# 引言

随着访问量的提升，使用关系型数据库的Web站点多多少少都开始在性能上出现了一些瓶颈，而瓶颈的源头一般是在磁盘的I/O上。导致在当今云计算、大数据盛行的时代，对性能有了更多的需求，主要体现在以下四个方面：

1. 低延迟的读写速度：应用快速地反应能极大地提升用户的满意度
2. 支撑海量的数据和流量：对于搜索这样大型应用而言，需要利用PB级别的数据和能应对百万级的流量
3. 大规模集群的管理：系统管理员希望分布式应用能更简单的部署和管理
4. 庞大运营成本的考量：IT部门希望在硬件成本、软件成本和人力成本能够有大幅度地降低

为了克服这一问题，NoSQL应运而生，它同时具备了高性能、可扩展性强、高可用等优点，受到广泛开发人员和仓库管理人员的青睐

# 是什么

Redis是现在最受欢迎的NoSQL数据库之一，Redis是一个使用ANSI C编写的开源、包含多种数据结构、支持网络、基于内存、可选持久性的键值对存储数据库，其

## 具备如下特性：

1. 基于内存运行，性能高效
2. 支持分布式，理论上可以无限扩展
3. key-value存储系统
4. 开源的使用ANSIC语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API

## 相比于其他数据库类型，Redis具备的特点是：

1. C/S通讯模型
2. 单进程单线程模型
3. 丰富的数据类型
4. 操作具有原子性
5. 持久化
6. 高并发读写
7. 支持lua脚本

## 哪些大厂在使用Redis？

1. Github
2. Twitter
3. 微博
4. Stack Overflow
5. 阿里巴巴
6. 百度
7. 美团
8. 搜狐

## Redis的应用场景有哪些？

Redis 的应用场景包括：缓存系统（“热点”数据：高频读、低频写）、计数器、消息队列系统、排行榜、社交网络和实时系统。

## Redis的数据类型及主要特性

Redis提供的数据类型主要分为5种自有类型和一种自定义类型，这5种自有类型包括：

String类型、

哈希类型、

列表类型、

集合类型和

顺序集合类型。

### String类型：

它是一个二进制安全的字符串，意味着它不仅能够存储字符串、还能存储图片、视频等多种类型, 最大长度支持512M。

对每种数据类型，Redis都提供了丰富的操作命令如：

GET/MGET

SET/SETEX/MSET/MSETNX

INCR/DECR

GETSET

DEL

### 哈希类型：

该类型是由field和关联的value组成的map。其中，field和value都是字符串类型的。

Hash的操作命令如下：

HGET/HMGET/HGETALL

HSET/HMSET/HSETNX

HEXISTS/HLEN

HKEYS/HDEL

HVALS

### 列表类型：

该类型是一个插入顺序排序的字符串元素集合, 基于双链表实现。

List的操作命令如下：

LPUSH/LPUSHX/LPOP/RPUSH/RPUSHX/RPOP/LINSERT/LSET

LINDEX/LRANGE

LLEN/LTRIM

### 集合类型：

Set类型是一种无顺序集合, 它和List类型最大的区别是：集合中的元素没有顺序, 且元素是唯一的。

Set类型的底层是通过哈希表实现的，其操作命令为：

SADD/SPOP/SMOVE/SCARD

SINTER/SDIFF/SDIFFSTORE/SUNION

Set类型主要应用于：在某些场景，如社交场景中，通过交集、并集和差集运算，通过Set类型可以非常方便地查找共同好友、共同关注和共同偏好等社交关系。

### 顺序集合类型：

ZSet是一种有序集合类型，每个元素都会关联一个double类型的分数权值，通过这个权值来为集合中的成员进行从小到大的排序。与Set类型一样，其底层也是通过哈希表实现的。

ZSet命令：

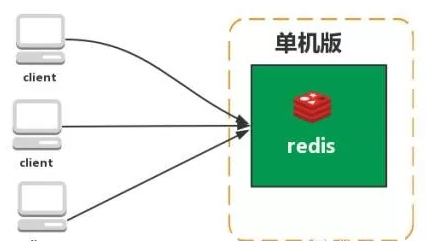
ZADD/ZPOP/ZMOVE/ZCARD/ZCOUNT

ZINTER/ZDIFF/ZDIFFSTORE/ZUNION

## Redis的数据结构

## 架构模式

### 单机版



1、内存容量有限 2、处理能力有限 3、无法高可用。

### 主从复制

Redis 的复制（replication）功能允许用户根据一个 Redis 服务器来创建任意多个该服务器的复制品，其中被复制的服务器为主服务器（master），而通过复制创建出来的服务器复制品则为从服务器（slave）。只要主从服务器之间的网络连接正常，主从服务器两者会具有相同的数据，主服务器就会一直将发生在自己身上的数据更新同步 给从服务器，从而一直保证主从服务器的数据相同。

特点：

1、master/slave 角色

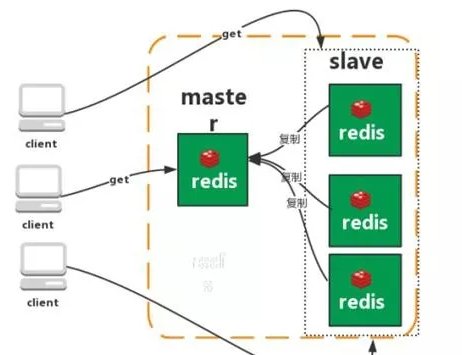
2、master/slave 数据相同

3、降低 master 读压力在转交从库

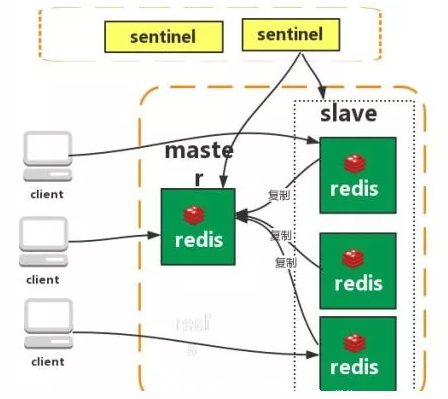
缺点：

无法保证高可用

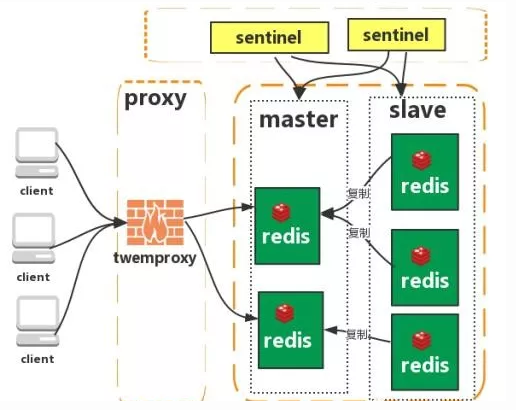
没有解决 master 写的压力



### 哨兵



### 集群（proxy 型）



Twemproxy 是一个 Twitter 开源的一个 redis 和 memcache 快速/轻量级代理服务器； Twemproxy 是一个快速的单线程代理程序，支持 Memcached ASCII 协议和 redis 协议。

特点：

1、多种 hash 算法：MD5、CRC16、CRC32、CRC32a、hsieh、murmur、Jenkins

2、支持失败节点自动删除

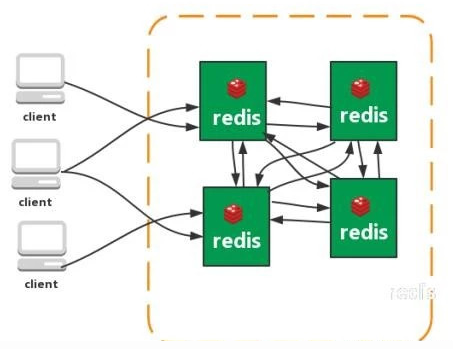
3、后端 Sharding 分片逻辑对业务透明，业务方的读写方式和操作单个 Redis 一致

缺点：

增加了新的 proxy，需要维护其高可用。

failover 逻辑需要自己实现，其本身不能支持故障的自动转移可扩展性差，进行扩缩容都需要手动干预

### 集群（直连型）



从redis 3.0之后版本支持redis-cluster集群，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。

特点：

1、无中心架构（不存在哪个节点影响性能瓶颈），少了 proxy 层。

2、数据按照 slot 存储分布在多个节点，节点间数据共享，可动态调整数据分布。

3、可扩展性，可线性扩展到 1000 个节点，节点可动态添加或删除。

4、高可用性，部分节点不可用时，集群仍可用。通过增加 Slave 做备份数据副本

5、实现故障自动 failover，节点之间通过 gossip 协议交换状态信息，用投票机制完成 Slave到 Master 的角色提升。

缺点：

1、资源隔离性较差，容易出现相互影响的情况。

2、数据通过异步复制,不保证数据的强一致性

## Redis核心概念

引用：<https://blog.csdn.net/gongpulin/article/details/80551438>

## 集群

集群教程：<http://doc.redisfans.com/topic/cluster-tutorial.html>

集群规范：<http://doc.redisfans.com/topic/cluster-spec.html>

# 一级标题

## 二级标题

### 三级标题

#### 四级标题

#### Xxx

# 特殊问题

## 击穿

这个跟缓存雪崩有点像，但是又有一点不一样，缓存雪崩是因为大面积的缓存失效，打崩了DB，而缓存击穿不同的是缓存击穿是指一个Key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个Key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个完好无损的桶上凿开了一个洞。

解决：

从缓存取不到的数据，在数据库中也没有取到，这时也可以将对应Key的Value对写为null、位置错误、稍后重试这样的值具体取啥问产品，或者看具体的场景，缓存有效时间可以设置短点，如30秒（设置太长会导致正常情况也没法使用）。

这样可以防止攻击用户反复用同一个id暴力攻击，但是我们要知道正常用户是不会在单秒内发起这么多次请求的，那网关层Nginx本渣我也记得有配置项，可以让运维大大对单个IP每秒访问次数超出阈值的IP都拉黑。

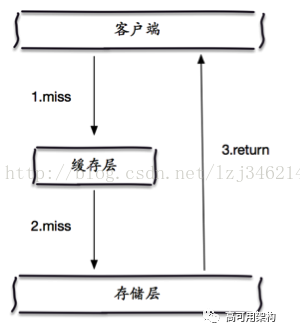
## 穿透

缓存穿透是指缓存和数据库中都没有的数据，而用户不断发起请求这时的用户很可能是攻击者，攻击会导致数据库压力过大，严重会击垮数据库

一般的缓存系统，都是按照key值去缓存查询，如果不存在对应的value，就应该去DB中查找 。这个时候，如果请求的并发量很大，就会对后端的DB系统造成很大的压力。这就叫做缓存穿透。关键词：缓存value为空；并发量很大去访问DB。

解决：

接口层增加校验，比如用户鉴权校验，参数做校验，不合法的参数直接代码Return



## 雪崩

**由于redis设置失效时间，当所有key在同一时间失效，并发直接打入DB，导致DB崩溃。**

举个简单的例子：如果所有首页的Key失效时间都是12小时，中午12点刷新的，我零点有个秒杀活动大量用户涌入，假设当时每秒 6000 个请求，本来缓存在可以扛住每秒 5000 个请求，但是缓存当时所有的Key都失效了。此时 1 秒 6000 个请求全部落数据库，数据库必然扛不住，它会报一下警，真实情况可能DBA都没反应过来就直接挂了。此时，如果没用什么特别的方案来处理这个故障，DBA 很着急，重启数据库，但是数据库立马又被新的流量给打死了。这就是我理解的缓存雪崩。

我刻意看了下我做过的项目感觉再吊的都不允许这么大的QPS直接打DB去，不过没慢SQL加上分库，大表分表可能还还算能顶，但是跟用了Redis的差距还是很大。

解决：

处理缓存雪崩简单，在批量往Redis存数据的时候，把每个Key的失效时间都加个随机值就好了，这样可以保证数据不会在同一时间大面积失效，我相信，Redis这点流量还是顶得住的。

如果Redis是集群部署，将热点数据均匀分布在不同的Redis库中也能避免全部失效的问题，不过本渣我在生产环境中操作集群的时候，单个服务都是对应的单个Redis分片，是为了方便数据的管理，但是也同样有了可能会失效这样的弊端，失效时间随机是个好策略。

或者设置热点数据永远不过期，有更新操作就更新缓存就好了（比如运维更新了首页商品，那你刷下缓存就完事了，不要设置过期时间），电商首页的数据也可以用这个操作，保险。

## 在生产环境中最好不能keys命令

引用：<https://blog.csdn.net/qq_39669058/article/details/107647688>

### 背景

#### Redis是单线程的，其所有操作都是原子的，不会因并发产生数据异常

#### 使用高耗时的Redis命令是很危险的，会占用唯一的一个线程的大量处理时间，导致所有的请求都被拖慢

### 场景

在生产环境中执行 KEYS 命令的时，因为Redis是单线程的，KEYS 命令的性能随着数据库数据的增多而越来越慢，使用 KEYS 命令时会占用唯一的一个线程的大量处理时间，引发Redis阻塞并且增加Redis的CPU占用，导致所有的请求都被拖慢，可能造成Redis所在的服务器宕机，情况是很恶劣的，在实际生产运用的过程中请忽略掉

如果所有的线程在Redis那取不到数据，情况严重时可能会导致应用程序出现雪崩的情况，一瞬间全去数据库取数据，数据库就宕机了。

### 其他危险（复杂度同样是O（N））命令

#### flushdb 命令用于清空当前数据库中的所有 key

#### flushall 命令用于清空整个 Redis 服务器的数据(删除所有数据库的所有 key )

#### config 客户端连接后可配置服务器

### 禁用以上危险的命令

rename-command FLUSHALL "" （此命令还需要设置配置文件中appendonly no，否则服务器是无法启动）

rename-command CONFIG ""

rename-command FLUSHDB ""

rename-command KEYS ""

### 改良方式

使用scan命令替代

Redis从2.8版本开始支持scan命令，可以用来分批次扫描Redis记录，这样肯定会导致整个查询消耗的总时间变大，影响服务使用，但不会影响redis服务卡顿。

Java中MyBlog项目中的RedisService已经实现了key方法，其内部使用的是scan命令对key进行模糊查找。

Scan的使用方式

命令格式 SCAN cursor [MATCH pattern] [COUNT count]

SCAN 命令用于迭代当前数据库中的数据库键。

SSCAN 命令用于迭代集合键中的元素。

HSCAN 命令用于迭代哈希键中的键值对。

ZSCAN 命令用于迭代有序集合中的元素（包括元素成员和元素分值）。

scan 0 从零开始查 返回结果会给一个游标数，下一条命令就可以这样写：

scan游标数，意思是接着下标往下找，最后返回游标数为0则查找结束，以此达到分批查找。

# 一级标题

## 二级标题

### D

### d

#### 四级标题

#### Xxx

引用<https://www.cnblogs.com/powertoolsteam/p/redis.html>

引用<https://blog.csdn.net/tjcyjd/article/details/78463056?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-7.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-7.channel_param>