1. **Redis介绍**

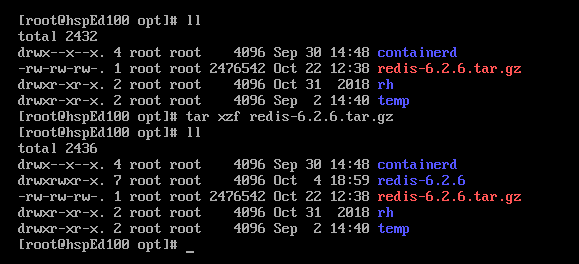
**Redis**（Remote Dictionary Server），即远程字典服务。

是一个开源的使用C语言编写，支持网络、可基于内存亦可持久化日志型，key-value数据库，redis会周期性的把更新的数据写入磁盘或者修改操作写入追加的记录文件，并且在次基础上实现了master-slave（主从）同步。

* **Redis可以做的东西**
* 内存存储、持久化，内存中是断电即失，所以说持久化很重要（rdb、aof）
* 效率高，可以用于高速缓存
* 发布订阅系统
* 地图信息分析
* 计时器、计数器（浏览量）

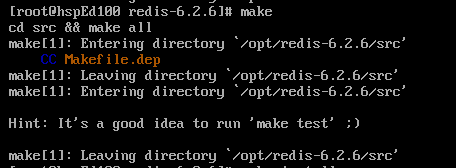
1. **安装**

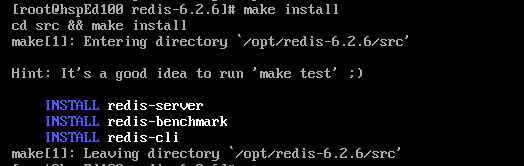
* **解压安装包**

****

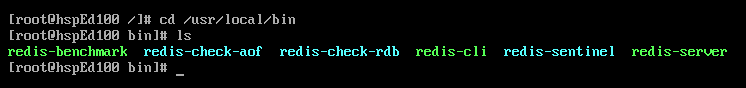
* **编译文件**

****

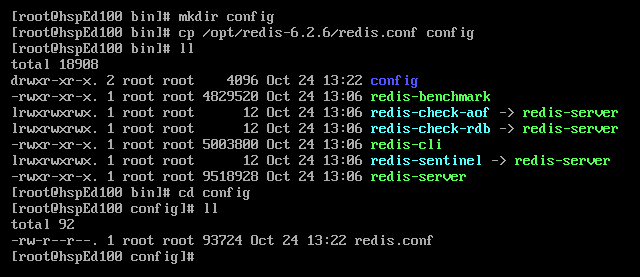
****

****

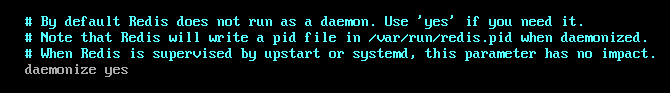
* **Redis默认安装路径**

****

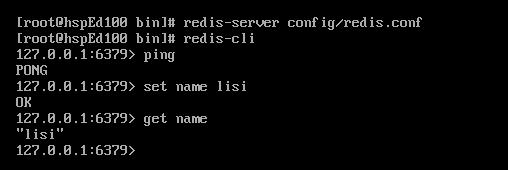
* **将redis配置文件复制到redis默认安装路径下**

****

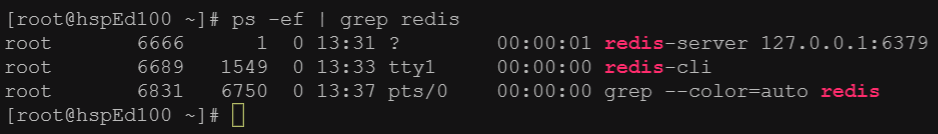
* **Redis默认不是后台启动的，需修改配置文件（daemonize为no改为yes）**

****

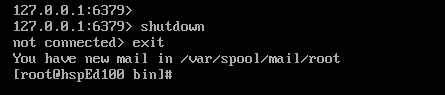
* **通过指定的配置文件启动Redis**

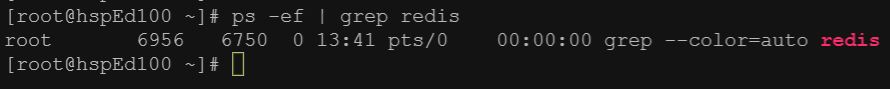
****

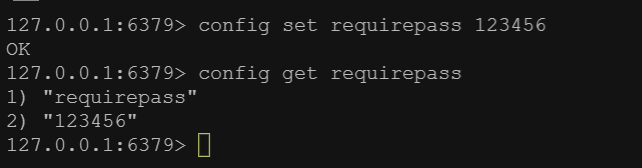
* **查看redis进程是否开启**

****

* **Redis服务关闭**

****

****

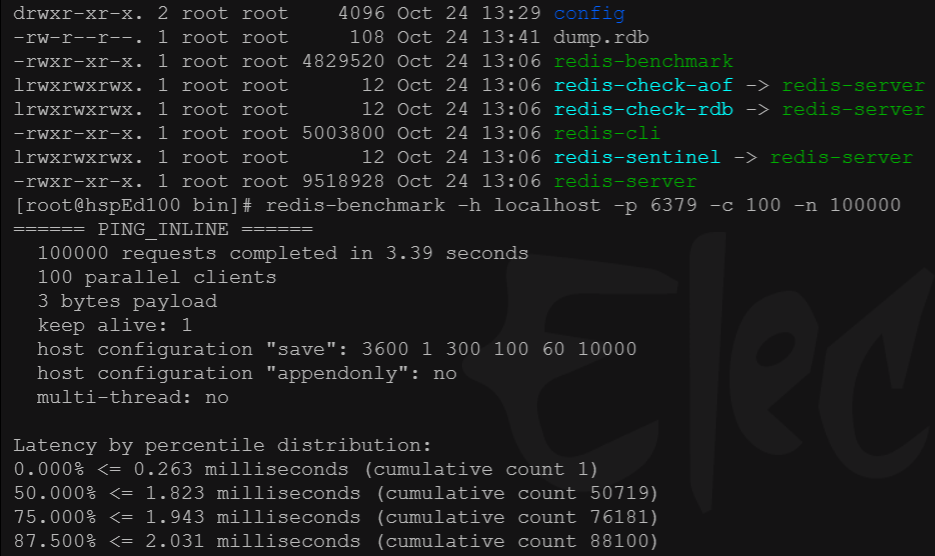
* **设置密码连接  
  **

1. **Redis性能测试**

可以使用**redis**自带工具（**redis-benchmark**压力测试工具）进行测试

|  |  |  |
| --- | --- | --- |
| **-h** | **服务器主机名** | **127.0.0.1** |
| **-p** | **端口** | **6379** |
| **-s** | **指定服务器socket** |  |
| **-c** | **并发连接数** | **60** |
| **-n** | **请求数** |  |
| **-d** | **以字节的形式指定set/get值得数据大小** | **10000** |
| **-k** | **1=keep alive 0=reconnect** | **2** |
| **-r** | **Set/get/INC使用随机key SADD使用随机值** | **1** |
| **-P** | **通过管道传输<numreq>请求** |  |
| **-q** | **强制退出redis，仅显示query/sec值** | **1** |
| **-csv** | **以CSV格式输出** |  |
| **-l** | **生成循环，永久执行测试** |  |
| **-t** | **仅运行以逗号分割得测试命令列表** |  |

* **测试100个并发连接，100000个请求**

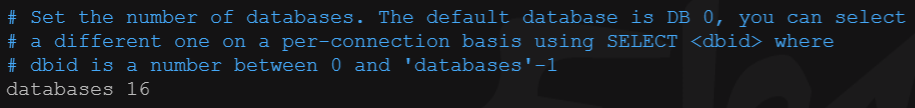


* **说明：**

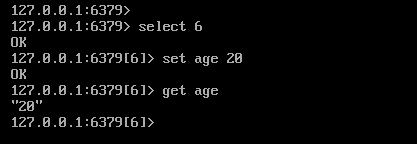
**100000**个请求用来**3.39**秒；**100**个并发得客户端；每次写入**3**个字节；只有**1**台服务器处理这些请求，单机性能

1. **Redis的基本知识**

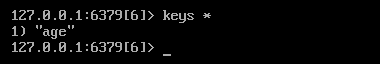
* **默认有16个数据库**

****

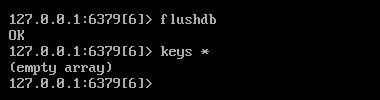
* **默认使用的第0个；可以使用select进行切换**

****

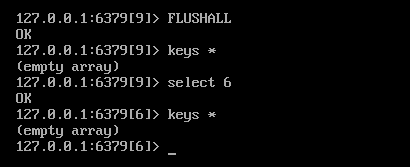
* **查看数据库所有的key**

****

* **清空当前数据库**

****

* **清空所有数据库**

****

* **Redis是单线程**

官方表示，redis是基于内存操作，CPU不是redis性能瓶颈，redis的瓶颈是根据机器内存和网络带宽，既然可以使用单线程来实现，就使用单线程了。

* **Redis为什么单线程还这么快**
* **误区1：**高性能的服务一定是多线程？
* **误区2：**多线程（CPU上下文会切换）一定比单线程效率高!

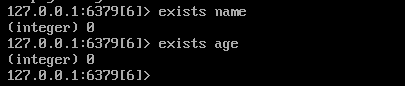
先去CPU>内存>硬盘的速度要有所了解

**核心：**

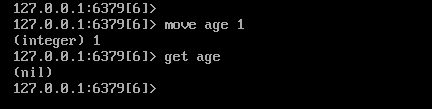
**Redis**是将所有的数据放在内存中，所有说使用单线程去操作效率就是最高的，多线程（**CPU上下文会切换**：耗时的操作），对于内存来说，如果没有**上下文切换**效率就是最高的！多次读写都是在一个**CPU**上的，在内存情况下这个就是最佳的方案

1. **Redis五大基本数据类型**

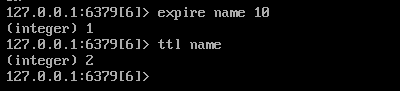
* **Key的基本命令**
  + - **keys \*：**查看所有的key
    - **exists key名：**判断key是否存在



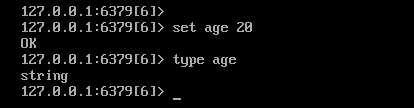
* + - **move key名 1：**移除当前的key



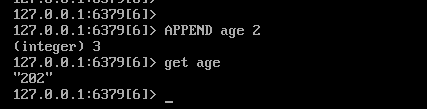
* + - **set key value：**设置key和值
    - **get key：**获取key值
    - **expire key 10：**设置当前key10秒过期
    - **ttl key：**查看key还有多长时间过期

****

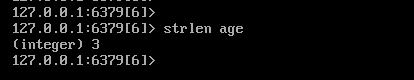
* + - **type key：**查看key值类型



* **String类型**
  + - **APPEND key “hello”：**追加字符串，如果当前key不存在，就相当于set key



* + - **获取字符长度**



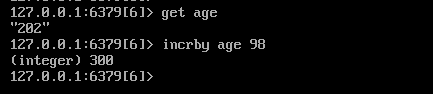
* + - **Key值自增加1（incr）**

****

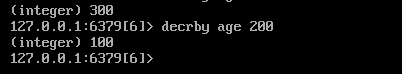
* + - **Key值自减1（decr）**

****

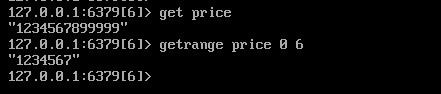
* + - **设置不长，指定增量**

****

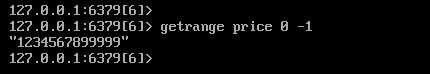
* + - **设置不长，指定减量**

****

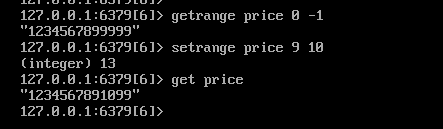
* + - **截取字符串**

****

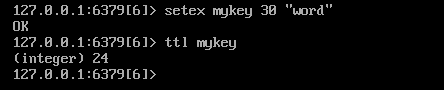
* + - **获取所有的字符串，和get key是一样的**

****

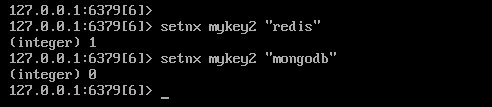
* + - **替换指定位置开始的字符串**

****

* + - **设置过期时间**

****

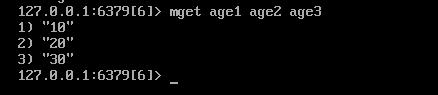
* + - **Key不存在创建，否则创建失败**

****

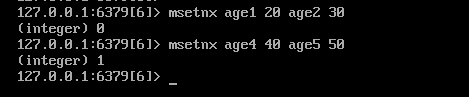
* + - **同时设置多个key-value**

****

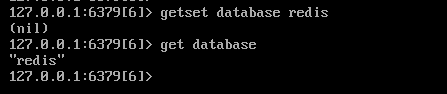
* + - **同时获取多个key**

****

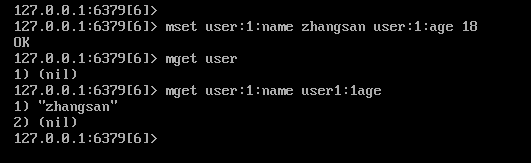
* + - **msetnx是一个原子性操作，要么一起成功，要么一起失败**

****

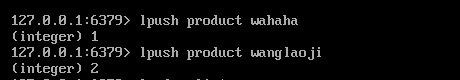
* + - **getset先获取再设置（不存在返回nil，存在返回值）**

****

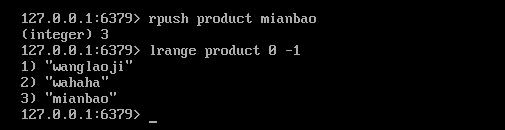
* + - **设置一个对象**

****

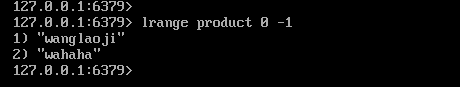
* **List类型**
  + - **将一个值插入到列表头部（左-lpush）**

****

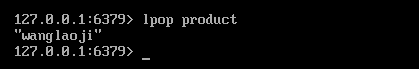
* + - **将一个值插入到列表尾部（右-rpush）**

****

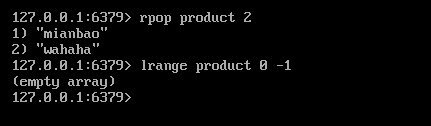
* + - **获取列表中的值（lrange）**

****

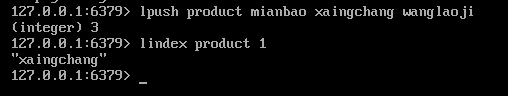
* + - **移除头部项（左-lpop）**

****

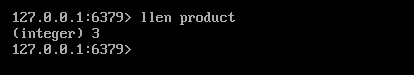
* + - **移除尾部项（右-rpop）**

****

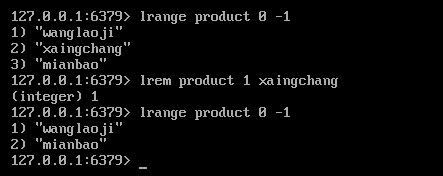
* + - **通过下标获取指定列表项（lindex）**

****

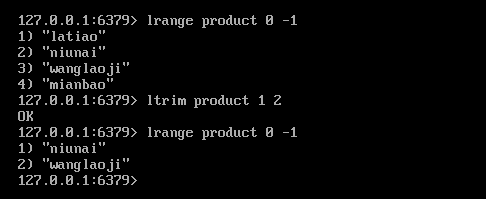
* + - **获取列表长度（llen）**

****

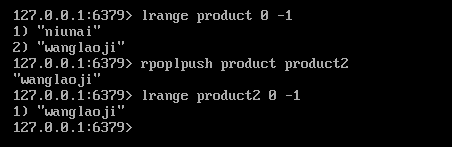
* + - **移除列表指定个数的value值（lrem）**

****

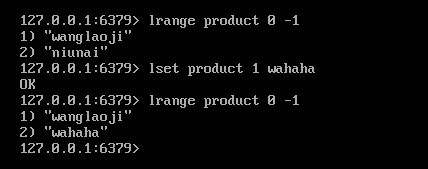
* + - **通过下标截取指定长度，被截取的list会发生改变，只剩下截取的元素（ltrim）**

****

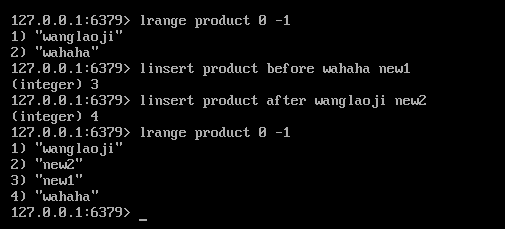
* + - **移除列表最后项，并移动到其它列表中（rpoplpush）**

****

* + - **更新指定列表下标的值（lset）**

****

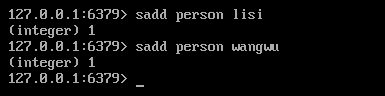
* + - **将某个value插入（linsert）到指定列的前面（before）或后面（after）**

****

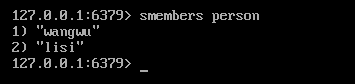
* **Set类型（key-value）**

注意：set中的值不能重复

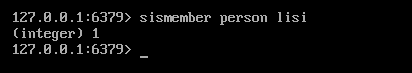
* **添加一个set元素（sadd）**

****

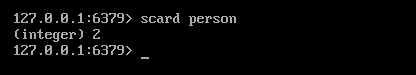
* **查看指定set的所有值（smembers）**

****

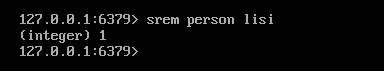
* **判断某个值是否在set集合中（sismember）**

****

* **获取set集合中元素个数（scard）**

****

* **移除set集合中的指定元素（srem）**

****

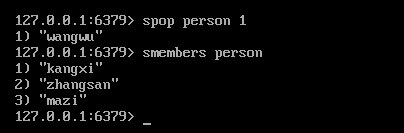
* **随机抽取出set集合元素（srandemember）**

****

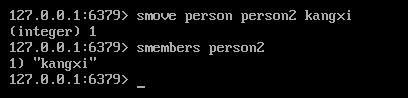
* **随机抽取出2个set集合元素**

****

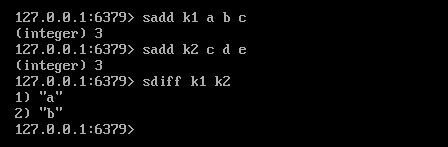
* **随机删除set集合元素**

****

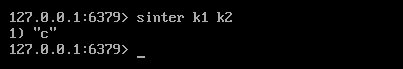
* **将一个指定set集合元素移除到指定set集合中**

****

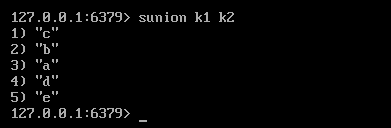
* **数字集合类**
* **差集**

****

* **交集**

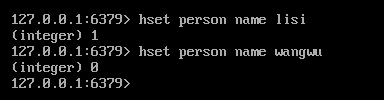
****

* **并集**

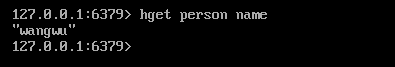
****

此功能类似于微博获取用户共同关注

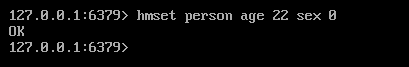
* **Hash类型（key-map(key-value)）**
* **插入map集合值**

****

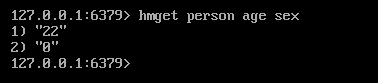
* **获取map集合值**

****

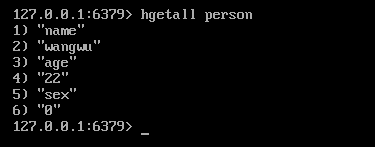
* **Set多个key-value**

****

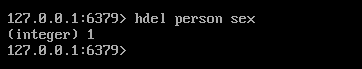
* **获取多个字段值（hmget）**

****

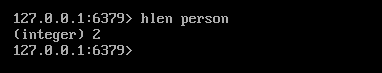
* **获取全部的数据（hgetall）**

****

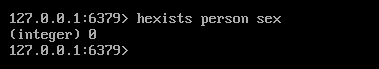
* **删除hash指定key字段（hdel）**

****

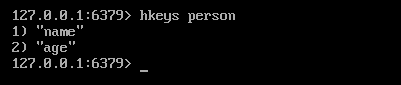
* **获取hash字段数量（hlen）**

****

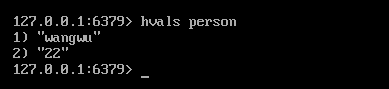
* **判断hash中的key是否存在（hexists）**

****

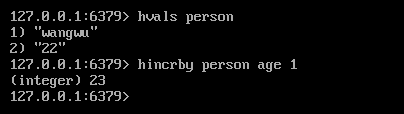
* **只获取hash的key（hkeys）**

****

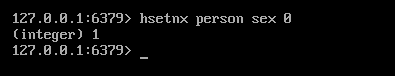
* **只获取hash的value（hvals）**

****

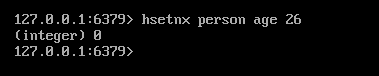
* **设置某个hash的key值指定增量加n（hincrby）**

****

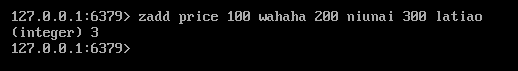
* **判断某个key是否存在，存在则设置**

****

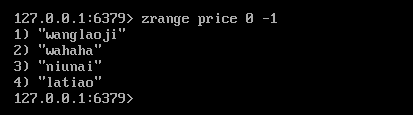
* **判断某个key是否存在，存在不设置**

****

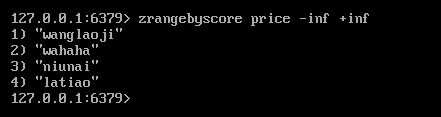
* **Zset（有序集合）**
* **添加多个元素**

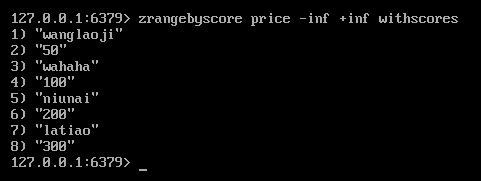
****

* **查看元素（zrange）**

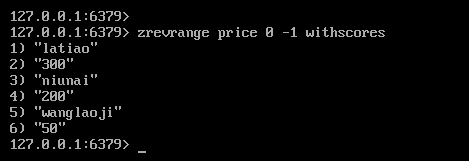
****

* **从小到大排序**

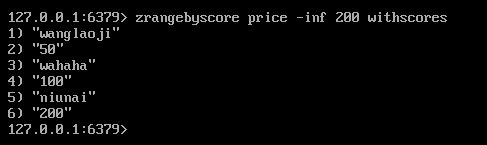
****

****

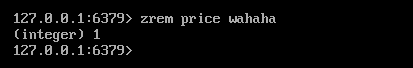
* **从大到小排序输出**

****

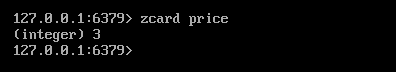
* **获取从指定大小之内的元素**

****

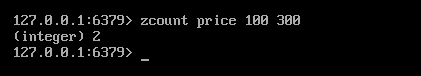
* **移除元素（zrem）**

****

* **获取有序集合中的个数（zcard）**

****

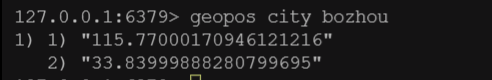
* **统计指定区间的值数量（zcount）**

****

* **Geospatial地理位置**
* **添加地址位置（geoadd）**

****

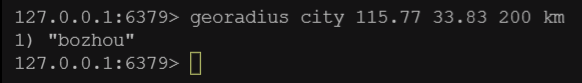
* **获取地址经纬度（geodist）**

****

* **获取两地之间的距离（geodist）**

****

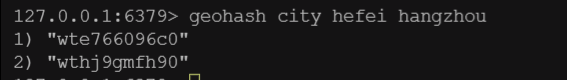
* **根据经纬度来获取指定范围内的地理位置集合（georadius）**

****

* **根据储存地点来获取指定范围内的地理位置集合（georadiusbymember）**

****

* **根据地点返回一个或多个位置元素geohash值（geohash）**

****

* **Hyperloglog**
* **添加**

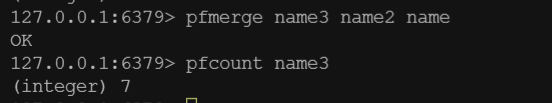
****

* **统计不重复的**

****

* **合并其它key的数据**

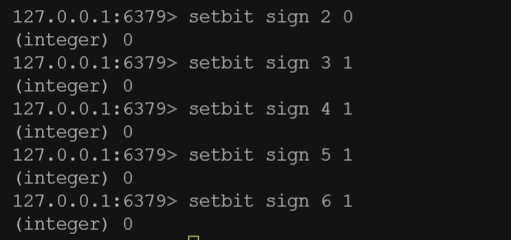
****

****

* **Bitmap位图**

**Bitmap位图，数据结构! 都是操作二进制位来进行记录，只有0和1两个状态**

* **添加打开记录**

****

* **查看某天打卡状态**

****

* **统计打卡天数**

****

**六、事务**

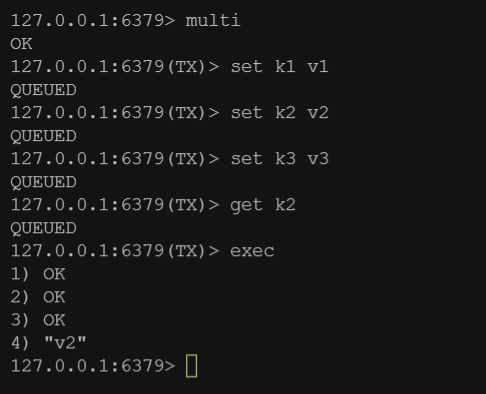
Redis事务本质：一组命令的集合，一个事务中的所有命令都会被序列化，在事务执行过程中，会按照顺序执行!

**一次性、顺序性、排查性!执行一些命令；**

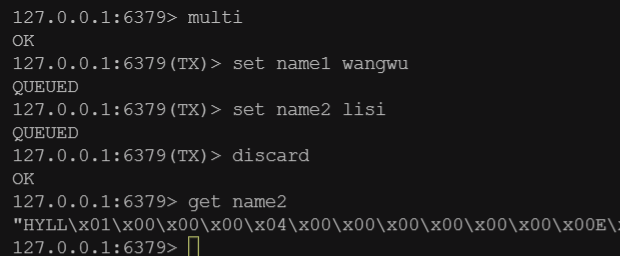
Redis事务没有隔离级别的概念；所有命令在事务中，并没有直接被执行!只有发起执行命令的时候才会执行

Redis单条命令式保存原子性的，但是事务不保证原子性！

* **开启事务（multi）**
* **命令入队**
* **执行事务（exec）**

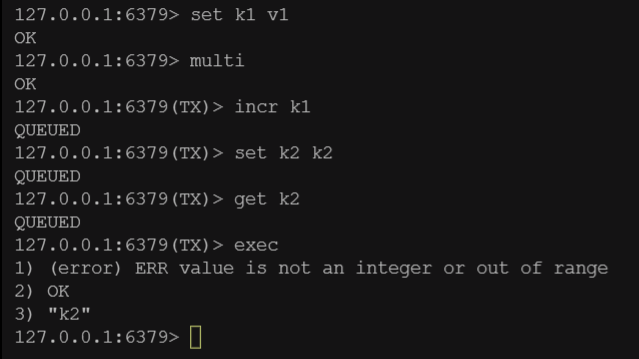
****

* **取消事务（discard）**

****

**注意：**

* **编译性异常（代码有问题! 命令有错!）,事务中所有的命令都不会被执行!**
* **运行时异常（），如果事务队列中存在语法性，那么执行命令的时候，其它命令是可以执行的，错误命令抛出异常**

****

**七、锁**

* **悲观锁**

很悲观，认为什么时候都会出问题，无论做什么都会加锁!

* **乐观锁（watch）**
* **很乐观，认为什么时候都不会出问题，所以不会上锁!更新数据的时候去判断一下，在此期间是否有人修改过这个数据**
* **获取version**
* **更新的时候比较version**

**案例：**

某商店库存有**100**件衣服，用户1买了**10**件衣服，库存还剩**90**件；正常执行成功！

**127.0.0.1:6379> set stock 100 库存有100件衣服**

**OK**

**127.0.0.1:6379> set salesVolume 0 卖出0件**

**OK**

**127.0.0.1:6379> watch stock 使用watch监听库存stock**

**OK**

**127.0.0.1:6379> multi 开启事务**

**OK**

**127.0.0.1:6379(TX)> decrby stock 10 卖出10件衣服还剩90件**

**QUEUED**

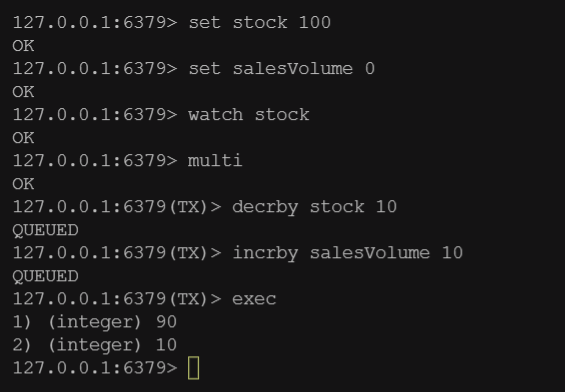
**127.0.0.1:6379(TX)> incrby salesVolume 10 总计卖出10件**

**QUEUED**

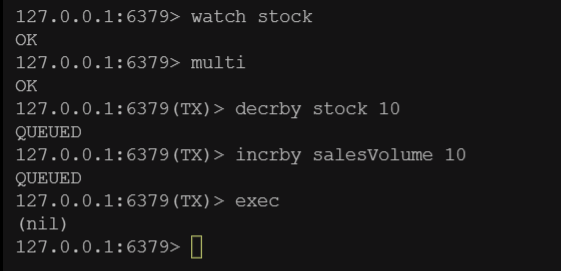
**127.0.0.1:6379(TX)> exec 执行事务，正常执行成功，期间没有监听到数据变动**

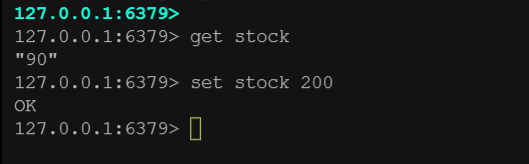
**1) (integer) 90**

**2) (integer) 10**

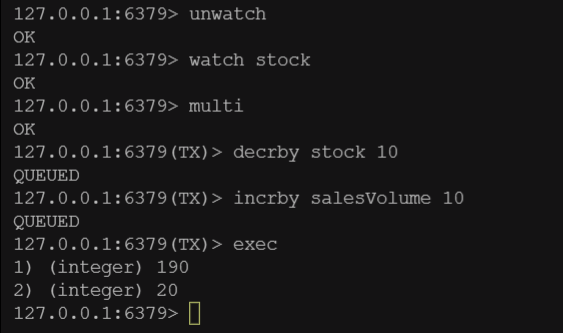


**当用户1购买衣服后事务还未执行时，如果存在用户2也进行购买，那么当用户1执行事务时会失败（数据发生变动），**





**解决以上问题，可以使用unwatch进行解锁，然后再使用watch获取新数据进行监听；**



八**、发布订阅**

Redis发布订阅（**pub/sub**）是一种通信模式；发送者（**pub**）发送消息，订阅者（**sub**）接受消息；

Redis客户端可以订阅任意数量的频道；

通过**subscribe**命令订阅某频道后，redis-server里维护一个字典，字典的键就是一个个channel（频道），二字典的值就是一个链表，链表中保存了所有订阅这个**channel**的客户端。Subscribe命令的关键，就是将客户端添加到给定channel的订阅链表中。

通过**publish**命令向订阅者发送消息，redis-server会使用给定的频道作为键，在它所维护的channel字典中查找记录订阅这个频道的所有客户端的链表，遍历这个链表，将消息发布给所有订阅者；

Pub/Sub从字面上理解就是发布（**publish**）与订阅（**subscribe**），在redis中，你可以设定对某个key值进行消息发布及消息订阅，当一个key值进行了消息发布后，所有订阅它的客户端都会收到相应的消息。

* **相关命令**
* **Psubscribe pattern**

订阅一个或多个符合给定模式的频道

* **Pubsub subcommand**

查看订阅与发布系统状态

* **Publish channel message**

将信息发送到指定频道

* **Punsubscribe pattern**

退订所有给定模式的频道

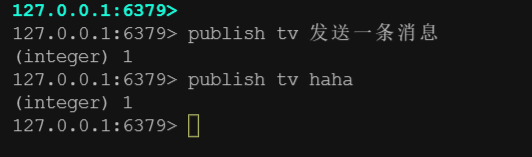
* **Subscribe channel**

订阅给定的一个或多个频道的信息

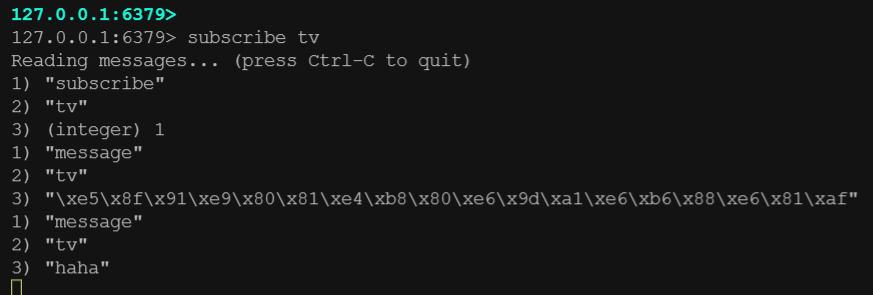
* **Unsubscribe channel**

指退订给定的频道

* **案例**
* **发布者（发送消息）**

****

* **订阅的频道（指定发送和接受的频道）**
* **订阅者（接受消息）**

****

**九、redis搭建集群**

* **主从复制**

主从复制是指将一台reids服务器的数据，复制到其它reids服务器。前者称为主节点（master/leader），后者称为从节点（slave/follower）;数据的复制是单向的，只能由主节点到从节点。Master以写为主，slave以读为主。

默认情况下，每台reids服务器都主节点；且一个主节点可以由多个从节点（或没有从节点），但一个从节点只能有一个主节点；

**主从复制的作用主要包括：**

* **数据冗余：**

主从复制实现来数据的热备份，是持久化之外的一中冗余方式；

* **故障恢复：**

当主节点出现故障时，可以由从节点提供服务，实现快速的故障恢复；实际上是一种服务的冗余；

* **负载均衡：**

在主从复制的基础上配合读写分离，可以由主节点提供服务，由从节点提供读的服务（既redis数据时应用连接主节点，读redis数据时应用连接从节点），分担服务器负载；尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提供reids服务器的高并发量。

* **高可用基石：**

除了上述作用外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是redis高可用基础。

一般来说，要将redis运用于工程项目中，只使用一台服务器是万万不能的，原因如下:

1. **从结构上，单个redis服务器会发生单点故障，并且一台服务器需要处理所有请求负载，压力较大；**
2. **从容量上，单redis服务器内存容量有限，就算一台服务器内存容量为256G，也不能将所有内存用作reids存储内存，一般来说，单台redis最大使用内存不应该超过20G.**

* **环境配置**

只需配置从库，不用配置主库!

**127.0.0.1:6379> info replication #查看当前库的信息**

**# Replication**

**role:master #角色**

**connected\_slaves:0 #没有从机**

**master\_failover\_state:no-failover**

**master\_replid:3d0d9d98730f660496fc631d2602896f162df1ee**

**master\_replid2:0000000000000000000000000000000000000000**

**master\_repl\_offset:0**

**second\_repl\_offset:-1**

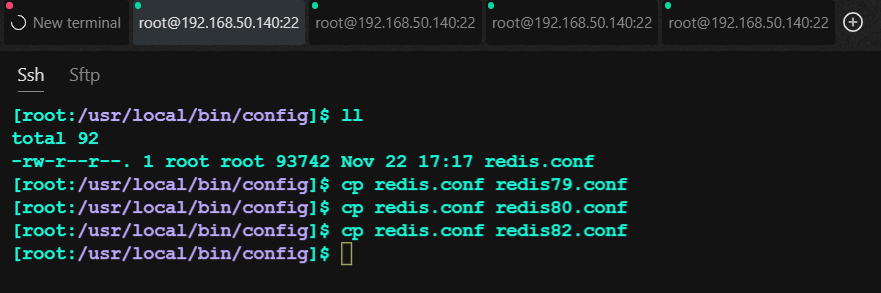
**repl\_backlog\_active:0**

**repl\_backlog\_size:1048576**

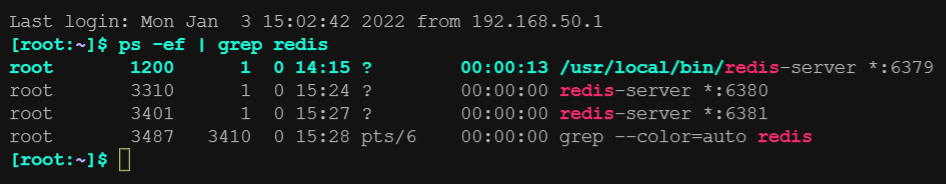
**repl\_backlog\_first\_byte\_offset:0**

**repl\_backlog\_histlen:0**

* **创建配置文件模拟集群**

****

* **修改配置文件（port、 filepid文件名、 log文件名、 rdb文件名）**
* **启动相对应的服务**
* **查看进程是否配置成功**

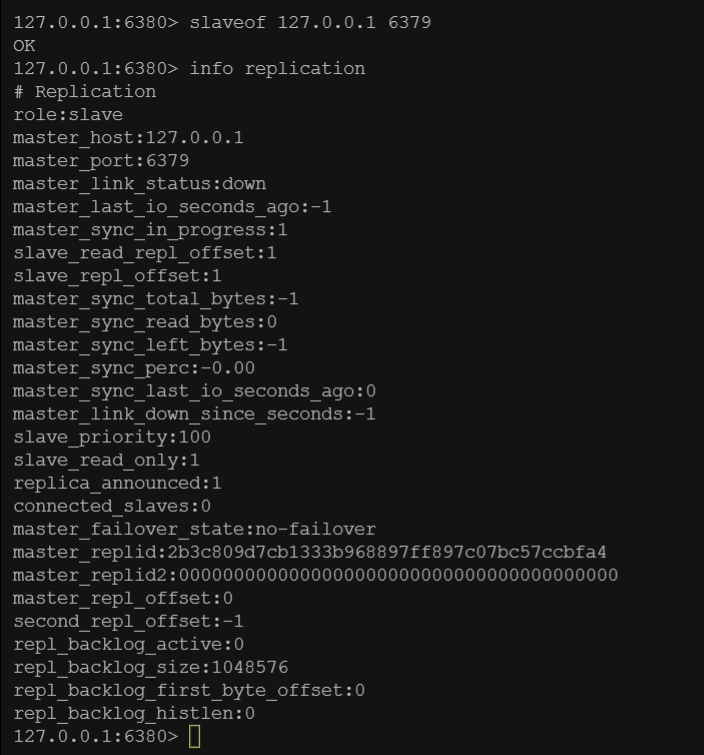
****

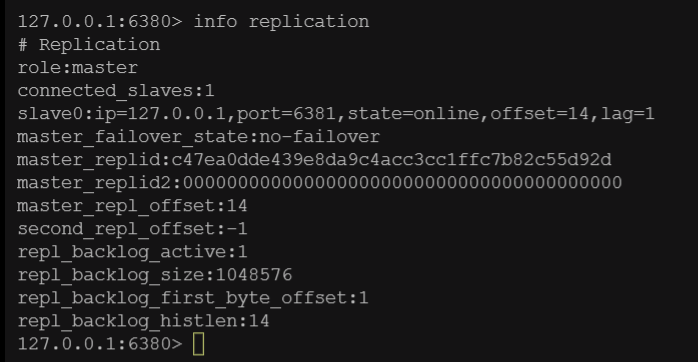
* **配置主从复制**

默认情况下每台服务器都是主节点，一般情况下只需要配置从机即可；

一主（6380）二从（6381、6382）

* **配置80和81从机（slaveof）**



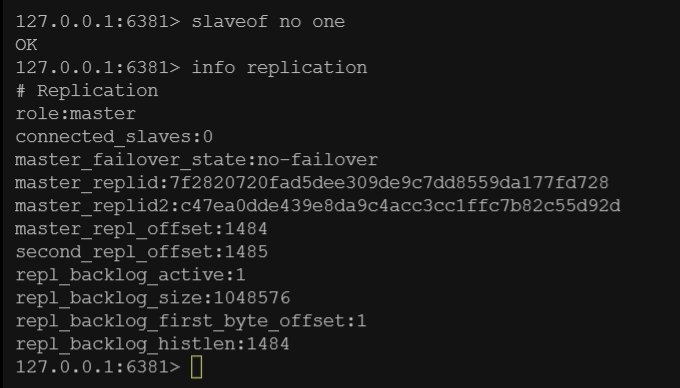


注意：如果主机设置的有密码需要在配置文件中设置**masterauth**

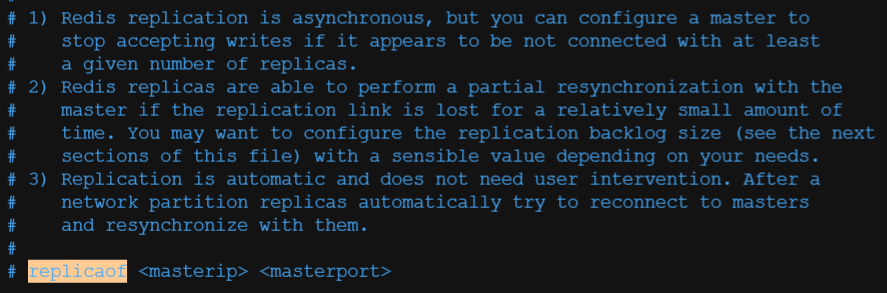
**说明：**如果主机断开连接，从机依旧连接到主机的，但是没有读的操作，这个时候主机如果回来了。从机依旧可以直接获取到主机写的信息。

如果使用的是命令式配置主从，那么当重启了，就会变回主机，只要变为从机，立马就会从主机中获取。

**主机宕机，手动设置使自己变为主机（slaveof no one）**

****

注意：真实环境中，配置主从复制是在配置文件中进行配置的



**十、哨兵模式**

**监控主机是否发生故障，如果故障了将自动将某个从机转换为主机**

哨兵模式是一种特殊的模式，redis提供了哨兵命令，哨兵是一个独立的进程，作为进程，它会独立运行。其原理是哨兵通过发送命令，等待redis服务器响应，从而监控运行多个redis实例。

* **作用**
* 通过发送命令，让redis服务器返回监控其运行状态，包括主服务器和从服务器。
* 当哨兵测到主机宕机，会自动将slave切换为master机，然后通过**发布订阅模式**通知其它服务器，修改配置文件，让它们切换主机

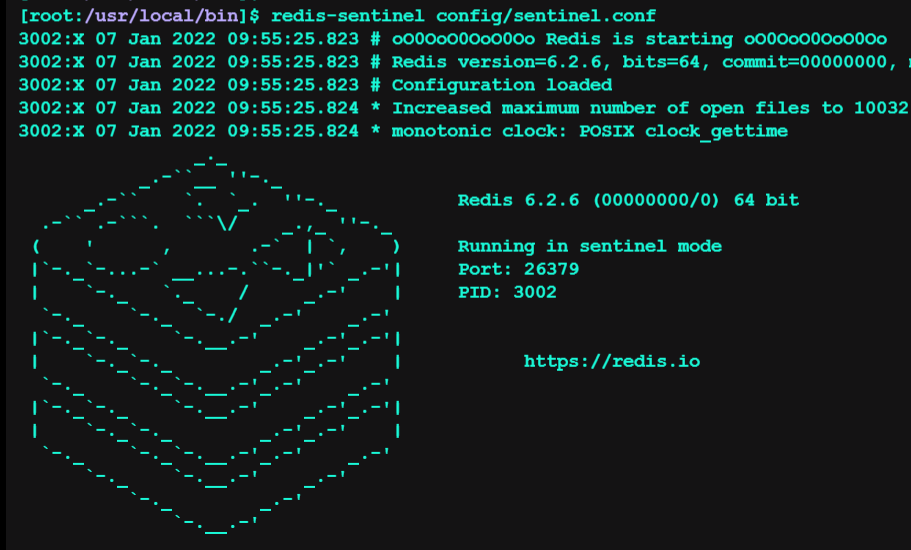
**原理介绍：**

假如主服务器宕机，哨兵1先检测到这个结果，系统并不会立刻进行failover过程，仅仅是哨兵1主观的认为主服务器不可用，这个现象成为**主观下线，**当后面的哨兵也检测到主服务器不可用，并且数量达到一定值时，那么哨兵之间就会进行一次投票选举，选举哪个从机成为主机，投票的结果由一个哨兵发起，进行**failover**（**故障转移**）操作。切换成功后，就会通过**发布订阅模式**，让各个哨兵把自己监控的从服务器实现切换主机，这个过程称为**客观下线**

* **配置哨兵**
* **创建sentinel.conf配置文件**

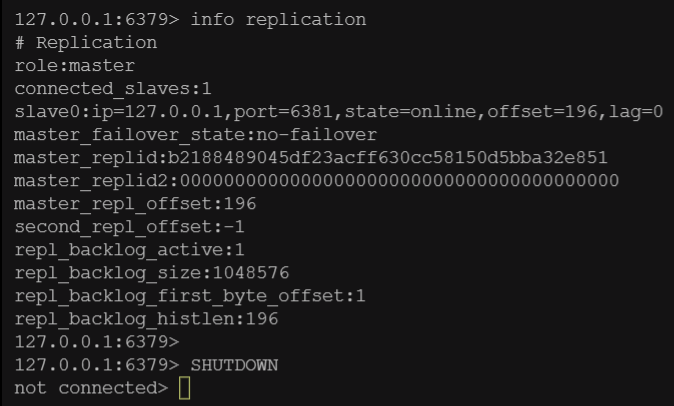
****

* **启动哨兵**

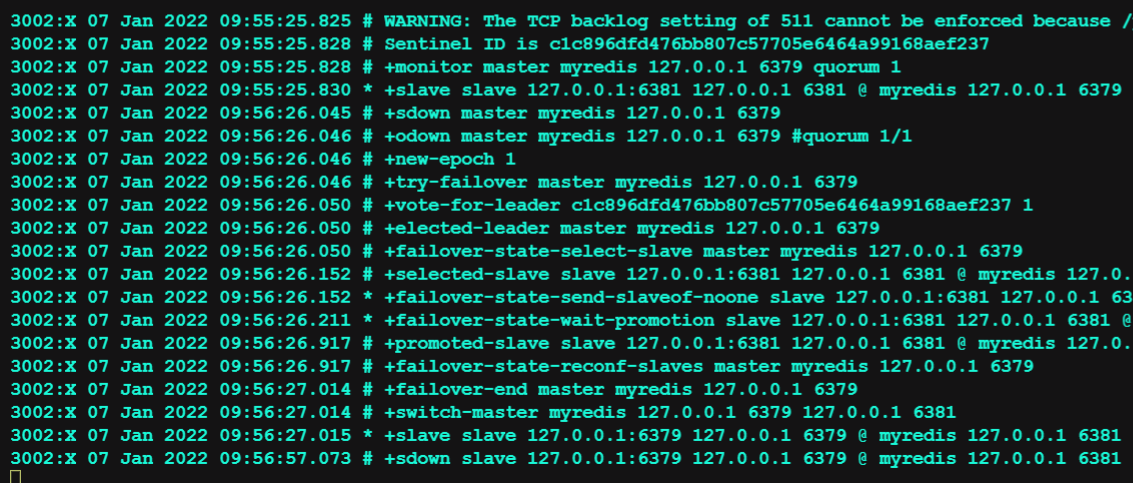


* **测试配置是否成功**

**执行命令，停止主机进行测试**



**哨兵控制台发生变化，以下为打印内容**



**以前的从机（81）当前变为了主机（哨兵模式配置成功）**

****

* **哨兵模式优缺点**
* **优点**

1. 哨兵集群，基于主从复制模式，所有的主从配置优点它全有
2. 主从可以切换，故障可以转移，系统的可用性就会更好
3. 哨兵模式就是主从模式的升级，手动到自动，更加健壮

* **缺点**

1. Redis不好在线扩容的，集群容量一旦达到上限，在线扩容就十分麻烦

**十一、缓存穿透和雪崩**

* **缓存穿透**

当缓存与数据库中都不存在该数据时，由于当数据库查询不到数据就不会写入缓存，这个时候如果用户不断的恶意发起请求，就会导致这个不存在的数据每次请求都会查询DB，请求量大的情况下，就会导致DB压力过大，直接挂掉。

**解决方案**

* **布隆过滤器**

布隆过滤器是一种数据结构，对所有可能查询的参数以hash形式存储，在控制层先进行校验，不符合则丢弃，从避免对底层数据库造成的压力。

将所有可能存在数据，分别通过多个哈希函数生成多个哈希值，然后将这些哈希值存到一个足够大的bitmap中，此时一个一定不存在的数据就会被这个bitmap拦截，从而减少了数据库的查询压力。

* **当查询返回一个空数据时，直接将这个空数据存到缓存中，过期时间不宜设置过长，建议不超过5分钟**
* **缓存击穿**

某一个数据缓存中没有但数据库中有的数据（一般是缓存时间到期），这时由于并发用户特别多，同时读缓存没读到数据，又同时去数据库去取数据，引起数据库压力瞬间增大，严重情况下会直接挂掉。

**解决方案**

添加互斥锁，只保证一个线程进去，其余的进行等待

* **缓存雪崩**

缓存中大批量的数据都到了**过期时间**，从而导致查询数据量巨大，引起数据库压力过大甚至down机。和缓存击穿不同，缓存击穿是指某一条数据到了过期时间，大量的并发请求都来查询这一条数据，缓存雪崩是不同数据都过期了，很多数据都查不到从而查数据库

**解决方案**

* 设置热点数据永不过期
* 缓存数据的过期时间设置随机，可以在原有的过期时间上加上一个随机值，比如1-3min，防止同一时间大量缓存数据集体失效，导致数据库压力过大。

* 如果是分布式部署缓存数据库，可将热点数据分别存放到不同的缓存数据库中，避免某一点由于压力过大而down掉。