# Redis复习

## 使用场景：

高并发场景下，Mysql 的I/O会暴涨，数据页命中率降低，频繁的淘汰页，加载新的页；所以加一个缓存，把**经常使用**的数据搞个缓存。

## Redis基础：

### 通用知识：

1:redis是二进制安全的

2:reids处理请求是单线程

3:redis支持负向索引

### 基础命令：

select：选择分区

help：

help @string

help set

help @generic

keys:列举所有的key

flushall/flushdb：清库

ttl：查看过期时间

expire/expireat：给key 设置过期时间(s)/设置到时间戳过期

type：查看value的类型(string/list/hash/set/zset)

object encoding：查看value的编码(对于string：int/embstr/raw)

config get \*：获取所有的配置

config set key value：设置配置项

### 常用数据类型：

#### string/int

1:set key value [ex|px] [nx|xx] 设置string value，过期时间(也可以使用expire)

2:get key 获取key对应的value

3:mset/msetnx 批量设置

4:strlen 字符串长度

5:append：拼接字符串

6:setrange key offset value

7:getrange key start end

8:decr key/decrby key num

9:incr key/incr key num

10:incrbyfloat key num

#### bitmap

1:setbit key offset value 给对应的offset设置值

2:getbit key offset 获取对应offset的值

3:bitpos key value start end 获取字节在start和end之间的 第一个是value的position

4:bitcount key start end 获取key在start 和end字节之间的，1的个数

5:bitop or/and dest K1 K2 对两个key做运算，放到dest中去

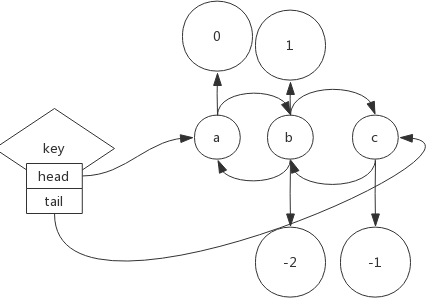
练手：

1:设置A，B 运算出C

2:记录一个人一年内都在哪些天登陆了（366/8个Byte）（key为user，value每天登陆信息）

3:记录某几天几天登陆的人数（key为日期，value为对应的用户是否登陆）

#### list



lpush/rpush key value：添加元素

lpop/rpop key value：取出元素

lset key offset value：修改对应下标值

lindex key offset：查看对应下标值

lrange key start end：查看范围值

lrem key count value：删除 count个value

count=0，所有

count>0 从左往右删除

count<0 从右往左删除

Ltrim key start end:让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。

Blpop/brpop：阻塞式获取

#### Hash

Hset key [hkey value]：批量设置

Hget key hkey：单个获取

Hmget key [hkeys]：批量获取

Hkeys key：获取所有key

Hvals key：获取所有的value

Hgetall key：获取所有key value

Hincryby key hkey value: 给对应的key+num

Hincrybyfloat key hkey num:给对应的key+num

#### Set

Sadd key [values]：批量添加

Srem key [values]：批量删除

Spop key：随机弹出一个

Smembers key:返回所有的值（慎用！！！）

Srandmember key count：随机返回count个数据（count>0 则返回指定数目的，非重复的，返回数量最大为 集合总数。 Count< 0,返回指定数据的，可以重复的数据）

集合运算：

sinter [keys]: 交集

sinterstore dest [keys]：交集，且存到dest

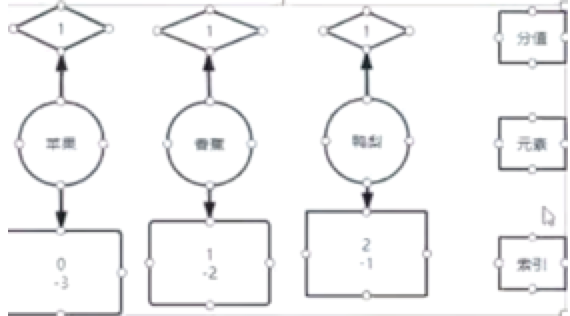
sunion [keys]：并集

sunionstore dest [keys]：并集，且存到dest

sdiff k1 k2：差集

sdiffstore dest k1 k2：差集，且存到dest

#### zset



zadd key [sorce member]：批量添加

zrem key [members]：批量删除

zrange/revrange key start end [withscores]：返回指定排名范围的set结果集

zrangebyscore/zrevrangebyscore key score score：返回指定分数范围的set结果集

zscore key member：获取具体成员的分数

zrank key member：获取具体成员的排名

zincrby key num member：给某个member的score+num

集合操作：

zunionstore destkey [opnums] [keys] [weightValues] aggregate [sum|min|max]

zinterstore destkey [opnums] [keys] [weightValues] aggregate [sum|min|max]

### Key过期清除策略：

#### 几个小的知识点：

1:使用expire/expireat或者 set key value ex 可以设置过期时间；或者是续期

2:使用ttl查询过期时间

3:set key 会删除过期时间

4:get key不会延长过期时间

#### 过期key清楚策略：

##### 1:定时任务主动清除：

1-1:随机一批keys进行过期检测

1-2:删除已经过期的keys

1-3:如果删除的数量超过25%，那么重复1-1

##### 2:被动清除

1:访问该key时检测，如果过期，删除

2:在内存空间不够时，volatail\_lru淘汰算法会淘汰过期的key

## 淘汰策略：

### noeviction：

如果内存不够，那么报错

### LRU：

Allkeys\_lru：针对所有的键，释放最久没有使用的

Volatile\_lru：针对设置了过期的键，释放最久没有使用的

### LFU：

释放少使用的

背景：缓存数据不是全量数据，是**热点数据**，缓存应该随着访问变化热数据；因为redis存在内存中，**内存的大小有限**。

Redis都有哪些清理数据机制？

## Redis持久化：

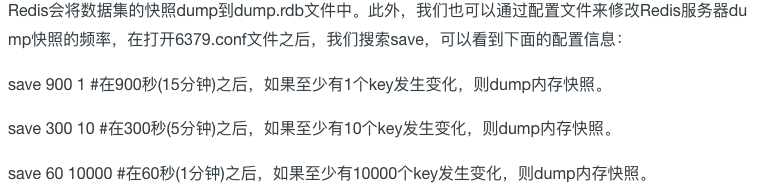
### RDB:

#### 特点：

定时全量落盘，类似快照，记录的是数据的二进制压缩；（数据库的全量物理备份）；子进程使用copy-on-write落盘；

#### 调用方式：

自动：



手动：

save：阻塞式

bgsave；非阻塞式

#### 缺点：

间隔时间久，丢的数据多

#### 优点：

恢复快

### AOF:

使用append-only开启

#### 特点：

使用追加的方式，记录执行的命令的文本信息，恢复时，重放一边命令既可恢复（类似statement格式的binlog）

#### 缺点：

实时落盘拖慢了redis的速度

#### 优点：

实时性好

#### 问题：

实时追加，导致文件会无限大

问题解决：

4.0之前，使用的命令抵消

4.0之后，使用rdb+aof混合模式选择：

#### 相关配置：

1:appendonly yes

2: appendfsync always/everysec/no

### 选择：

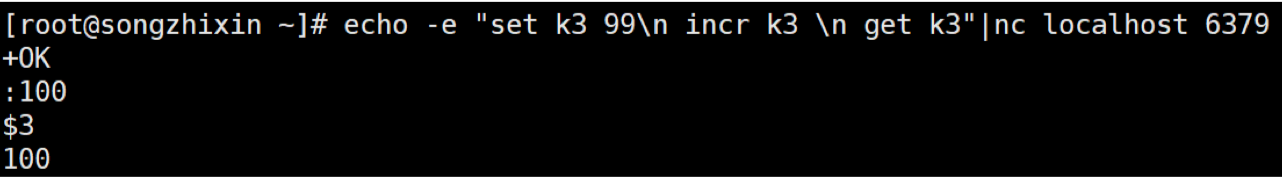
如果事实性要求高，同时开启，rdb+aof模式

如果事实性要求低，只开启rdb

## Redis高级特性：

### 管道：

背景：允许一次发送一批命令，然后结果也按照一批返回



### 订阅发布：



### 事务：

#### 相关命令：

1:mutli开启事务

2:exec 执行事务

3:watch 开启事务前监听key，如果key在执行事务前有变化，那么取消事务

#### 知识点：

1:只要exec没到，mutli后面的命令就缓存起来，直到exec到了才一股脑执行。

2:事务之间串行，谁的exec先到，谁先执行

## Redis分布式：

<https://segmentfault.com/a/1190000022808576>

## 单机问题：

单点故障

并发量

存储容量

## X轴：引入多机，进行主从

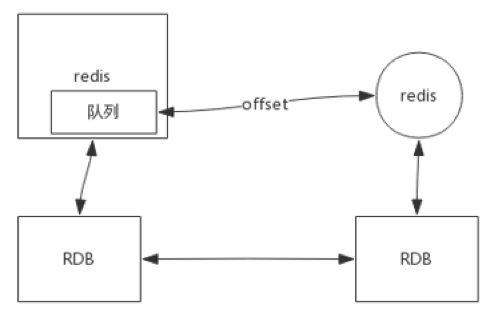
工作原理：

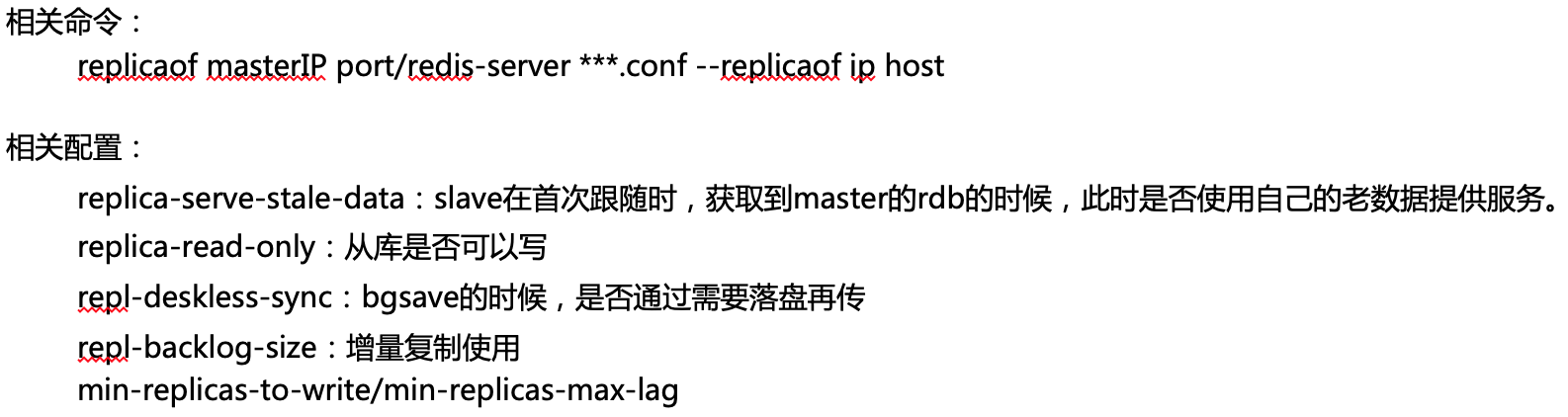
slave启动时，指定follow的主节点

主节点收到连接后，bgsave一个快照，发给follower

master收到客户端请求之后，异步增量同步给follower

slave重启之后，master根据offset给slave进行数据恢复





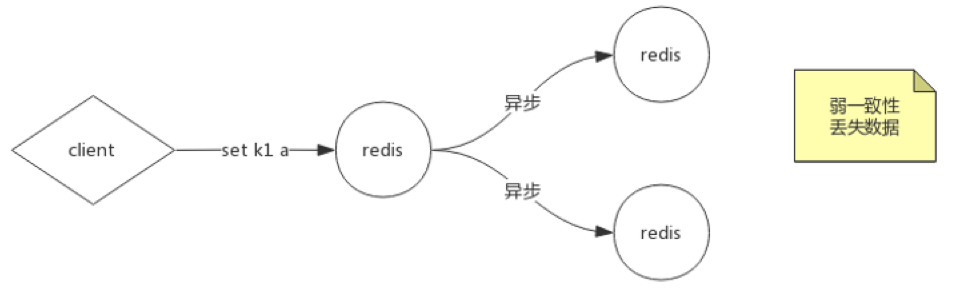
#### 新问题：1:主机HA；2:数据一致性

### 1：主机HA：

引入哨兵，见哨兵章节

### 2：数据一致性：

redis使用异步的同步方式，此方式注重于速度，但是会有数据丢失的问题。当数据还没有同步到从节点时，主节点挂掉，主从切换，那么数据就会丢失。



### Y轴：业务切分

就是每个业务，使用自己的redis集群

### Z轴：业务内部切分

1: client端实现hash数据到 多个独立的redis主从中（使用环形hash）

问题：client实现复杂+需要和每个结群连接，连接数过多

2:代理；

2-1代理实现 环形hash+哨兵模式，并且减少了client和server的连接

2-1 封装cluster模式，并且减少了client和server的连接

3:cluster模式

## Redis哨兵：

使用一群哨兵来监控 master的HA状态，发现下线，那么进行切换。

#### 前提知识：

1: 配置哨兵时，只配置主节点信息，主节点知道从节点信息，所以哨兵可以从主节点拿到从节点信息

2:哨兵之间通过 redis集群的订阅发布机制进行发现

#### 主要流程：

1:哨兵定时向master发送ping命令，在指定时间内没反回，主观认定下线、

2:当足够数量的哨兵发现master失效，客观认为下线

3:哨兵中使用raft算法选择一个 leader，去进行主从切换

4:选择offset最新的那个节点，主从切换

#### Raft选择过程：

1:每次选举，不管是否成功 epoch 加1

2:每次epoch，一个哨兵只能投一票

3:每个哨兵收到选举请求的时候，比较自身的纪元，如果比自己大，那么投票

4:超过半数投票称为leader

## Redis集群/代理：

<https://wylong.top/redis/10-Redis%E9%AB%98%E5%8F%AF%E7%94%A8%E9%9B%86%E7%BE%A4-Cluster%E6%A8%A1%E5%BC%8F.html>

解决Z轴问题的第三种方式

### 前提知识：

1: 每个集群都有16384个槽位

2:集群由多个独立单机组成，集群根据 参数，设置组件master和slave

3:cluster会分配槽位

4:通过对key进行crc16%16384

5:可以人工迁移槽位，添加节点，或者删除节点（rehash）

5-1:移动槽位，删除节点

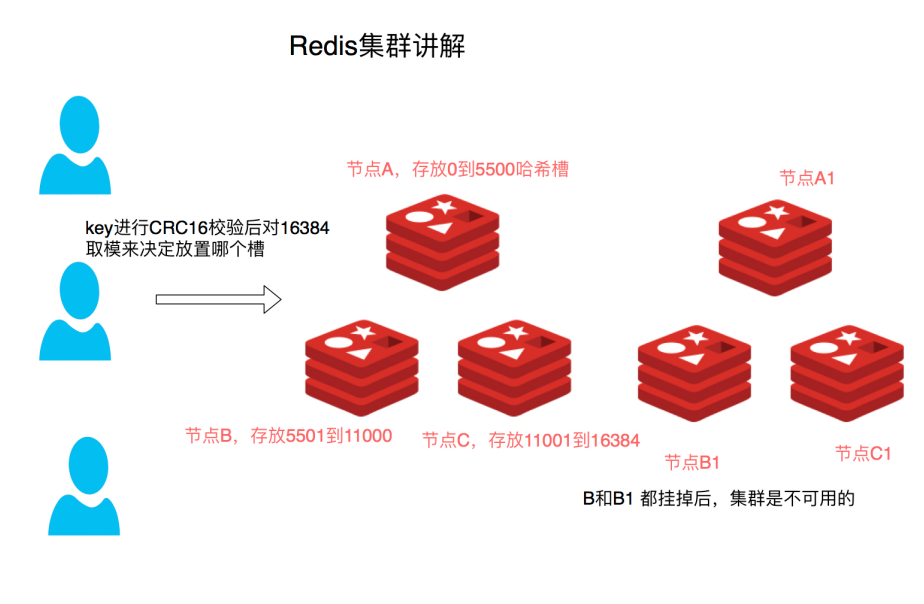
5-2:添加节点，移动槽位

### master的主从问题：

单点故障：每个master都有slave

主从一致性：弱一致

master HA：master节点直接raft



# Redis常见名词

### 缓存击穿：

###### 问题原因：

**一个** 会被**大量访问的key** **过期**之后，会把大量请求压到数据库上。

###### 解决办法：

更新数据前，写一个null到key，然后进行加分布式锁，只有一个请求访问到数据库上，访问完成之后，更新key；

其他请求获取null之后，过一会再来获取。

### 缓存穿透：

###### 问题原因：

client一直查询**一个不存在的数据**，而且**并发量很大**，造成数据库压力

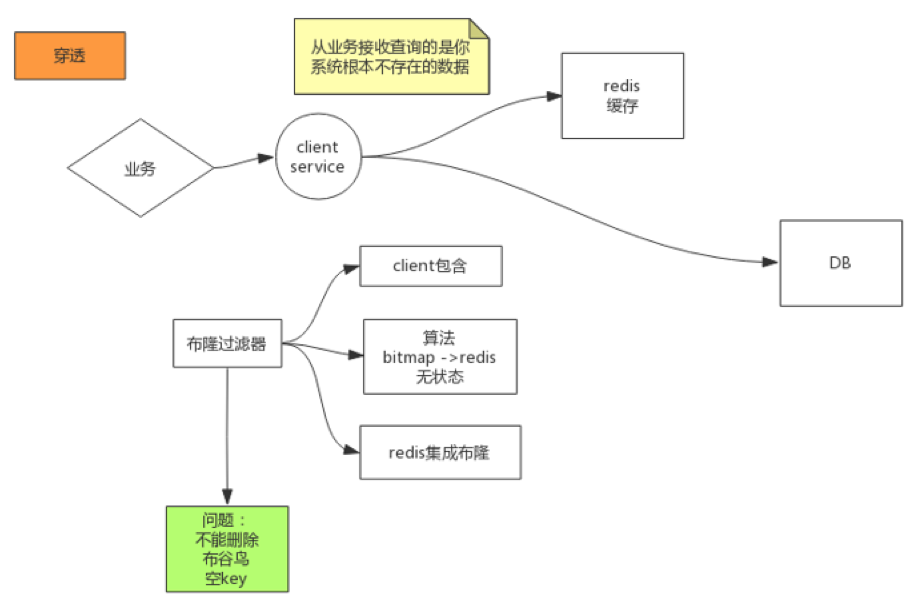
###### 解决办法：

访问数据库之前，使用布隆过滤器查询是否存在，如果存在，才放行

###### 原理：

使用bitmap，插入数据库数据时，hash 几个offset，给bitmap设置为1

###### 问题：布隆过滤器不能删除，需要使用布谷鸟过滤器



### 雪崩：

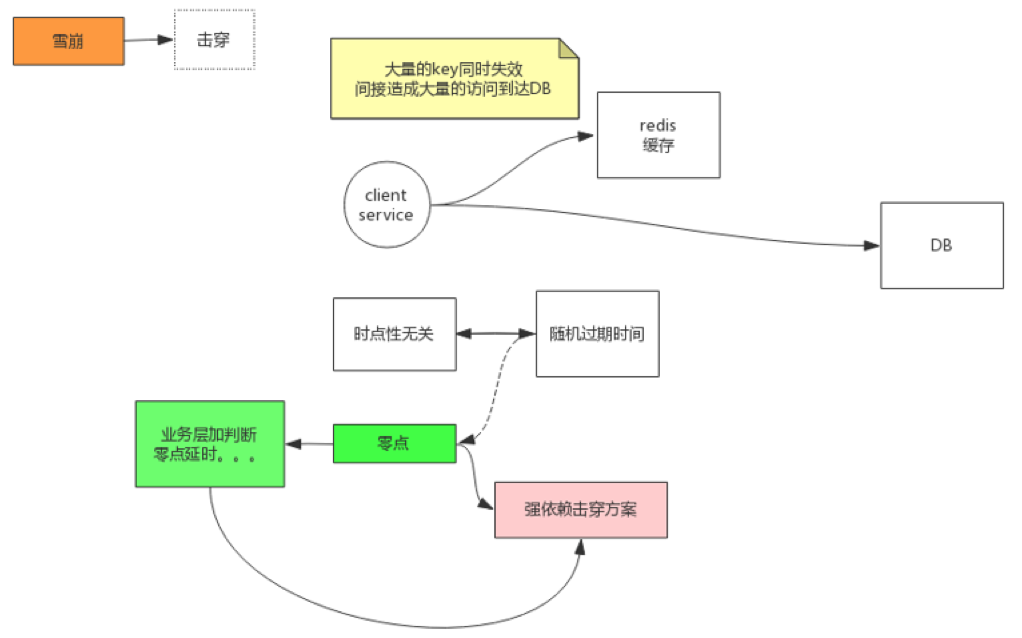
###### 问题原因：

大量的key 同时过期，导致大量请求打到mysql上，导致mysql压力

###### 解决办法：

批量设置key的过期时间的时候，不要设置成一样。避免同时过期

如果必须是0点，那么使用缓存击穿的方案



### 分布式锁：

1:set ex nx

2:一个线程定时维护过期时间

问题：不稳定，主从切换可能会导致数据丢失，从而导致同时获取到锁

tradeoff：可以使用zookeeper完成锁，但是性能比较差，redis还是比较快的，最好还是使用业务做等幂来保证把。

Redis：常见面试题

Keys \*

Redis底层数据结构

Redis zset 跳表

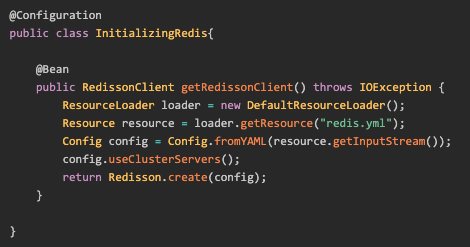
### Redission:

<https://www.jianshu.com/p/e9b26c743cae>

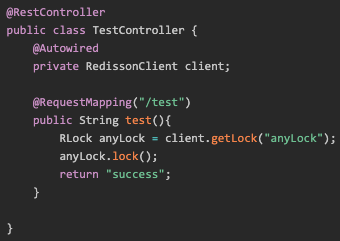
1:Resission配置文件



2:定义RedissionClient



3:创建锁，注意需要加 name



防止挂掉死锁的问题：

Watchdog，一直维护时间

### Redis锁存在的问题：

<https://blog.csdn.net/qq_38238041/article/details/115273101>

https://www.cnblogs.com/crazymakercircle/p/14731826.html

#### 单节点加锁：

Set nx + 线程维护的方式

#### 主从节点加锁：

问题1：如果主从切换，并且锁的数据还没有同步，那么可能会重复获取锁

解决：使用多个节点来搞，获取超过一半的锁，才算获取到真正获取到

问题2：重启问题，一个线程，拿到了三个锁，第三台被重启了  
然后第二个线程，拿到了3，4，5的锁，也加锁成功，这就是红锁的问题  
解决方案：第三台—挂了的，延迟启动，24H之后再重启(时间短一点也可以，比如理论上几分钟就好了)

**所以，最好使用单Redis来当锁。不然会有前面两个问题的。**

#### Java使用三方锁的共通问题：

获取锁之后，stw了，然后锁超时了，那么watchDog也不能工作了。那么此时redis中的锁就会被释放了，但是stw恢复之后呢，之前获取锁的线程还是会往下走。同时其他线程也可以获取到锁。

这个是JAVA的问题，对于类似的机制（需要维护时间，或者zk的e节点，需要有心跳的）都会有问题的。

##### 解决方案：阿里都不会这样用

ZK+Mysql 乐观锁

