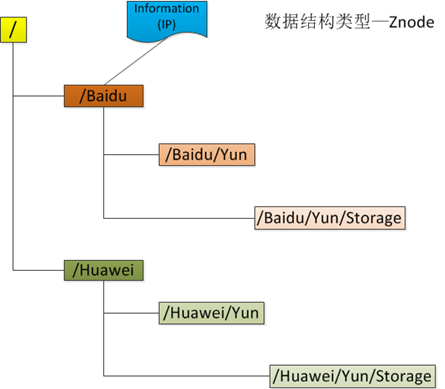
zooKeeper 是一个开源的分布式协调服务，由雅虎创建，是 Google Chubby 的开源实现。

分布式应用程序可以基于 ZooKeeper 实现诸如数据发布/订阅、负载均衡、命名服务、分布式协

调/通知、集群管理、Master 选举、配置维护，名字服务、分布式同步、分布式锁和分布式队列

等功能。

有很多的开源系统用到了zk：Dubbo、Kafka、HBase



(1) 引用方式

Zonde通过路径引用，如同Unix中的文件路径。路径必须是绝对的，因此他们必须由斜杠字符来开头。除此以外，他们必须是唯一的，也就是说每一个路径只有一个表示，因此这些路径不能改变。在ZooKeeper中，路径由Unicode字符串组成，并且有一些限制。字符串"/zookeeper"用以保存管理信息，比如关键配额信息。

(2) Znode结构

ZooKeeper命名空间中的Znode，兼具文件和目录两种特点。既像文件一样维护着数据、元信息、ACL、时间戳等数据结构，又像目录一样可以作为路径标识的一部分。图中的每个节点称为一个Znode。 每个Znode由3部分组成:

① stat：此为状态信息, 描述该Znode的版本, 权限等信息

② data：与该Znode关联的数据

③ children：该Znode下的子节点

ZooKeeper虽然可以关联一些数据，但并没有被设计为常规的数据库或者大数据存储，相反的是，它用来管理调度数据，比如分布式应用中的配置文件信息、状态信息、汇集位置等等。这些数据的共同特性就是它们都是很小的数据，通常以KB为大小单位。ZooKeeper的服务器和客户端都被设计为严格检查并限制每个Znode的数据大小至多1M，但常规使用中应该远小于此值。

(3) 数据访问

ZooKeeper中的每个节点存储的数据要被原子性的操作。也就是说读操作将获取与节点相关的所有数据，写操作也将替换掉节点的所有数据。另外，每一个节点都拥有自己的ACL(访问控制列表)，这个列表规定了用户的权限，即限定了特定用户对目标节点可以执行的操作。

(4) 节点类型

ZooKeeper中的节点有两种，分别为临时节点和永久节点。节点的类型在创建时即被确定，并且不能改变。

① 临时节点：该节点的生命周期依赖于创建它们的会话。一旦会话(Session)结束，临时节点将被自动删除，当然可以也可以手动删除。虽然每个临时的Znode都会绑定到一个客户端会话，但他们对所有的客户端还是可见的。另外，ZooKeeper的临时节点不允许拥有子节点。

② 永久节点：该节点的生命周期不依赖于会话，并且只有在客户端显示执行删除操作的时候，他们才能被删除。

(5) 顺序节点

当创建Znode的时候，用户可以请求在ZooKeeper的路径结尾添加一个递增的计数。这个计数对于此节点的父节点来说是唯一的，它的格式为"%10d"(10位数字，没有数值的数位用0补充，例如"0000000001")。当计数值大于232-1时，计数器将溢出。

(6) 观察

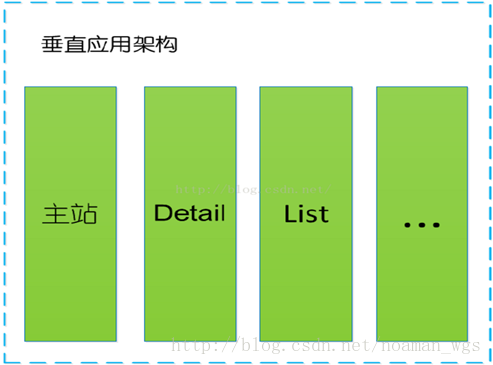
客户端可以在节点上设置watch，我们称之为监视器。当节点状态发生改变时(Znode的增、删、改)将会触发watch所对应的操作。当watch被触发时，ZooKeeper将会向客户端发送且仅发送一条通知，因为watch只能被触发一次，这样可以减少网络流量。

## 框架演变

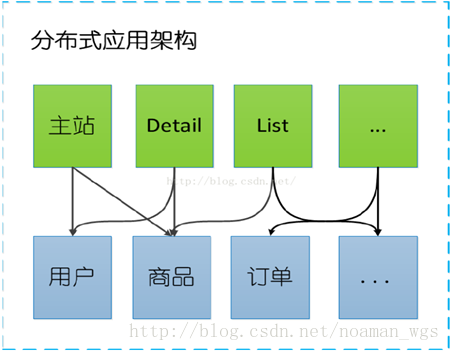
1. 单一应用框架(ORM)   
   当网站流量很小时，只需一个应用，将所有功能如下单支付等都部署在一起，以减少部署节点和成本。   
   **缺点**：单一的系统架构，使得在开发过程中，占用的资源越来越多，而且随着流量的增加越来越难以维护



1. 垂直应用框架(MVC)   
   垂直应用架构解决了单一应用架构所面临的扩容问题，流量能够分散到各个子系统当中，且系统的体积可控，一定程度上降低了开发人员之间协同以及维护的成本，提升了开发效率。   
   **缺点**：但是在垂直架构中相同逻辑代码需要不断的复制，不能复用。



1. 分布式应用架构(RPC)   
   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



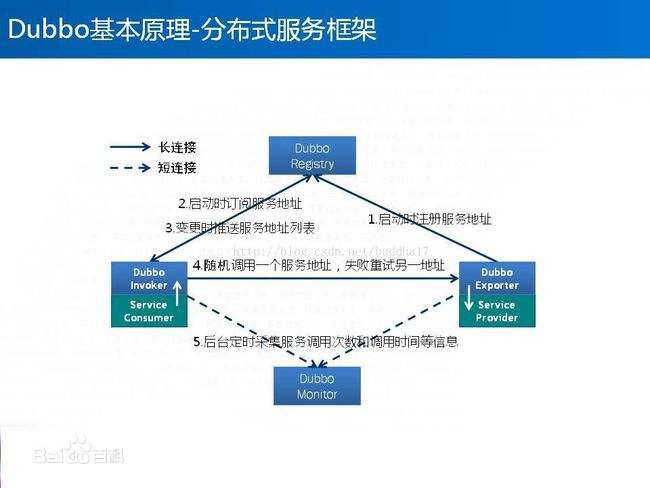
4. 流动计算架构(SOA)   
随着服务化的进一步发展，服务越来越多，服务之间的调用和依赖关系也越来越复杂，诞生了面向服务的架构体系(SOA)，也因此衍生出了一系列相应的技术，如对服务提供、服务调用、连接处理、通信协议、序列化方式、服务发现、服务路由、日志输出等行为进行封装的服务框架

## RPC

### Dubbo

<http://dubbo.apache.org/zh-cn/>

* 一款分布式服务框架
* 高性能和透明化的RPC远程服务调用方案
* SOA服务治理方案



### spring cloud

Spring Cloud是一个微服务框架,相比Dubbo等RPC框架, Spring Cloud提供的全套的分布式系统解决方案。

## 消息中间件

#### 1. 系统解耦

交互系统之间没有直接的调用关系，只是通过消息传输，故系统侵入性不强，耦合度低。

#### 2. 提高系统响应时间

例如原来的一套逻辑，完成支付可能涉及先修改订单状态、计算会员积分、通知物流配送几个逻辑才能完成；通过MQ架构设计，就可将紧急重要（需要立刻响应）的业务放到该调用方法中，响应要求不高的使用消息队列，放到MQ队列中，供消费者处理。

#### 3.为大数据处理架构提供服务

通过消息作为整合，大数据的背景下，消息队列还与实时处理架构整合，为数据处理提供性能支持。

### Kafka

不支持事务，对消息的重复、丢失、错误没有严格要求，适合产生大量数据的互联网服务的数据收集业务。

日志收集、行情数据接收

### RocketMQ

阿里开源的分布式消息系统，基于高可用分布式集群技术，提供低延时的、高可靠的消息发布与订阅服务。同时，广泛应用于多个领域，包括异步通信解耦、企业解决方案、金融支付、电信、电子商务、快递物流、广告营销、社交、即时通信、移动应用、手游、视频、物联网、车联网等。

### Redis的发布/订阅

多用于实时性较高的消息推送，并不保证可靠。

## 搜索引擎

### ElasticSearch

<https://www.elastic.co/cn/>

即时搜索服务器,也是数据存储

正在将文章、资讯板块向ElasticSearch迁移