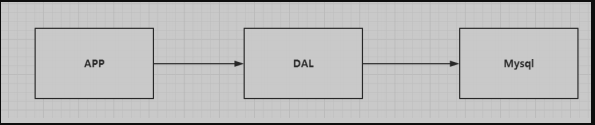
# Redis

# NoSqL概述

# 1.架构的演进

## 单机MySQL的年代



90年代 一个基本的网站访问量一般不会太大，单个数据库完全足够

那个时候，更多的去使用静态网页 Html ~ 服务器根本没有太大的压力！

思考一下，这种情况下：整个网站的瓶颈是什么？

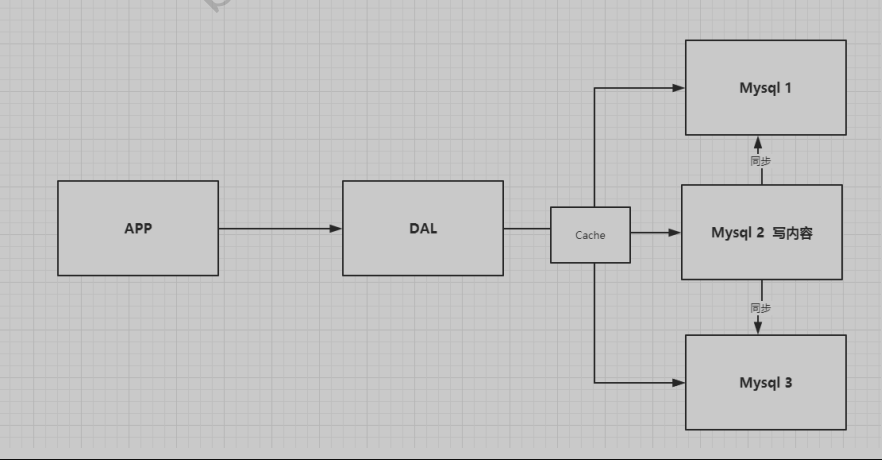
1 、数据量如果太大、一个机器放不下了！

2 、数据的索引 （B+ Tree），一个机器内存也放不下

3 、访问量（读写混合），一个服务器承受不了~

只要你开始出现以上的三种情况之一，那么你就必须要晋级！

## Memcached(缓存) + MySQL +垂直拆分(读写分离)



网站80%的情况都是在读取数据，每次都要查询数据库的话就十分的麻烦，所以我们希望减轻数据库的压力，我们可以使用缓存来保证效率。

发展过程：优化数据结构和索引--->文件缓存(IO)---->Memcached(当时热门技术)

## 分库分表+水平拆分+MySQL集群

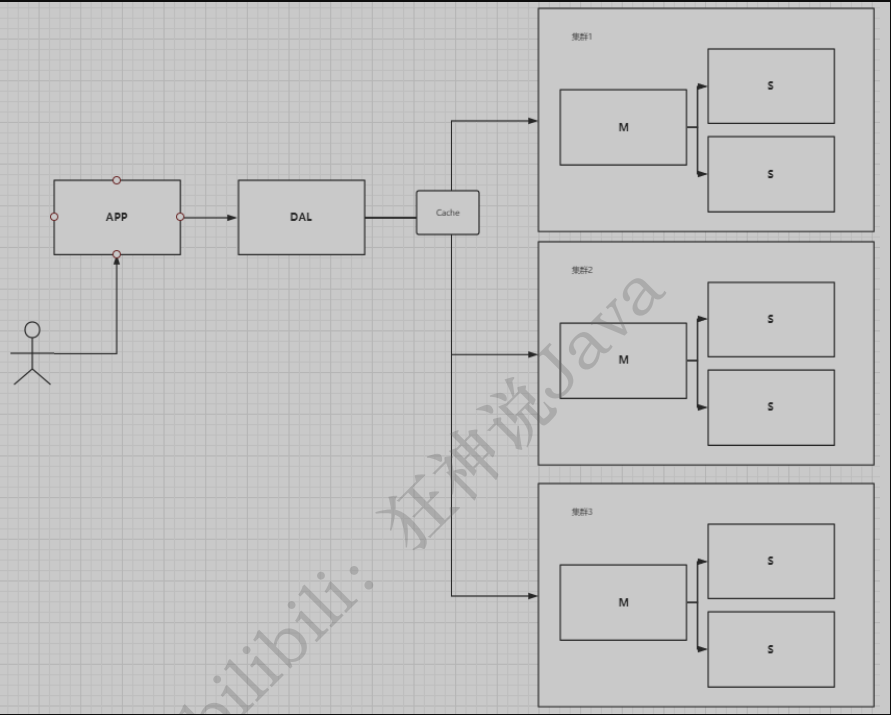
技术和业务在发展的同时，对人的要求也越来越高

本质：数据库（读+写）

早些年MyISAM 表锁 十分影响效率，高并发下就会出现严重的问题

早些年Innodb 行锁

慢慢的就开始使用分库分表解决写的压力，MySQL慢慢的退出表分区，这个并没有多少公司使用，还退出了MySQL的集群，有了集群就很好的满足了那个年代所有的需求



## 最近的年代

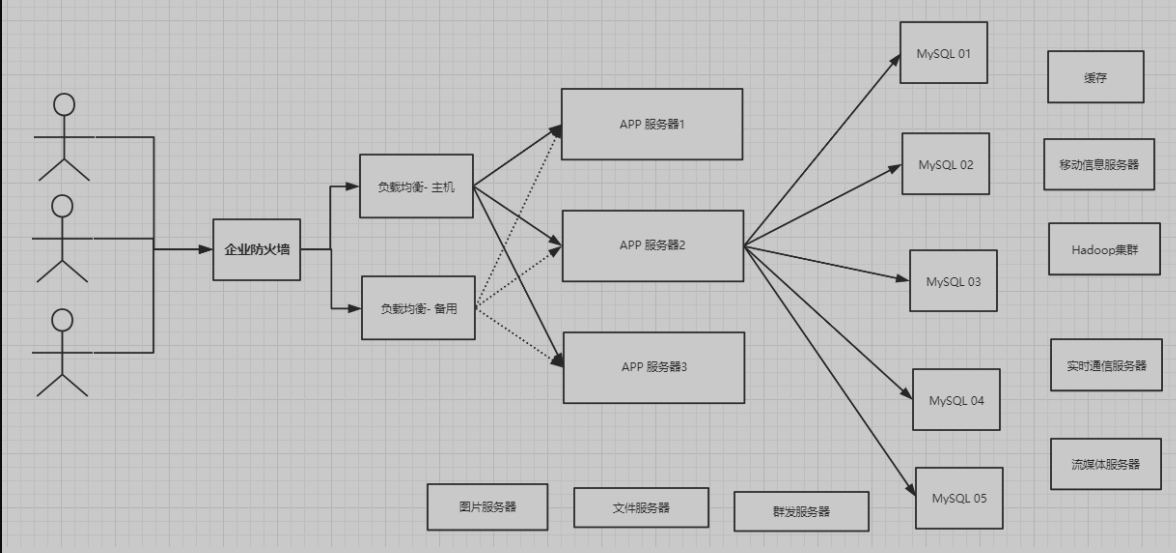
2010--2020 十年之间，世界已经发生了翻天覆地的变化；（定位，也是一种数据，音乐，热榜！）

MySQL 等关系型数据库就不够用了！数据量很多，变化很快~！

MySQL 有的使用它来村粗一些比较大的文件，博客，图片！数据库表很大，效率就低了！如果有一种数  
据库来专门处理这种数据,

MySQL压力就变得十分小（研究如何处理这些问题！）大数据的IO压力下，表几乎没法更大！

目前一个基本的互联网项目！



## 5.为什么要使用NoSQL

用户的个人信息，社交网络，地理位置。用户自己产生的数据，用户日志等等爆发式增长！

这时候我们就需要使用NoSQL数据库的，Nosql 可以很好的处理以上的情况！

# NoSQL

## 什么是NoSQL

NoSQL = Not Only SQL （不仅仅是SQL）

关系型数据库：表格 ，行 ，列

泛指非关系型数据库的，随着web2.0互联网的诞生！传统的关系型数据库很难对付web2.0时代！尤其  
是超大规模的高并发的社区！ 暴露出来很多难以克服的问题，NoSQL在当今大数据环境下发展的十分迅  
速，Redis是发展最快的，而且是我们当下必须要掌握的一个技术！

很多的数据类型用户的个人信息，社交网络，地理位置。这些数据类型的存储不需要一个固定的格式！  
不需要多月的操作就可以横向扩展的 ！ Map<String,Object> 使用键值对来控制！

## 2.NoSQL的特点

解耦！

1 、方便扩展（数据之间没有关系，很好扩展！）

2 、大数据量高性能（Redis 一秒写 8 万次，读取 11 万，NoSQL的缓存记录级，是一种细粒度的缓存，性能会比较高！）

3 、数据类型是多样型的！（不需要事先设计数据库！随取随用！如果是数据量十分大的表，很多人就无法设计了！）

## 3.NoSQL的四大分类

### 1.KV键值对：

新浪 Redis

美团 Redis + Tair

阿里，百度 Redis + memecache

### 文档型数据库（bson格式 和json一样）

MongoDB (一般必须掌握)

MongoDB 是一个基于分布式文件存储的数据库 C++编写 主要用来处理大量的文档

MongoDB 是一个介于关系型数据库和非关系型数据中中间的产品，MongoDB 是非关系型数据库中功能最丰富的，最想关系型数据库的

ConthDB

### 列存储数据库

Hbase 分布式文件系统

### 图关系型数据库

他不是存储图形，放的是关系，比如朋友圈社交网络，广告推荐

### 四者对比



# Redis入门

## redis是什么

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

是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Val数据库，并提供多种语言的API。



redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

免费和开源！是当下最热门的 NoSQL 技术之一！也被人们称之为结构化数据库！

## Redis能干嘛？

1 、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）

2 、效率高，可以用于高速缓存

3 、发布订阅系统

4 、地图信息分析

5 、计时器、计数器（浏览量！）

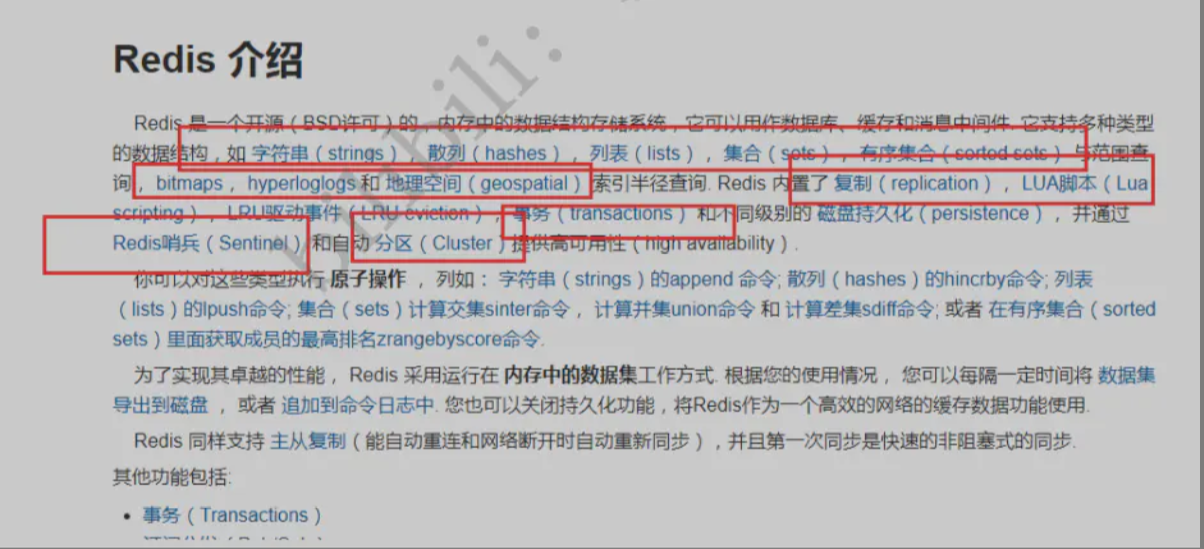
## 特性

1 、多样的数据类型

2 、持久化

3 、集群

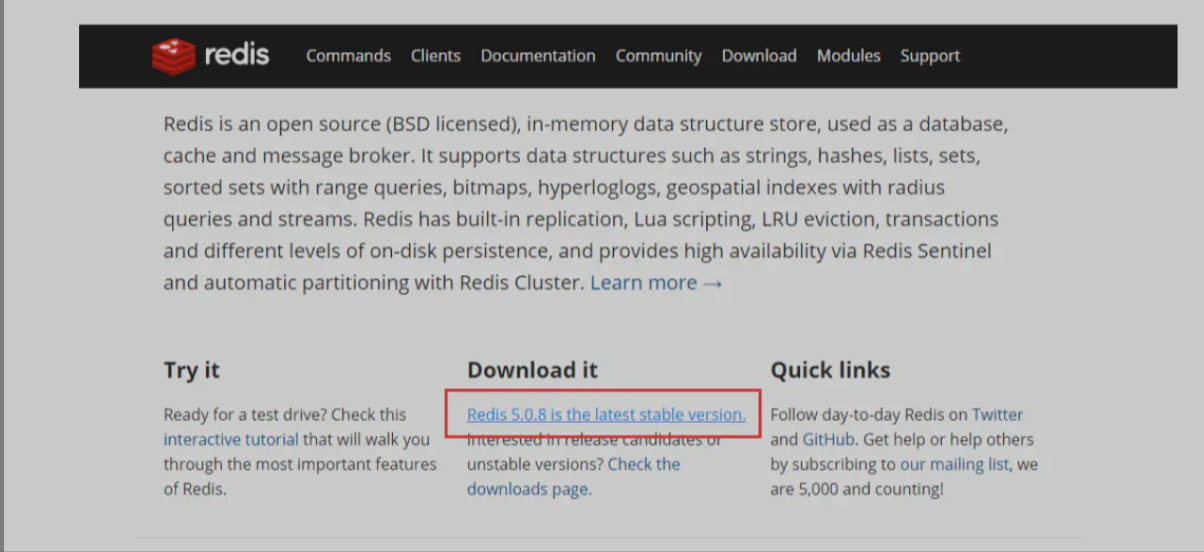
4 、事务



官网：[https://redis.io/](https://links.jianshu.com/go?to=https://redis.io/" \t "https://www.jianshu.com/p/_blank)

中文网：[http://www.redis.cn/](https://links.jianshu.com/go?to=http://www.redis.cn/" \t "https://www.jianshu.com/p/_blank)

下载地址：通过官网下载即可！



## Windows安装

1 、下载安装包：[https://github.com/dmajkic/redis/releases](https://links.jianshu.com/go?to=https://github.com/dmajkic/redis/releases" \t "_blank)

2 、下载完毕得到压缩包：

3 、解压到自己电脑上的环境目录下的就可以的！Redis 十分的小，只有5M

4 、开启Redis，双击运行服务即可！

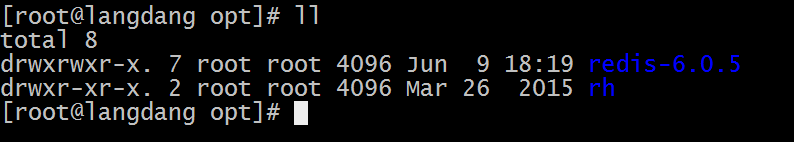
5 、使用redis客户单来来连接redis

记住一句话，Window下使用确实简单，但是Redis 推荐我们使用Linux去开发使用！

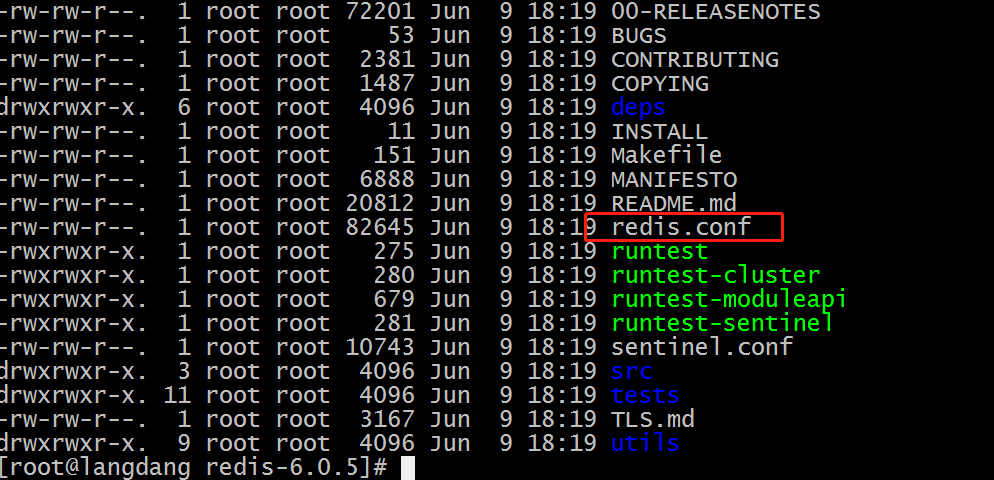
## 5.Linux安装

1 、下载安装包！ redis-5.0.8.tar.gz

2 、解压Redis的安装包！ 程序/opt



3 、进入解压后的文件，可以看到我们redis的配置文件



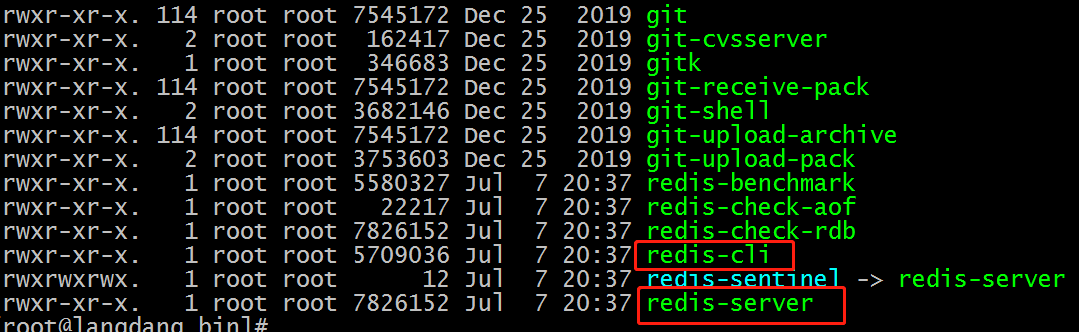
4 、基本的环境安装

yum install gcc-c++

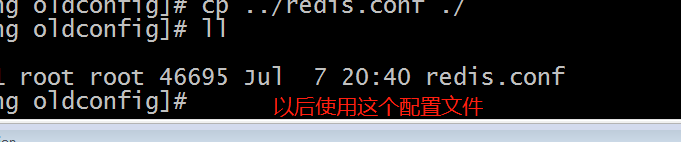
make

make install

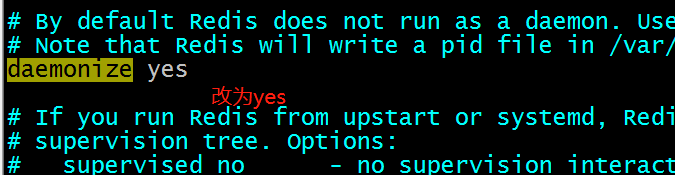
5 、redis的默认安装路径 /usr/local/bin



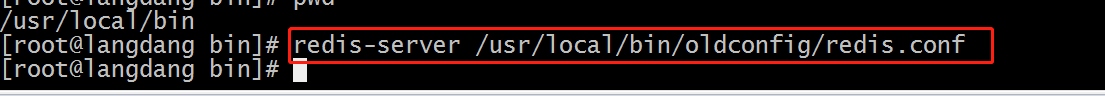
6 、将redis配置文件。复制到我们当前目录下



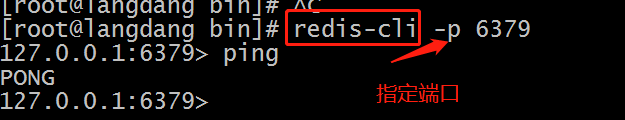
7 、redis默认不是后台启动的，修改配置文件！



8 、启动Redis服务！



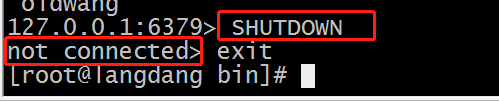
9 、使用redis-cli 进行连接测试！



10 、查看redis的进程是否开启！



11 、如何关闭Redis服务呢？ shutdown



# 4.Redis的使用

## 1.性能测试

redis-benchmark 是一个压力测试工具！官方自带的性能测试工具！

redis-benchmark 命令参数！

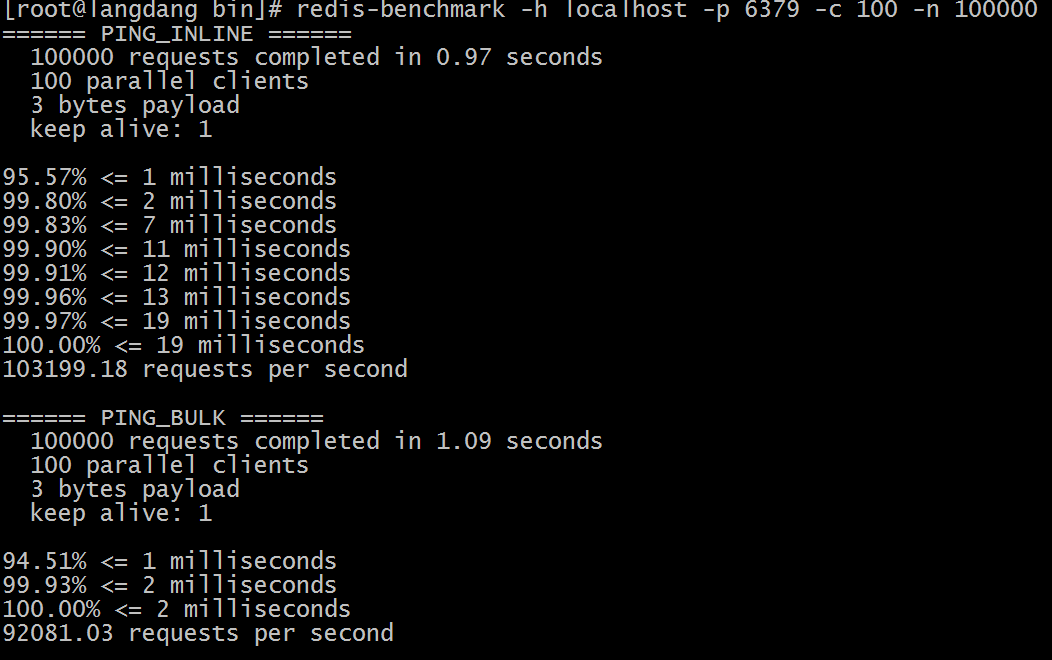


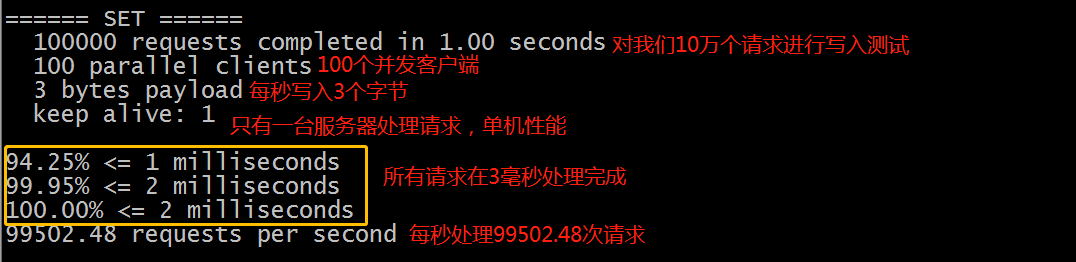
我们来简单测试下：

如何查看这些分析呢？

测试： 100 个并发连接 100000 请求

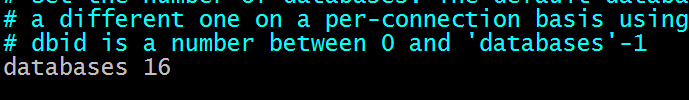
**redis-benchmark -h localhost -p 6379 -c 100 -n 100000**

**`**



## 2.基础知识

redis默认有 16 个数据库



默认使用的是第 0 个

可以使用 select 进行切换数据库！

|  |
| --- |
| 127.0.0.1:6379> select 3 切换数据库  127.0.0.1:6379[3]> DBSIZE 查看数据库大小  127.0.0.1:6379[3]> keys \* 查看数据库所有的key  127.0.0.1:6379[3]> FLUSHDB 清除当前数据库 flushdb  127.0.0.1:6379[3]> FLUSHALL 清除全部数据库的内容 |

为什么redis端口号是6379 粉丝效应

## 3.Redis为什么是单线程

明白Redis是很快的，官方表示，Redis是基于内存操作，CPU不是Redis性能瓶颈，Redi的瓶颈是根据机器的内存和网络带宽，既然可以使用单线程来实现，就使用单线程了！所有就使用了单线程了！

Redis 是C 语言写的，官方提供的数据为 100000+ 的QPS，完全不比同样是使用 key-vale的Memecache差！

## 4.为什么单线程这么快？

1 、误区 1 ：高性能的服务器一定是多线程的？

2 、误区 2 ：多线程（CPU上下文会切换！）一定比单线程效率高！

CPU>内存>硬盘的速度要有所了解！

核心：redis 是将所有的数据全部放在内存中的，所以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！！！），对于内存系统来说，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！

# 五大基本数据类型

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件MQ。 它支持多种类型的数据结构，如 字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。 Redis 内置了 复制（replication），LUA脚本（Lua scripting）， LRU驱动事件（LRU eviction），事务（transactions） 和不同级别的 磁盘持久化（persistence）， 并通过Redis哨兵（Sentinel）和自动 分区（Cluster）提供高可用性（high availability）。

## RedisKey的基本命令

|  |
| --- |
| 127 .0.0.1:6379> keys \* 查看所有的key(empty list or set)  127 .0.0.1:6379> set name oldwang # set key  OK  127 .0.0.1:6379> keys \*  1 ) "name"  127 .0.0.1:6379> set age 1  OK  127 .0.0.1:6379> keys \*  1 ) "age"  2 ) "name"  127 .0.0.1:6379> EXISTS name 判断当前的key是否存在  (integer) 1  127 .0.0.1:6379> EXISTS name  (integer) 0  127 .0.0.1:6379> move name 1 移除当前的key  (integer) 1  127 .0.0.1:6379> keys \*  1 ) "age"  127 .0.0.1:6379> set name oldwang  OK  127 .0.0.1:6379> keys \*  1 ) "age"  2 ) "name"  127 .0.0.1:6379> get name"oldwang"  127 .0.0.1:6379> EXPIRE name 10 设置key的过期时间，单位是秒(integer) 1  127 .0.0.1:6379> ttl name 查看当前key的剩余时间  (integer) 4  127 .0.0.1:6379> ttl name  (integer) 3  127 .0.0.1:6379> ttl name  (integer) -  127 .0.0.1:6379> get name  (nil)  127 .0.0.1:6379> type name 查看当前key的一个类型！  string  127 .0.0.1:6379> type age  string |

## 2.String(字符串类型)详解

|  |
| --- |
| 127.0.0.1:6379> set key1 v1 设置值  OK  127.0.0.1:6379> get key1 获取值  "v1"  127.0.0.1:6379> keys \* 获取所有key  1) "key1"  127.0.0.1:6379> EXISTS key1 判断某一个key是否存在  (integer) 1  127.0.0.1:6379> APPEND key1 "hello" 追加字符串如果 key不存在就相当于set key  (integer) 7  127.0.0.1:6379> get key1  "v1hello"  127.0.0.1:6379> STRLEN key1 获取字符串的长度  (integer) 7  127.0.0.1:6379> APPEND key1 oldwang  (integer) 14  127.0.0.1:6379> STRLEN key1  (integer) 14  ----------------------------------------------------------------------------------------------------------------  127.0.0.1:6379> set views 0 初始值为0  OK  127.0.0.1:6379> get views  "0"  127.0.0.1:6379> INCR views 自增1  (integer) 1  127.0.0.1:6379> INCR views  (integer) 2  127.0.0.1:6379> get views  "2"  127.0.0.1:6379> DECR views 自减1  (integer) 1  127.0.0.1:6379> DECR views  (integer) 0  127.0.0.1:6379> get views  "0"  127.0.0.1:6379> INCRBY views 10 可以设置步长指定增量  (integer) 10  127.0.0.1:6379> INCRBY views 10  (integer) 20  127.0.0.1:6379> DECRBY views 5 减去步长  (integer) 15  ----------------------------------------------------------------------------------------------------------------  字符串范围 range  127.0.0.1:6379> set key1 "hello oldwang" 设置key1的值  OK  127.0.0.1:6379> GETRANGE key1 0 3 截取字符串，  "hell"  127.0.0.1:6379> GETRANGE key1 0 -1 获取全部字符串，和get key是一样的  "hello oldwang"  127.0.0.1:6379>  ----------------------------------------------------------------------------------------------------------------  替换字符串  127.0.0.1:6379> SETRANGE key2 1 xxx 替换指定位置开始的字符串  (integer) 7  127.0.0.1:6379> get key2  "axxxefg"  -------------------------------------------------------------------------------------------------------------  setex (set with expire) 设置过期时间  setnx (set if not exists) 不存在设置。 分布式锁中经常使用  127.0.0.1:6379> set key3 60 设置过期时间  OK  127.0.0.1:6379> SETEX key3 30 "hello" 设置key3的值为hello 30S过期  OK  127.0.0.1:6379> ttl key3  (integer) 23  127.0.0.1:6379> SETNX mykey "redis" 如果mykey不存在就创建  (integer) 1  127.0.0.1:6379> keys \*  1) "key1"  2) "mykey"  3) "key2"  127.0.0.1:6379> SETNX mykey "hello" 如果存在就创建失败  (integer) 0  127.0.0.1:6379> keys \*  1) "key1"  2) "mykey"  3) "key2"  127.0.0.1:6379>  -------------------------------------------------------------------------------------------------------------  批量设置 mset  批量过期 mget  127.0.0.1:6379> mset key1 v1 key2 v2 key3 v3 同时设置多个值  OK  127.0.0.1:6379> keys \*  1) "key1"  2) "key3"  3) "key2"  127.0.0.1:6379> mget key1 key2 key3 同时获取多个值  1) "v1"  2) "v2"  3) "v3"  127.0.0.1:6379> MSETNX key1 v1 k4 v4 msetnx是一个原子性操作  (integer) 0  127.0.0.1:6379> get k4  (nil)  127.0.0.1:6379>  设置对象  set user:1{name:zhangsan,age；3} 设置一个user:1对象 值为json字符串保存对象  这里的key是一个巧妙的设计 user:{id}:{filed},如此设计在redis中是ok的  127.0.0.1:6379> mset user:1:name zhangsan user:1:age 2  OK  127.0.0.1:6379> mget user:1:name user:1:age  1) "zhangsan"  2) "2"  -------------------------------------------------------------------------------------------------------------  getset 先获取在设置  127.0.0.1:6379> getset db redis 如果不存在返回nil  (nil)  127.0.0.1:6379> getset db redis  "redis"  127.0.0.1:6379> getset db mysql 如果存在值 获取原来的值，并设置新的值  "redis"  127.0.0.1:6379> getset db mysql  "mysql"  127.0.0.1:6379> |

String类似的使用场景：value除了是我们的字符串还可以是我们的数字！

计数器

统计多单位数量

粉丝数

对象缓存存储

## 3.List列表类型详解

基本的数据类型，列表

在redis中我们可以把List玩成 栈，队列，阻塞队列

所有的List命令都是L开头的

|  |
| --- |
| 127.0.0.1:6379> LPUSH list one 将一个值或多个值插入到列表的头部（左边）  (integer) 1  127.0.0.1:6379> LPUSH list two  (integer) 2  127.0.0.1:6379> LPUSH list three  (integer) 3  127.0.0.1:6379> LRANGE list 0 -1 获取所有值  1) "three"  2) "two"  3) "one"  127.0.0.1:6379> LRANGE list 0 1 通过区间获取具体的值  1) "three"  2) "two"  127.0.0.1:6379> RPUSH list right 将一个值或多个值插入到列表的尾部（右边）  (integer) 4  127.0.0.1:6379> LRANGE list 0 -1  1) "three"  2) "two"  3) "one"  4) "right"  LPOP  RPOP  127.0.0.1:6379> LRANGE list 0 -1  1) "three"  2) "two"  3) "one"  4) "right"  127.0.0.1:6379> LPOP list 移除List的第一个元素  "three"  127.0.0.1:6379> RPOP list 移除List的最后一个元素  "right"  127.0.0.1:6379>  127.0.0.1:6379> LINDEX list 1 通过下标获取List中某一个值  "one"  127.0.0.1:6379> LINDEX list 0  "two"  127.0.0.1:6379>  Llen  127.0.0.1:6379> LPUSH list one two three  (integer) 3  127.0.0.1:6379> LLEN list 获取List的长度  (integer) 3  127.0.0.1:6379>  移除指定的值  Lrem  127.0.0.1:6379> LREM list 1 one 移除List集合中指定个数的value 精确匹配  (integer) 1  127.0.0.1:6379> LRANGE list 0 -1  1) "three"  2) "three"  3) "two"  127.0.0.1:6379> LREM list 2 three  (integer) 2  127.0.0.1:6379>  127.0.0.1:6379> RPUSH list "hello"  (integer) 1  127.0.0.1:6379> RPUSH list "hello1"  (integer) 2  127.0.0.1:6379> RPUSH list "hello2"  (integer) 3  127.0.0.1:6379> RPUSH list "hello3"  (integer) 4  127.0.0.1:6379> LTRIM list 1 2 通过下标截取指定的长度，这个List已经改变了，只剩下截取的元素  OK  127.0.0.1:6379> LRANGE list 0 -1  1) "hello1"  2) "hello2"  127.0.0.1:6379>  RPOPLPUSH 移除列表的最后一个元素，移动到新的列表中  127.0.0.1:6379> RPUSH list "hello1"  (integer) 1  127.0.0.1:6379> RPUSH list "hello2"  (integer) 2  127.0.0.1:6379> RPUSH list "hello3"  (integer) 3  127.0.0.1:6379> RPOPLPUSH list mylist 移除列表的最后一个元素，移动到新的列表中  "hello3"  127.0.0.1:6379> LRANGE list 0 -1  1) "hello1"  2) "hello2"  127.0.0.1:6379> LRANGE mylist 0 -1   1. "hello3"   Lset 将列表中指定下标的值替换为另一个值   1. 127.0.0.1:6379> EXISTS list 判断是否存在 2. (integer) 0 3. 127.0.0.1:6379> LSET list 0 one 如果不存在更新就会保存 4. (error) ERR no such key 5. 127.0.0.1:6379> LPUSH list one 6. (integer) 1   127.0.0.1:6379> LSET list 0 two 如果存在更新当前下标的值   1. OK 2. 127.0.0.1:6379> LRANGE list 0 0 3. 1) "two" 4. 127.0.0.1:6379>   LInsert 将某个具体的value插入到列表中某个元素的前面或后面  127.0.0.1:6379> FLUSHALL  OK  127.0.0.1:6379> RPUSH list hello  (integer) 1  127.0.0.1:6379> RPUSH list world  (integer) 2  127.0.0.1:6379> LINSERT list before world other 插入到指定元素的前边  (integer) 3  127.0.0.1:6379> LRANGE list 0 -1  1) "hello"  2) "other"  3) "world"  127.0.0.1:6379> LINSERT list after hello new 插入到指定元素的后边  (integer) 4  127.0.0.1:6379> LRANGE list 0 -1  1) "hello"  2) "new"  3) "other"  4) "world"  127.0.0.1:6379> |

List实际上是一个列表 是Node before，after left right都可以插入

如果key不存在 创建新的列表

如果key存在 新增内容

如果移除了所以的值就是空链表 也代表不存在

在两边插入或者改动效率最高，中间元素相对来说效率会低一点

场景，消息队列(lpush rpop) 栈(lpush,lpop)

## Set集合类型详解

Set中的值不能重复

|  |
| --- |
| 127.0.0.1:6379> SADD list one set中添加元素  (integer) 1  127.0.0.1:6379> SADD list two  (integer) 1  127.0.0.1:6379> SMEMBERS list 查看set中所有值  1) "two"  2) "one"  127.0.0.1:6379> SISMEMBER list list 判断某一个值是否存在  (integer) 0  127.0.0.1:6379> SISMEMBER list one  (integer) 1  127.0.0.1:6379>  127.0.0.1:6379> SCARD list 获取set集合中的个数  (integer) 2  127.0.0.1:6379>  127.0.0.1:6379> SCARD list  (integer) 2  127.0.0.1:6379> SREM list one 移除集合中的元素  (integer) 1  127.0.0.1:6379> SCARD list  (integer) 1  127.0.0.1:6379> SMEMBERS list  1) "two"  127.0.0.1:6379>  Set 抽随机  127.0.0.1:6379> SMEMBERS list 随机抽选出一个元素  1) "two"  127.0.0.1:6379> SRANDMEMBER list  "two"  127.0.0.1:6379> SRANDMEMBER list 1 随机抽选一个元素  "two"  127.0.0.1:6379> SRANDMEMBER list  "two"  127.0.0.1:6379>  删除随机的key  127.0.0.1:6379> SMEMBERS list  1) "two"  2) "xx"  3) "oo"  127.0.0.1:6379> SPOP list 随机删除集合中的元素  "two"  127.0.0.1:6379> SMEMBERS list  1) "xx"  2) "oo"  127.0.0.1:6379> SPOP list  "oo"  127.0.0.1:6379>  将一个指定的值移动到另外一个集合  127.0.0.1:6379> SADD list1 set2  (integer) 1  127.0.0.1:6379> keys \*  1) "list1"  2) "list"  127.0.0.1:6379> SMOVE list list2 a 将一个指定的值移动到另外一个集合  (integer) 1  127.0.0.1:6379> SMEMBERS list2  1) "a"  127.0.0.1:6379>  共同关注（并集）  数字集合类  差集 SDIFF  交集 SINTER  并集 SUNION  127.0.0.1:6379> SADD k1 a  (integer) 1  127.0.0.1:6379> SADD k1 b  (integer) 1  127.0.0.1:6379> SADD k1 c  (integer) 1  127.0.0.1:6379> SADD k2 c  (integer) 1  127.0.0.1:6379> SADD k2 d  (integer) 1  127.0.0.1:6379> SADD k2 e  (integer) 1  127.0.0.1:6379> SDIFF k1 k2 差集  1) "b"  2) "a"  127.0.0.1:6379> SINTER k1 k2 交集  1) "c"  127.0.0.1:6379> SUNION k1 k2 并集  1) "b"  2) "c"  3) "a"  4) "e"  5) "d"  127.0.0.1:6379> |

微博，A用户将所有关注的人放在set集合中，将粉丝放在一个集合中，

场景 共同关注，共同爱好，二度好友，推荐好友

## Hash哈希类型详解

Map集合 key-<key value> 这时候这个值就是一个map集合 hash本质和string类型没有太大区别，还是一个简单的集合

|  |
| --- |
| 127.0.0.1:6379> HSET hash field1 old set一个具体的 key-value  (integer) 1  127.0.0.1:6379> HGET hash field1  "old"  127.0.0.1:6379> HMSET hash field1 wang field2 oldwang set多个key-value  OK  127.0.0.1:6379> HMGET hash field1 field2 获取多个字段的值  1) "wang"  2) "oldwang"  127.0.0.1:6379> HGETALL hash 获取hash中全部的数据  1) "field1"  2) "wang"  3) "field2"  4) "oldwang"  127.0.0.1:6379>  127.0.0.1:6379> HDEL hash field1 删除指定的key字段，对应的value值也就消失了  (integer) 1  127.0.0.1:6379> HGETALL hash  1) "field2"  2) "oldwang"  127.0.0.1:6379>  Hlen  127.0.0.1:6379> HGETALL hash  1) "field2"  2) "oldwang"  3) "f1"  4) "hello"  5) "f2"  6) "world"  127.0.0.1:6379> HLEN hash 获取hash表的字段数量  (integer) 3  127.0.0.1:6379>  127.0.0.1:6379> HEXISTS hash f1 判断hash指定字段是否存在  (integer) 1  127.0.0.1:6379> HEXISTS hash f3  (integer) 0  127.0.0.1:6379>  只获取所有的field  只获取所有的值value  127.0.0.1:6379> HKEYS hash 只获取所有的field  1) "field2"  2) "f1"  3) "f2"  127.0.0.1:6379> HVALS hash 只获取所有的值value  1) "oldwang"  2) "hello"  3) "world"  127.0.0.1:6379>  127.0.0.1:6379> hset hash field3 5  (integer) 1  127.0.0.1:6379> HINCRBY hash field3 1 指定增量  (integer) 6  127.0.0.1:6379> HINCRBY hash field3 -1  (integer) 5  127.0.0.1:6379> HSETNX hash field4 hello 如果不存在则可以设置  (integer) 1  127.0.0.1:6379> HSETNX hash field4 hellow 如果不存在不能设置  (integer) 0  127.0.0.1:6379> |

Hash存储变更数据 尤其是用户信息保存 经常变动的信息

## Zset有序集合详解

在set集合的基础上增加了一个值

|  |
| --- |
| 127.0.0.1:6379> ZADD set 1 one 添加一个值  (integer) 1  127.0.0.1:6379> ZADD set 2 two 3 three 添加多个值  (integer) 2  127.0.0.1:6379> ZRANGE set 0 -1  1) "one"  2) "two"  3) "three"  127.0.0.1:6379>  127.0.0.1:6379> ZADD salary 2500 xiaohong 添加用户  (integer) 1  127.0.0.1:6379> ZADD salary 5000 zhangsan  (integer) 1  127.0.0.1:6379> ZADD salary 50000 oldwang  (integer) 1  127.0.0.1:6379> ZRANGEBYSCORE salary -inf +inf 显示全部用户 从小到大  1) "xiaohong"  2) "zhangsan"  3) "oldwang"  127.0.0.1:6379> ZRANGEBYSCORE salary -inf +inf withscores 显示全部用户并且附带成绩  1) "xiaohong"  2) "2500"  3) "zhangsan"  4) "5000"  5) "oldwang"  6) "50000"  127.0.0.1:6379> ZRANGEBYSCORE salary -inf 2500 withscores 显示工资小于2500的员工升序  1) "xiaohong"  2) "2500"  127.0.0.1:6379>  127.0.0.1:6379> ZREVRANGE salary 0 -1 从大到小进行排序  1) "oldwang"  2) "zhangsan"  移除元素  127.0.0.1:6379> ZRANGE salary 0 -1  1) "xiaohong"  2) "zhangsan"  3) "oldwang"  127.0.0.1:6379> ZREM salary xiaohong 移除有序集合中的指定元素  (integer) 1  127.0.0.1:6379> ZRANGE salary 0 -1  1) "zhangsan"  2) "oldwang"  127.0.0.1:6379>  127.0.0.1:6379> ZCARD salary 获取有序集合中的个数  (integer) 2  127.0.0.1:6379>  区间计算  127.0.0.1:6379> ZADD set 1 a  (integer) 1  127.0.0.1:6379> ZADD set 2 b  (integer) 1  127.0.0.1:6379> ZADD set 3 c  (integer) 1  127.0.0.1:6379> ZCOUNT set 1 3 获取集合的区间个数  (integer) 3  127.0.0.1:6379> ZCOUNT set 1 2  (integer) 2  127.0.0.1:6379> |

案例 存储班级成绩表，工资表 带权重进排序，排行榜应用实现 TopN

# 6.三种特殊数据类型

## 1.Geospatial 地理位置

朋友的定位，附近的人，打车距离

Redis的Geo 在redis3.2版本就推出了，这个功能可以推算地理位置信息，

可以查一些数据 [http://www.jsons.cn/lngcodeinfo/0706D99C19A781A3/](https://links.jianshu.com/go?to=http://www.jsons.cn/lngcodeinfo/0706D99C19A781A3/" \t "https://www.jianshu.com/p/_blank)



官方文档：[https://www.redis.net.cn/order/3685.html](https://links.jianshu.com/go?to=https://www.redis.net.cn/order/3685.html" \t "https://www.jianshu.com/p/_blank)

有效的经度从-180度到180度。

有效的纬度从-85.05112878度到85.05112878度。

|  |
| --- |
| Getadd 添加地理位置 规则南北极无法添加，一般会通过java程序一次性导入  参数 key 值 (维度，精度，名称)  127.0.0.1:6379> GEOADD china:city 116.40 39.90 beijing  (integer) 1  127.0.0.1:6379> GEOADD china:city 121.47 31.23 shanghai  (integer) 1  127.0.0.1:6379> GEOADD china:city 106.50 29.53 chongqiang 114.05 22 shenzheng  (integer) 2  127.0.0.1:6379> GEOADD china:city 120.16 30.24 hangzhou 108.96 34.26 xian  (integer) 2  127.0.0.1:6379>  (error) ERR invalid longitude,latitude pair 39.900000,116.400000 超出范围  GEOPOS 命令 - 从key里返回所有给定位置元素的位置（经度和纬度）  127.0.0.1:6379> GEOPOS china:city beijing 获取指定的城市经度和纬度  1) 1) "116.39999896287918091"  2) "39.90000009167092543"  127.0.0.1:6379> GEOPOS china:city chongqiang  1) "106.49999767541885376"  2) "29.52999957900659211"  127.0.0.1:6379>  两个人之间的距离  GEODIST 命令 - 返回两个给定位置之间的距离  指定单位的参数 unit 必须是以下单位的其中一个：  m 表示单位为米。  km 表示单位为千米。  mi 表示单位为英里。  ft 表示单位为英尺。  127.0.0.1:6379> GEODIST china:city beijing shanghai 查看北京到上海的距离  "1067378.7564"  127.0.0.1:6379>  附近的人（获取所有附近的人地址，定位） 通过半径查询  GEORADIUS 命令 - 以给定的经纬度为中心， 找出某一半径内的元素  127.0.0.1:6379> GEORADIUS china:city 110 30 1000 km 获取以100 30这个经纬度为中心寻找1000KM的城市，  1) "chongqiang"  2) "xian"  3) "shenzheng"  4) "hangzhou"  127.0.0.1:6379> GEORADIUS china:city 110 30 300 km withdist 显示到中间的距离位置  (empty list or set)  127.0.0.1:6379> GEORADIUS china:city 110 30 500 km  1) "chongqiang"  2) "xian"  127.0.0.1:6379>  127.0.0.1:6379> GEORADIUS china:city 110 30 500 km withcoord 显示他人定位信息  1) 1) "chongqiang"  2) "341.9374"  3) 1) "106.49999767541885376"  2) "29.52999957900659211"  127.0.0.1:6379> GEORADIUS china:city 110 30 500 km withcoord withdist count 2 筛选指定的结果  1) 1) "chongqiang"  2) "341.9374"  3) 1) "106.49999767541885376"  2) "29.52999957900659211"  2) 1) "xian"  2) "483.8340"  3) 1) "108.96000176668167114"  2) "34.25999964418929977"  127.0.0.1:6379>  GEORADIUSBYMEMBER 命令 - 找出位于指定范围内的元素，中心点是由给定的位置元素决定  127.0.0.1:6379> GEORADIUSBYMEMBER china:city beijing 1000 km 找出位于指定元素周围的内容  1) "beijing"  2) "xian"  127.0.0.1:6379> GEORADIUSBYMEMBER china:city shanghai 400 km  1) "hangzhou"  2) "shanghai"  127.0.0.1:6379> GEOHASH 命令 - 返回一个或多个位置元素的 Geohash 表示 127.0.0.1:6379> GEOHASH china:city beijing 将二维的经纬度转为一维的字符串，两个字符串越接近，说明位置越接近  1) "wx4fbxxfke0"  127.0.0.1:6379> |

GEO原理：底层的实现原理其实就是Zset 所以我们可以使用Zset操作

|  |
| --- |
| 127.0.0.1:6379> ZRANGE china:city 0 -1 查看地图中全部元素  1) "chongqiang"  2) "xian"  3) "shenzheng"  4) "hangzhou"  5) "shanghai"  6) "beijing"  127.0.0.1:6379> ZREM china:city beijing 移除指定元素  (integer) 1  127.0.0.1:6379> ZRANGE china:city 0 -1  1) "chongqiang"  2) "xian"  3) "shenzheng"  4) "hangzhou"  5) "shanghai"  127.0.0.1:6379> |

## 2.Hyperloglog

简介：Redis2.8.9更新了Hyperloglog

Redis hyperLogLog 基数统计算法

优点：占用的内存是固定的2^64不同的元素的技术，只需要费12KB内存，如果要从内存角度比较的话HyperLogLog是首选

注意：0.81%的错误率

网页的UV 传统方式，set保存用户的ID，然后就可以统计Set中的元素数量作为标准判断

这种方式如果保存大量的数据，就会比较麻烦，我们的目的是为计数，而不是保存用户的ID

|  |
| --- |
| 127.0.0.1:6379> PFADD mykey a b c d e f g h j i 创建第一组元素  (integer) 1  127.0.0.1:6379> PFCOUNT mykey 统计基数数量  (integer) 10  127.0.0.1:6379> PFADD mykey2 i j z x c v b n m 创建第二组元素  (integer) 1  127.0.0.1:6379> PFCOUNT mykey2  (integer) 9  127.0.0.1:6379> PFMERGE mykey mykey2 合并两组元素  OK  127.0.0.1:6379> PFCOUNT mykey  (integer) 15  127.0.0.1:6379> |

## 3.Bitmap

位存储 统计用户信息，活跃，不活跃 两个状态都可以使用Bitmap

Bitmap位图 数据结构，都是二进制来进行记录，就只有0，和1两个状态

使用BitMaps记录打卡记录

|  |
| --- |
| 127.0.0.1:6379> SETBIT sign 0 1  (integer) 0  127.0.0.1:6379> SETBIT sign 1 0  (integer) 0  127.0.0.1:6379> SETBIT sign 2 0  (integer) 0  127.0.0.1:6379> SETBIT sign 3 1  (integer) 0  127.0.0.1:6379> SETBIT sign 4 1  (integer) 0  127.0.0.1:6379> SETBIT sign 5 1  (integer) 0  查看某一天是否打卡  127.0.0.1:6379> GETBIT sign 3  (integer) 1  127.0.0.1:6379> GETBIT sign 1  (integer) 0  统计操作 统计打卡天数  127.0.0.1:6379> BITCOUNT sign  (integer) 4 |

# Redis基本事务操作

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，在事务的执行过程中会按照顺序执行，一次性，顺序性，排他性，执行一系列的命令

Redis事务没有隔离级别的概念。所有的命令在事务中，并没有被执行，只有发起命令的时候才会执行Exec

Redis单条命令保证原子性 但是事务不保证原子性

|  |
| --- |
| Redis的事务：  开启事务（MULTI）  命令入队（其他命令）  执行事务（exec）  注意：Redis的事务是一次性的  127.0.0.1:6379> MULTI 开启事务  OK  127.0.0.1:6379> set k1 v1 命令入队  QUEUED  127.0.0.1:6379> set k2 v2 命令入队  QUEUED  127.0.0.1:6379> get k2 命令入队  QUEUED  127.0.0.1:6379> set k3 v3 命令入队  QUEUED  127.0.0.1:6379> exec 执行事务  1) OK  2) OK  3) "v2"  4) OK  127.0.0.1:6379>  127.0.0.1:6379> MULTI 开启事务  OK  127.0.0.1:6379> set k1 v1  QUEUED  127.0.0.1:6379> set k2 v2  QUEUED  127.0.0.1:6379> set k3 v4  QUEUED  127.0.0.1:6379> DISCARD 取消事务  OK  127.0.0.1:6379> get k4 事务队列中的命令都不会执行  (nil)  127.0.0.1:6379>  编译异常 命令有错 事务中所有的命令都不会执行  运行异常 如果事务队列中存在语法性错误，那么执行命令的时候，其他命令是可以正常执行，错误命令抛出异常  编译异常  127.0.0.1:6379> MULTI  OK  127.0.0.1:6379> set k1 v1  QUEUED  127.0.0.1:6379> set k2 v2  QUEUED  127.0.0.1:6379> set k3 v3  QUEUED  127.0.0.1:6379> GETSET k3 错误命令  (error) ERR wrong number of arguments for 'getset' command  127.0.0.1:6379> set k4 v4  QUEUED  127.0.0.1:6379> set k5 v5  QUEUED  127.0.0.1:6379> EXEC 执行事务报错了  (error) EXECABORT Transaction discarded because of previous errors.  127.0.0.1:6379> get k5 所有的命令都不会被执行  (nil)  127.0.0.1:6379>  运行异常  127.0.0.1:6379> set k1 "v1"  OK  127.0.0.1:6379> MULTI  OK  127.0.0.1:6379> INCR k1 会执行失败  QUEUED  127.0.0.1:6379> set k2 v2  QUEUED  127.0.0.1:6379> set k4 v3  QUEUED  127.0.0.1:6379> get k4  QUEUED  127.0.0.1:6379> EXEC  1) (error) ERR value is not an integer or out of range 虽然第一条命令保存了，但是依旧正常执行成功了  2) OK  3) OK  4) "v3"  127.0.0.1:6379> |

# Redis实现乐观锁

监控：watch （面试常问）

悲观锁：

很悲观，认为什么时候都可能出现问题，无论什么时候都加锁

乐观锁：

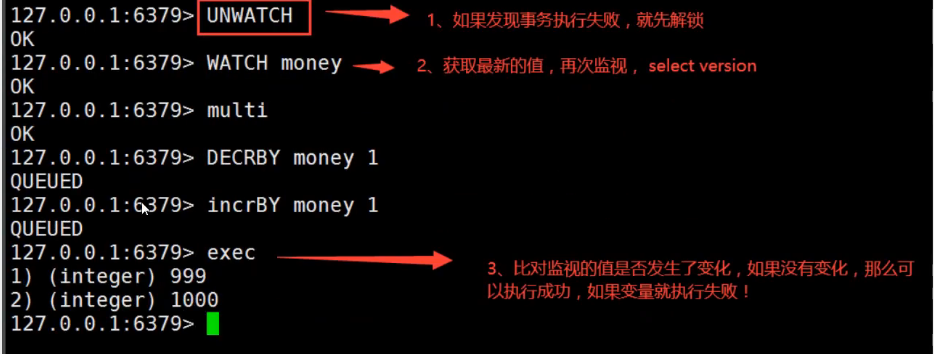
很乐观，认为什么都不会出现问题，所以不会加锁，更新数据的时候判断以下在此期间是否有人修改过这个数据。

获取version

更新的时候比较version

正常执行成功

|  |
| --- |
| 127.0.0.1:6379> set money 100  OK  127.0.0.1:6379> set out 0  OK  127.0.0.1:6379> WATCH money 监视money对象  OK  127.0.0.1:6379> MULTI 事务正常结束，数据期间没有变动，这个时候正常执行成功  OK  127.0.0.1:6379> DECRBY money 20  QUEUED  127.0.0.1:6379> INCRBY out 20  QUEUED  127.0.0.1:6379> EXEC  1) (integer) 80  2) (integer) 20  测试多线程修改值监视失败，使用watch当做redis的乐观锁操作  127.0.0.1:6379> WATCH money 监视money  OK  127.0.0.1:6379> MULTI  OK  127.0.0.1:6379> DECRBY money 10  QUEUED  127.0.0.1:6379> INCRBY out 10  QUEUED  127.0.0.1:6379> EXEC 执行之前另外一个线程修改我们的值这个时候就会导致食物执行失败  (nil)  如果修改失败获取最新的值就行 |



# Jedis操作Redis

我们要使用java来操作redis

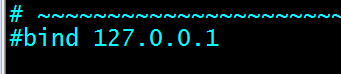
Jedis是redis官方推荐的java连接开发工具，使用java操作Redis中间件，

导入依赖

|  |
| --- |
| <dependencies>  <!-- 导入jedis的依赖包 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>  <!--导入fastjson-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.28</version>  </dependency> </dependencies> |

注意：Windows上连接Linux上的Redis需要对配置文件进行修改

１.修改redis.conf配置文件，将绑定的ip地址端口号给注释见下图



1. 由于Linux上的redis处于安全保护模式，这就让你无法从虚拟机外部去轻松建立连接， 这里就有两种解决方法，一种是在redis.conf中设置保护模式为no，见下图



编码测试：

连接数据库

|  |
| --- |
| //测试连接  Public static void main(String[] args) {  //1.new Jedis对象  Jedis jedis = new Jedis("192.168.2.110",6379);  //测试连接  System.out.println(jedis.ping()); } |

常用API

Key相关操作

|  |
| --- |
| */*\*\*  \* redis key相关操作  \*/ public class RedisKeyOperation {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  System.out.println(jedis.ping());  System.out.println("清空数据" + jedis.flushDB());  System.out.println("判断某个键是否存在" + jedis.exists("username"));  System.out.println("新增<'username','oldwang'>的键值对" + jedis.set("username","oldwang"));  System.out.println("新增<'password','password'>的键值对" + jedis.set("password","password"));  System.out.println("系统中所有的键如下");  Set<String> keys = jedis.keys("\*");  System.out.println(keys);  System.out.println("删除password" + jedis.del("password"));  System.out.println("判断password是否存在" + jedis.exists("password"));  System.out.println("查看username所存储的值的类型" + jedis.type("username"));  System.out.println("随机返回key空间一个" + jedis.randomKey());  System.out.println("重命名key" + jedis.rename("username","name"));  System.out.println("取出改名后的name" + jedis.get("name"));  System.out.println("按索引查询" + jedis.select(0));  System.out.println("删除当前选中数据库所有的key" + jedis.flushDB());  System.out.println("返回当前数据库中key的数目" + jedis.dbSize());  System.out.println("删除所有数据库中的所有key" + jedis.flushAll());  } } |

String相关操作

|  |
| --- |
| */*\*\*  \* String相关操作  \*/ public class RedisStringOperation {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  jedis.flushDB();  System.out.println("================增加数据===============");  System.out.println(jedis.set("key1","value1"));  System.out.println(jedis.set("key2","value2"));  System.out.println(jedis.set("key3","value3"));  System.out.println("删除key2" + jedis.del("key2"));  System.out.println("获取key2" + jedis.get("key2"));  System.out.println("修改key1的值" + jedis.set("key1","oldwangchange"));  System.out.println("在key3后边加入值"+ jedis.append("key3","append"));  System.out.println("key3的值"+jedis.get("key3"));  System.out.println("增加多个键值对" + jedis.mset("key01","value01","key02","value02"));  System.out.println("获取多个键值对" + jedis.mget("key01","key02"));  System.out.println("删除多个键值对" + jedis.del("key01","key02"));  System.out.println("获取多个键值对" + jedis.mget("key01","key02"));   jedis.flushDB();  System.out.println("====================新增键值对防止覆盖原来值===============");  System.out.println(jedis.setnx("key1","value1"));  System.out.println(jedis.setnx("key2","value2"));  System.out.println(jedis.setnx("key2","value-new"));  System.out.println(jedis.get("key1"));  System.out.println(jedis.get("key2"));   System.out.println("====================新增键值设置有效时间===============");  System.out.println(jedis.setex("key3",2,"value3"));  System.out.println(jedis.get("key3"));  try {  TimeUnit.SECONDS.sleep(3);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println(jedis.get("key3"));   System.out.println("====================获取原值，更新新值===============");  System.out.println(jedis.getSet("key2","key2GetSet"));  System.out.println(jedis.get("key2"));  System.out.println("获取key的字符串" + jedis.getrange("key2",2,4));  } } |

List操作

|  |
| --- |
| */*\*\*  \* List相关操作  \*/ public class RedisListOperation {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  jedis.flushDB();   System.out.println("======添加List=====");  jedis.lpush("collections","ArrayList","Vector","stack");  jedis.lpush("collections","HashSet");  jedis.lpush("collections","TreeSet");  jedis.lpush("collections","TreeMap");  System.out.println("collections的内容" + jedis.lrange("collections",0,-1));  System.out.println("collections区间0-3的元素" + jedis.lrange("collections",0,3));  System.out.println("==============================");  //删除列表指定的值，第二个参数为删除的个数（有重复时） 后add进去的元素先被删除，类似于出栈  System.out.println("删除指定元素个数" + jedis.lrem("collections",2,"HashSet"));  System.out.println("删除0-3之间之外的元素" + jedis.ltrim("collections",0,3));  System.out.println("collections列表出栈(左端)" + jedis.lpop("collections"));  } } |

其余API与所学命令一致

# 通过Jedis再次理解事务

|  |
| --- |
| Public class Transactional {  public static void main(String[] args) {  Jedis jedis = new Jedis("192.168.2.110",6379);  jedis.flushDB();  JSONObject jsonObject = new JSONObject();  jsonObject.put("hello" ,"oldwang");  jsonObject.put("name","old");  //开启事务  Transaction multi = jedis.multi();  String string = jsonObject.toJSONString();  jedis.watch(string);  //事务操作  try{  multi.set("user1",string);  int i = 1 /0; //抛出异常 事务执行失败  multi.set("user2",string);  multi.exec();  }catch (Exception e){  multi.discard(); // 放弃事务  e.printStackTrace();  }finally {  System.out.println(jedis.get("user1"));  System.out.println(jedis.get("user2"));  //关闭连接  jedis.close();  }  } } |

# SpringBoot整合Redis

说明;在springboot-2.x之后原来使用的Jedis被替换了lettuce

Jedis：底层采用的直接连。多线程操作不安全，想要避免不安全，使用Jedispool连接池

Lettuce：底层采用netty实例可以在多个线程中共享，不存在线程不安全情况，可以减少线程数量，

源码分析：

|  |
| --- |
| @Bean @ConditionalOnMissingBean(name = "redisTemplate") // 我们可以自己定义一个 redisTemplate来替换这个默认的！ public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) throws UnknownHostException { // 默认的 RedisTemplate 没有过多的设置，redis 对象都是需要序列化！ // 两个泛型都是 Object, Object 的类型，我们后使用需要强制转换 <String, Object> RedisTemplate<Object, Object> template = new RedisTemplate<>(); template.setConnectionFactory(redisConnectionFactory); return template; }  @Bean @ConditionalOnMissingBean // 由于 String 是redis中最常使用的类型，所以说单独提出来了一 个bean！ public StringRedisTemplate stringRedisTemplate(RedisConnectionFactory redisConnectionFactory) throws UnknownHostException { StringRedisTemplate template = new StringRedisTemplate(); template.setConnectionFactory(redisConnectionFactory); return template; |

1.导入依赖

|  |
| --- |
| <!--操作redis--> <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId> </dependency> |

2.配置连接

|  |
| --- |
| # 配置redis spring.redis.host=192.168.2.110  spring.redis.port= 6379 |

测试

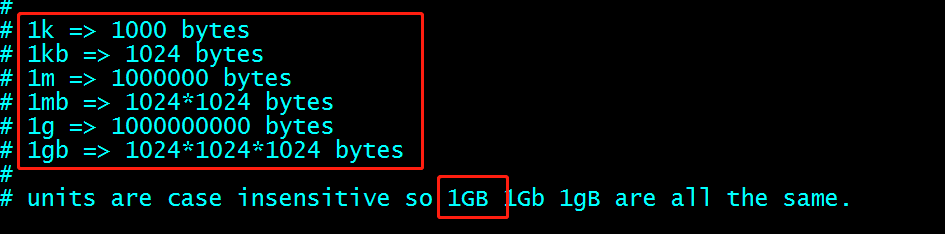
|  |
| --- |
| @SpringBootTest public class RedisSpringBootTest {   @Autowired  private RedisTemplate redisTemplate;   @Test  public void test(){  //操作字符串 相当于Spring  //opsForList 操作List  //opsForSet 操作Set  //opsForHash 操作hash  //opsForZSet 操作Zset  //opsForGeo 操作Geo  //opsForHyperLogLog 操作hyperLogLog   //除了基本操作 我们常用的方法都可以直接通过redisTemplate操作，  比如事务和基本的CRUD   //获取连接  RedisConnectionFactory factory = redisTemplate.getConnectionFactory();   ValueOperations forValue = redisTemplate.opsForValue();  forValue.set("key","oldwang");  System.out.println(forValue.get("key"));  }  } |

# 自定义RedisTemplate

|  |
| --- |
| package com.oldwang.redisspringboot.config; @Configuration public class RedisConfig {  //编写自己的redisTemplate  // 自己定义了一个 RedisTemplate  @Bean  @SuppressWarnings("all")  public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {  // 我们为了自己开发方便，一般直接使用 <String, Object>  RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();  template.setConnectionFactory(factory);  // Json序列化配置  Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);  ObjectMapper om = new ObjectMapper();  om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);  om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);  jackson2JsonRedisSerializer.setObjectMapper(om);  // String 的序列化  StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();  // key采用String的序列化方式  template.setKeySerializer(stringRedisSerializer);  // hash的key也采用String的序列化方式  template.setHashKeySerializer(stringRedisSerializer);  // value序列化方式采用jackson  template.setValueSerializer(jackson2JsonRedisSerializer);  // hash的value序列化方式采用jackson  template.setHashValueSerializer(jackson2JsonRedisSerializer);  template.afterPropertiesSet();  return template;  } } |

# redis配置文件详解

单位



配置文件unit对大小写不敏感

就好比我们学习spring的时候 import include标签

|  |
| --- |
| 网络  bind 127 .0.0.1 # 绑定的ip protected-mode yes # 保护模式 port 6379 # 端口设置  通用  daemonize yes # 以守护进程的方式运行，默认是 no，我们需要自己开启为yes！  pidfile /var/run/redis\_6379.pid # 如果以后台的方式运行，我们就需要指定一个 pid 文件！ # debug (a lot of information, useful for development/testing) # verbose (many rarely useful info, but not a mess like the debug level) # notice (moderately verbose, what you want in production probably) 生产环境 # warning (only very important / critical messages are logged) loglevel notice logfile "" # 日志的文件位置名 databases 16 # 数据库的数量，默认是 16 个数据库 always-show-logo yes # 是否总是显示LOGO  stop-writes-on-bgsave-error yes # 持久化如果出错，是否还需要继续工作！ rdbcompression yes # 是否压缩 rdb 文件，需要消耗一些cpu资源！ rdbchecksum yes # 保存rdb文件的时候，进行错误的检查校验！ dir ./ # rdb 文件保存的目录！  APPEND ONLY 模式 aof配置  # 日志 # Specify the server verbosity level. # This can be one of:  Database 16 数据库的数量 默认是16个  安全相关  127 .0.0.1:6379> ping PONG 127 .0.0.1:6379> config get requirepass # 获取redis的密码 1 ) "requirepass" 2 ) "" 127 .0.0.1:6379> config set requirepass "123456" # 设置redis的密码 OK 127 .0.0.1:6379> config get requirepass # 发现所有的命令都没有权限了 (error) NOAUTH Authentication required. 127 .0.0.1:6379> ping (error) NOAUTH Authentication required. 127 .0.0.1:6379> auth 123456 # 使用密码进行登录！ OK 127 .0.0.1:6379> config get requirepass 1 ) "requirepass" 2 ) "123456"  限制相关  maxclients 10000 # 设置能连接上redis的最大客户端的数量 maxmemory <bytes> # redis 配置最大的内存容量  内存相关  maxmemory-policy noeviction # 内存到达上限之后的处理策略 1 、volatile-lru：只对设置了过期时间的key进行LRU（默认值） 2 、allkeys-lru ： 删除lru算法的key 3 、volatile-random：随机删除即将过期key 4 、allkeys-random：随机删除 5 、volatile-ttl ： 删除即将过期的 6 、noeviction ： 永不过期，返回错误 |

快照：在规定的时间内指定了多少次操作，则会持久化到文件.rdb .aof

Redis是内存数据库，如果没有持久化，name数据断点即会失去数据

Save 900 1 如果900秒内 至少有一个key进行操作我们进行持久化

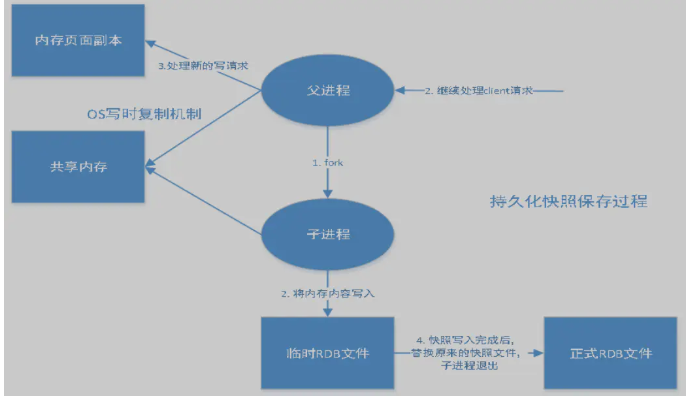
Save 300 10 如果900秒内 至少有十个key进行操作我们进行持久化

Save 60 10000 如果60秒内 至少有一万个key进行操作我们进行持久化

# Redis持久化操作

Redis 是内存数据库，如果不将内存中的数据库状态保存到磁盘，那么一旦服务器进程退出，服务器中  
的数据库状态也会消失。所以 Redis 提供了持久化功能！

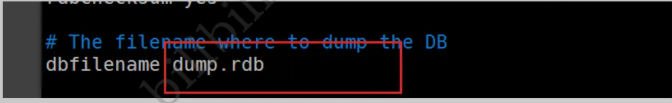
## RDB （Redis database）

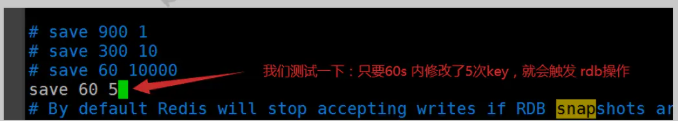


Redis会单独创建（fork）一个子进程来进行持久化，会先将数据写入到一个临时文件中，待持久化过程都结束了，再用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何IO操作的。这就确保了极高的性能。如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那RDB方式要比AOF方式更加的高效。RDB的缺点是最后一次持久化后的数据可能丢失。我们默认的就是。

RDB，一般情况下不需要修改这个配置！有时候在生产环境我们会将这个文件进行备份！

rdb保存的文件是dump.rdb 都是在我们的配置文件中快照中进行配置的！





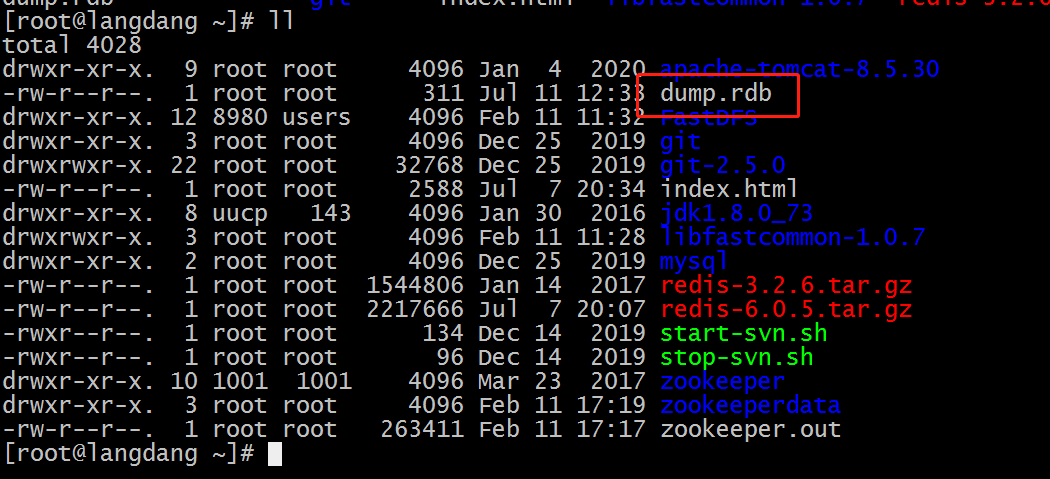
触发机制

1 、save的规则满足的情况下，会自动触发rdb规则

2 、执行 flushall 命令，也会触发我们的rdb规则！

3 、退出redis，也会产生 rdb 文件！

备份就自动生成一个 dump.rdb



恢复RDB：

1 、只需要将rdb文件放在我们redis启动目录就可以，redis启动的时候会自动检查dump.rdb 恢复其中的数据！

2 、查看需要存在的位置几乎就他自己默认的配置就够用了，但是我们还是需要去学习！

优点：

1 、适合大规模的数据恢复！

2 、对数据的完整性要不高！

缺点：

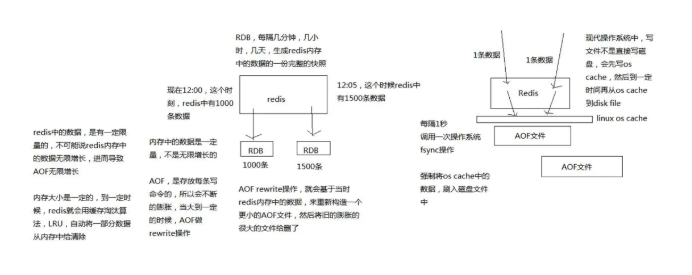
1 、需要一定的时间间隔进程操作！如果redis意外宕机了，这个最后一次修改数据就没有的了！

2 、fork进程的时候，会占用一定的内容空间！！

## AOF（Append Only File）

appendonly no # 默认是不开启aof模式的，默认是使用rdb方式持久化的，在大部分所有的情况下，rdb完全够用！  
appendfilename "appendonly.aof" # 持久化的文件的名字  
# appendfsync always # 每次修改都会 sync。消耗性能  
appendfsync everysec # 每秒执行一次 sync，可能会丢失这1s的数据！  
# appendfsync no # 不执行 sync，这个时候操作系统自己同步数据，速度最快！  
# rewrite 重写，

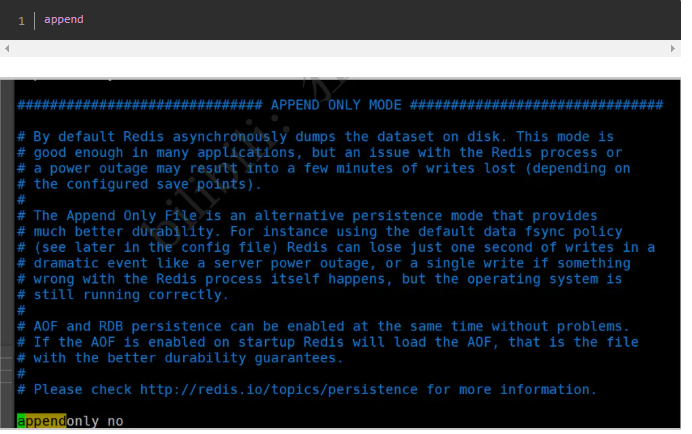
将我们的所有命令都记录下来，history，恢复的时候就把这个文件全部在执行一遍！



127 .0.0.1:6379> config get dir  
1 ) "dir"  
2 ) "/usr/local/bin" # 如果在这个目录下存在 dump.rdb 文件，启动就会自动恢复其中的数据

以日志的形式来记录每个写操作，将Redis执行过的所有指令记录下来（读操作不记录），只许追加文件  
但不可以改写文件，redis启动之初会读取该文件重新构建数据，换言之，redis重启的话就根据日志文件  
的内容将写指令从前到后执行一次以完成数据的恢复工作

Aof保存的是 appendonly.aof 文件

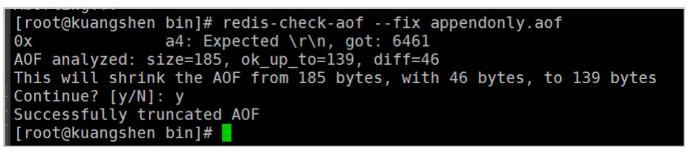


默认是不开启的，我们需要手动进行配置！我们只需要将 appendonly 改为yes就开启了 aof！

重启，redis 就可以生效了！

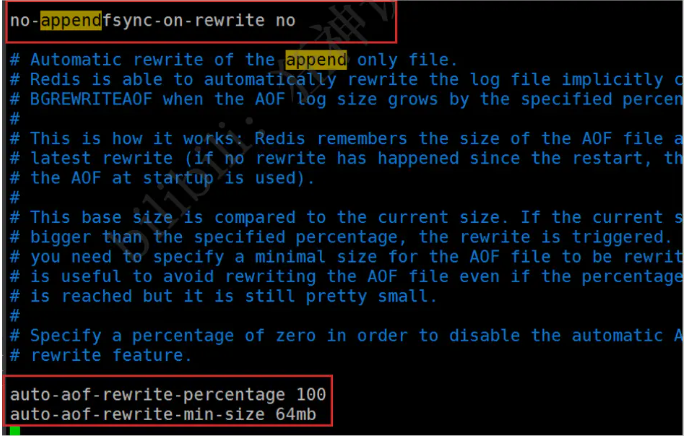
如果这个 aof 文件有错误，这时候 redis 是启动不起来的吗，我们需要修复这个aof文件

redis 给我们提供了一个工具 redis-check-aof --fix



重写规则说明

aof 默认就是文件的无限追加，文件会越来越大！



如果 aof 文件大于 64m，太大了！ fork一个新的进程来将我们的文件进行重写！

优点和缺点！

优点：

1 、每一次修改都同步，文件的完整会更加好！

2 、每秒同步一次，可能会丢失一秒的数据

3 、从不同步，效率最高的！

缺点：

1 、相对于数据文件来说，aof远远大于 rdb，修复的速度也比 rdb慢！

2 、Aof 运行效率也要比 rdb 慢，所以我们redis默认的配置就是rdb持久化！

扩展：

1 、RDB 持久化方式能够在指定的时间间隔内对你的数据进行快照存储

2 、AOF 持久化方式记录每次对服务器写的操作，当服务器重启的时候会重新执行这些命令来恢复原始的数据，AOF命令以Redis 协议追加保存每次写的操作到文件末尾，Redis还能对AOF文件进行后台重  
写，使得AOF文件的体积不至于过大。

3 、只做缓存，如果你只希望你的数据在服务器运行的时候存在，你也可以不使用任何持久化

4 、同时开启两种持久化方式

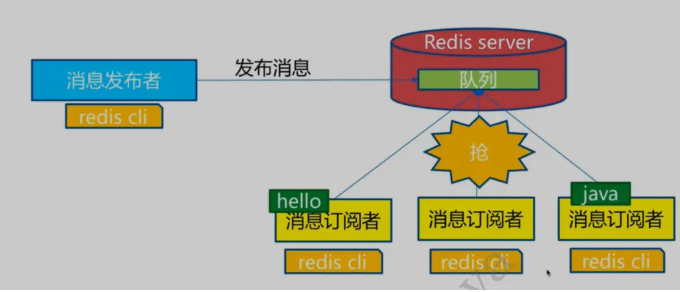
在这种情况下，当redis重启的时候会优先载入AOF文件来恢复原始的数据，因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整。RDB 的数据不实时，同时使用两者时服务器重启也只会找AOF文件，那要不要只使用AOF呢？作者  
建议不要，因为RDB更适合用于备份数据库（AOF在不断变化不好备份），快速重启，而且不会有AOF可能潜在的Bug，留着作为一个万一的手段。

5 、性能建议

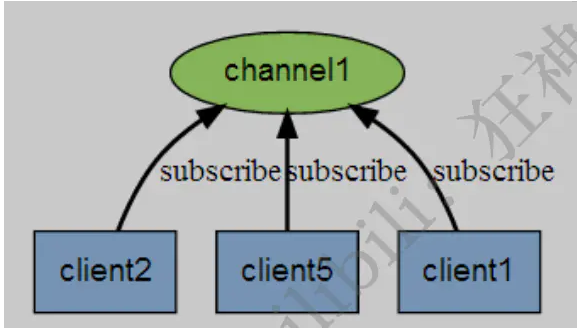
因为RDB文件只用作后备用途，建议只在Slave上持久化RDB文件，而且只要 15 分钟备份一次就够了，只保留 save 900 1 这条规则。如果Enable AOF ，好处是在最恶劣情况下也只会丢失不超过两秒数据，启动脚本较简单只load自己的AOF文件就可以了，代价一是带来了持续的IO，二是AOF rewrite 的最后将 rewrite 过程中产生的新数据写到新文件造成的阻塞几乎是不可避免的。只要硬盘许可，应该尽量减少AOF rewrite的频率，AOF重写的基础大小默认值64M太小了，可以设到5G以上，默认超过原大小100%大小重  
写可以改到适当的数值。如果不Enable AOF ，仅靠 Master-Slave Repllcation 实现高可用性也可以，能省掉一大笔IO，也减少了rewrite时带来的系统波动。代价是如果Master/Slave 同时倒掉，会丢失十几分钟的数据，启动脚本也要比较两个 Master/Slave 中的 RDB文件，载入较新的那个，微博就是这种架构。

# Redis发布订阅

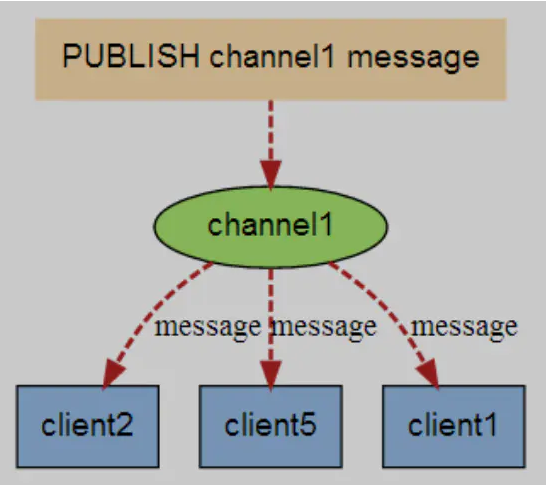
Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。微信、微博、关注系统！  
Redis 客户端可以订阅任意数量的频道。  
订阅/发布消息图：第一个：消息发送者， 第二个：频道 第三个：消息订阅者！



下图展示了频道 channel1 ， 以及订阅这个频道的三个客户端 —— client2 、 client5 和 client1 之间的  
关系：



当有新消息通过 PUBLISH 命令发送给频道 channel1 时， 这个消息就会被发送给订阅它的三个客户  
端：



命令

这些命令被广泛用于构建即时通信应用，比如网络聊天室(chatroom)和实时广播、实时提醒等



测试

订阅端：

发送端：

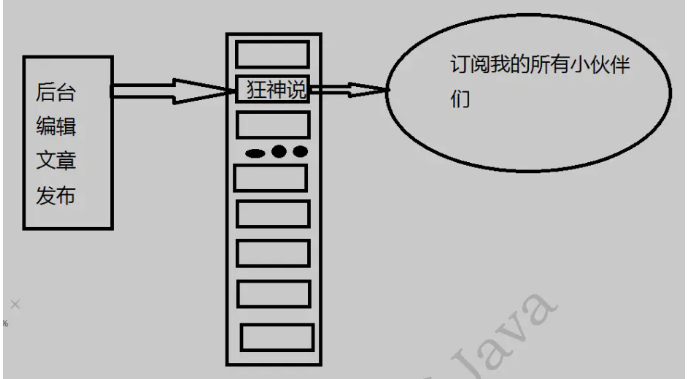
原理

Redis是使用C实现的，通过分析 Redis 源码里的 pubsub.c 文件，了解发布和订阅机制的底层实现，籍  
此加深对 Redis 的理解。

|  |
| --- |
| 127 .0.0.1:6379> SUBSCRIBE oldwang # 订阅一个频道 oldwang  Reading messages... (press Ctrl-C to quit) 1 ) "subscribe" 2 ) "oldwang" 3 ) (integer) 1 # 等待读取推送的信息 1 ) "message" # 消息 2 ) "oldwang" # 那个频道的消息 3 ) "hello,oldwang" # 消息的具体内容 1 ) "message" 2 ) "oldwang" 3 ) "hello,redis"  发送端 127 .0.0.1:6379> PUBLISH oldwang "hello,oldwang" # 发布者发布消息到频道！ (integer) 1 127 .0.0.1:6379> PUBLISH oldwang "hello,redis" # 发布者发布消息到频道！ (integer) 1 127 .0.0.1:6379> |

Redis 通过 PUBLISH 、SUBSCRIBE 和 PSUBSCRIBE 等命令实现发布和订阅功能。

微信：通过 SUBSCRIBE 命令订阅某频道后，redis-server 里维护了一个字典，字典的键就是一个个 频道！，而字典的值则是一个链表，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字典中查找记录了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，你可以设定对某一个key值进行消息发布及消息订阅，当一个key值上进行了消息发布后，所有订阅它的客户端都会收到相应的消息。这一功能最明显的用法就是用作实时消息系统，比如普通的即时聊天，群聊等功能。

使用场景：

1 、实时消息系统！

2 、事实聊天！（频道当做聊天室，将信息回显给所有人即可！）

3 、订阅，关注系统都是可以的！稍微复杂的场景我们就会使用 消息中间件 MQ （）

# redis的主从环境搭建

概念：

主从复制，是指将一台Redis服务器的数据，复制到其他的Redis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Redis服务器都是主节点；且一个主节点可以有多个从节点(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：

1 、数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。

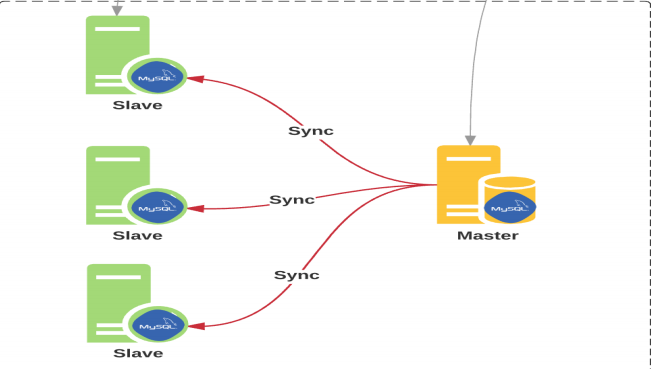
2 、故障恢复：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复；实际上是一种服务的冗余。

3 、负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写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，可以大大提高Redis服务器的并发量。

4 、高可用（集群）基石：除了上述作用以外，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高可用的基础。一般来说，要将Redis运用于工程项目中，只使用一台Redis是万万不能的（宕机），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的请求负载，压力较大；

2 、从容量上，单个Redis服务器内存容量有限，就算一台Redis服务器内存容量为256G，也不能将所有内存用作Redis存储内存，一般来说，单台Redis最大使用内存不应该超过20G。电商网站上的商品，一般都是一次上传，无数次浏览的，说专业点也就是"多读少写"。对于这种场景，我们可以使如下这种架构：



环境配置：

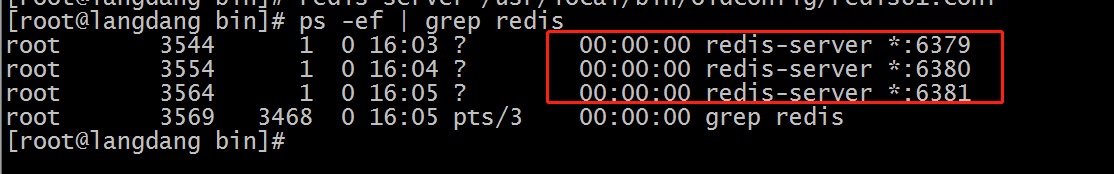
只配置从库，不用配置主库

|  |
| --- |
| 127.0.0.1:6379> info 查看当前库的信息  # Replication  role:master 角色  connected\_slaves:0 没有从机  master\_repl\_offset:0  repl\_backlog\_active:0  repl\_backlog\_size:1048576  repl\_backlog\_first\_byte\_offset:0  repl\_backlog\_histlen:0 |

复制3个配置文件然后修改对应的信息

1. 端口
2. PId名字
3. Log文件名字
4. Dump.rdb 名字

修改之后启动三个redis服务，通过进程查看



一主二从

默认情况下，每台redis服务器都是主节点

一般情况下只用配置从机就可以

一主（6379）二从（6380,6381）

|  |
| --- |
| 127.0.0.1:6381> SLAVEOF 127.0.0.1 6379 slaveof host 6379 找谁当老大  OK  127.0.0.1:6381>  127.0.0.1:6381> info replication  # Replication  role:slave 当前角色是从机  master\_host:127.0.0.1 可以看到主机的信息  master\_port:6379  master\_link\_status:up  master\_last\_io\_seconds\_ago:5  master\_sync\_in\_progress:0  slave\_repl\_offset:169  slave\_priority:100  slave\_read\_only:1  connected\_slaves:0  master\_repl\_offset:0  repl\_backlog\_active:0  repl\_backlog\_size:1048576  repl\_backlog\_first\_byte\_offset:0  repl\_backlog\_histlen:0  在主机中可以看到从节点  127.0.0.1:6379> info replication  # Replication  role:master  connected\_slaves:2  slave0:ip=127.0.0.1,port=6381,state=online,offset=323,lag=1  slave1:ip=127.0.0.1,port=6380,state=online,offset=323,lag=1  master\_repl\_offset:323  repl\_backlog\_active:1  repl\_backlog\_size:1048576  repl\_backlog\_first\_byte\_offset:2  repl\_backlog\_histlen:322  127.0.0.1:6379> |

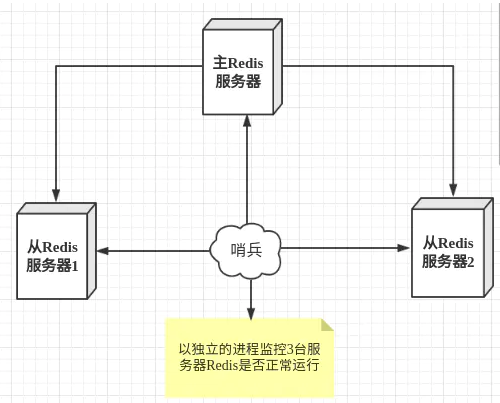
真实的配置应该在配置文件中配置，这样是永久的 命令行配置是临时的

|  |
| --- |
| REPLICATION 主从配置    细节：  主机可以写，从机不能写只负责读取数据，主机中的所有信息和数据，都会自动保存到从机中    主机断开连接从机依旧连接到主机但是没有写操作了，这个时候主机如果回来了从机依旧可以直接获取主机写入的信息  如果使用命令行配置的主从，这个时候重启了就会变成主节点，只要变为从机立马就会从主机中获取值  复制原理：  Slave 启动成功连接到 master 后会发送一个sync同步命令  Master 接到命令，启动后台的存盘进程，同时收集所有接收到的用于修改数据集命令，在后台进程执行完毕之后，master将传送整个数据文件到slave，并完成一次完全同步。  全量复制：  而slave服务在接收到数据库文件数据后，将其存盘并加载到内存中。  增量复制：  Master 继续将新的所有收集到的修改命令依次传给slave，完成同步  但是只要是重新连接master，一次完全同步（全量复制）将被自动执行！ 我们的数据一定可以在从机中看到！  宕机后手动配置主机：  层层链路  上一个M链接下一个 S！    这时候也可以完成我们的主从复制！  如果没有老大了，这个时候能不能选择一个老大出来呢？ 手动！  谋朝篡位  如果主机断开了连接，我们可以使用 SLAVEOF no one 让自己变成主机！其他的节点就可以手动连接到最新的这个主节点（手动）！如果这个时候老大修复了，那就重新连接！ |

# 哨兵（自动选举master）模式详解

概述：

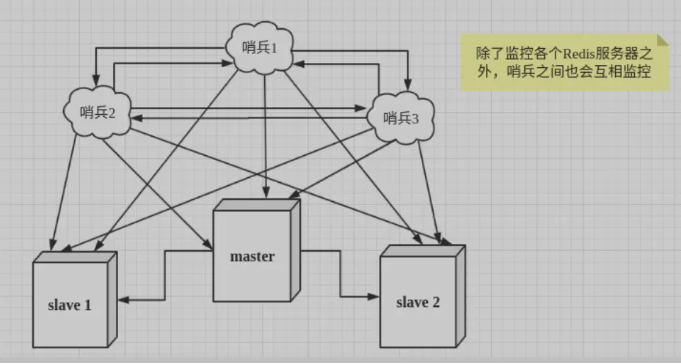
主从切换技术的方法是：当主服务器宕机后，需要手动把一台从服务器切换为主服务器，这就需要人工干预，费事费力，还会造成一段时间内服务不可用。这不是一种推荐的方式，更多时候，我们优先考虑哨兵模式。Redis从2.8开始正式提供了Sentinel（哨兵） 架构来解决这个问题。谋朝篡位的自动版，能够后台监控主机是否故障，如果故障了根据投票数自动将从库转换为主库。哨兵模式是一种特殊的模式，首先Redis提供了哨兵的命令，哨兵是一个独立的进程，作为进程，它会独立运行。其原理是 哨兵通过发送命令，等待Redis服务器响应，从而监控运行的多个Redis实例。



这里的哨兵有两个作用

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。  
当哨兵监测到master宕机，会自动将slave切换成master，然后通过 发布订阅模式 通知其他的从服务器，修改配置文件，让它们切换主机。

而一个哨兵进程对Redis服务器进行监控，可能会出现问题，为此，我们可以使用多个哨兵进行监控。各个哨兵之间还会进行监控，这样就形成了多哨兵模式。



假设主服务器宕机，哨兵 1 先检测到这个结果，系统并不会马上进行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，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** 。

测试：

我们目前的状态是一主二从

1. 配置哨兵配置文件sentinel.conf

sentinel monitor 主机名称host port 1 后面这个1代表主机挂了 slave投票接替主机

sentinel monitor masterredis 127.0.0.1 6379 1

1. 启动哨兵

Redis-sentinel sentinel.conf

|  |
| --- |
| redis-sentinel redis-server  [root@langdang bin]# redis-sentinel sentinel.conf  3722:X 11 Jul 16:54:50.262 \* Increased maximum number of open files to 10032 (it was originally set to 1024).  \_.\_  \_.-``\_\_ ''-.\_  \_.-`` `. `\_. ''-.\_ Redis 3.2.6 (00000000/0) 64 bit  .-`` .-```. ```\/ \_.,\_ ''-.\_  ( ' , .-` | `, ) Running in sentinel mode  |`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 26379  | `-.\_ `.\_ / \_.-' | PID: 3722  `-.\_ `-.\_ `-./ \_.-' \_.-'  |`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|  | `-.\_`-.\_ \_.-'\_.-' | http://redis.io  `-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'  |`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|  | `-.\_`-.\_ \_.-'\_.-' |  `-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'  `-.\_ `-.\_\_.-' \_.-'  `-.\_ \_.-'  `-.\_\_.-'  3722:X 11 Jul 16:54:50.265 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.  3722:X 11 Jul 16:54:50.276 # Sentinel ID is 47ec027eb5a07908e4da87482a23d8111fe63243  3722:X 11 Jul 16:54:50.276 # +monitor master masterredis 127.0.0.1 6379 quorum 1  3722:X 11 Jul 16:54:50.276 \* +slave slave 127.0.0.1:6381 127.0.0.1 6381 @ masterredis 127.0.0.1 6379  3722:X 11 Jul 16:54:50.278 \* +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ masterredis 127.0.0.1 6379  如果主节点断开了这个时候就会从从机中随机选择一个服务器    哨兵日志    如果主机此时回来了，只能归并到新的主机下，当做从机，这就是哨兵模式的规则！ |

哨兵模式的优缺点：

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 不好啊在线扩容的，集群容量一旦到达上限，在线扩容就十分麻烦！

2 、实现哨兵模式的配置其实是很麻烦的，里面有很多选择！

**哨兵模式的全部配置！**

|  |
| --- |
| # Example sentinel.conf # 哨兵sentinel实例运行的端口 默认 26379 port 26379 # 哨兵sentinel的工作目录 dir /tmp # 哨兵sentinel监控的redis主节点的 ip port # master-name 可以自己命名的主节点名字 只能由字母A-z、数字0-9 、这三个字符".-\_"组成。 # quorum 配置多少个sentinel哨兵统一认为master主节点失联 那么这时客观上认为主节点失联了 # sentinel monitor <master-name> <ip> <redis-port> <quorum> sentinel monitor mymaster 127 .0.0.1 6379 2 # 当在Redis实例中开启了requirepass foobared 授权密码 这样所有连接Redis实例的客户端都要提供 密码 # 设置哨兵sentinel 连接主从的密码 注意必须为主从设置一样的验证密码 # sentinel auth-pass <master-name> <password> sentinel auth-pass mymaster MySUPER--secret-0123passw0rd # 指定多少毫秒之后 主节点没有应答哨兵sentinel 此时 哨兵主观上认为主节点下线 默认 30 秒 # sentinel down-after-milliseconds <master-name> <milliseconds> sentinel down-after-milliseconds mymaster 30000 # 这个配置项指定了在发生failover主备切换时最多可以有多少个slave同时对新的master进行 同步， 这个数字越小，完成failover所需的时间就越长， 但是如果这个数字越大，就意味着越 多的slave因为replication而不可用。 可以通过将这个值设为 1 来保证每次只有一个slave 处于不能处理命令请求的状态。 # sentinel parallel-syncs <master-name> <numslaves> sentinel parallel-syncs mymaster 1 # 故障转移的超时时间 failover-timeout 可以用在以下这些方面： #1. 同一个sentinel对同一个master两次failover之间的间隔时间。 #2. 当一个slave从一个错误的master那里同步数据开始计算时间。直到slave被纠正为向正确的master那 里同步数据时。 #3.当想要取消一个正在进行的failover所需要的时间。 #4.当进行failover时，配置所有slaves指向新的master所需的最大时间。不过，即使过了这个超时， slaves依然会被正确配置为指向master，但是就不按parallel-syncs所配置的规则来了 # 默认三分钟 # sentinel failover-timeout <master-name> <milliseconds> |

# **18.Redis缓存穿透和雪崩**

Redis缓存的使用，极大的提升了应用程序的性能和效率，特别是数据查询方面。但同时，它也带来了一些问题。其中，最要害的问题，就是数据的一致性问题，从严格意义上讲，这个问题无解。如果对数据的一致性要求很高，那么就不能使用缓存。另外的一些典型问题就是，缓存穿透、缓存雪崩和缓存击穿。目前，业界也都有比较流行的解决方案。

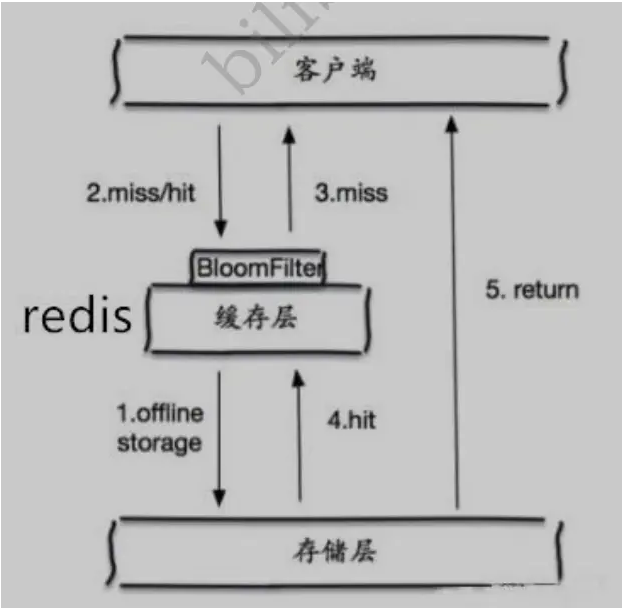
缓存穿透：

概念：

缓存穿透的概念很简单，用户想要查询一个数据，发现redis内存数据库没有，也就是缓存没有命中，于是向持久层数据库查询。发现也没有，于是本次查询失败。当用户很多的时候，缓存都没有命中（秒杀！），于是都去请求了持久层数据库。这会给持久层数据库造成很大的压力，这时候就相当于出现了缓存穿透。

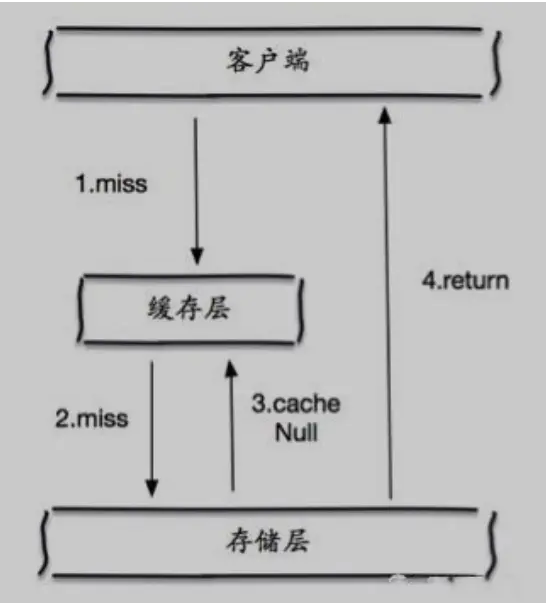
解决方案：

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，从而避免了对底层存储系统的查询压力；



缓存空对象当存储层不命中后，即使返回的空对象也将其缓存起来，同时会设置一个过期

间，之后再访问这个数



据将会从缓存中获取，保护了后端数据源；但是这种方法会存在两个问题：

1 、如果空值能够被缓存起来，这就意味着缓存需要更多的空间存储更多的键，因为这当中可能会有很多的空值的键；

2 、即使对空值设置了过期时间，还是会存在缓存层和存储层的数据会有一段时间窗口的不一致，这对于需要保持一致性的业务会有影响。缓存击穿（量太大，缓存过期！）

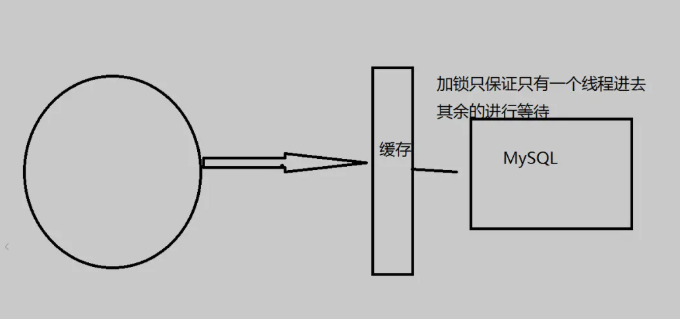
概述：这里需要注意和缓存击穿的区别，缓存击穿，是指一个key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就穿破缓存，直 接请求数据库，就像在一个屏障上凿开了一个洞。当某个key在过期的瞬间，有大量的请求并发访问，这类数据一般是热点数据，由于缓存过期，会同时访问数据库来查询最新数据，并且回写缓存，会导使数据库瞬间压力过大。

解决方案：

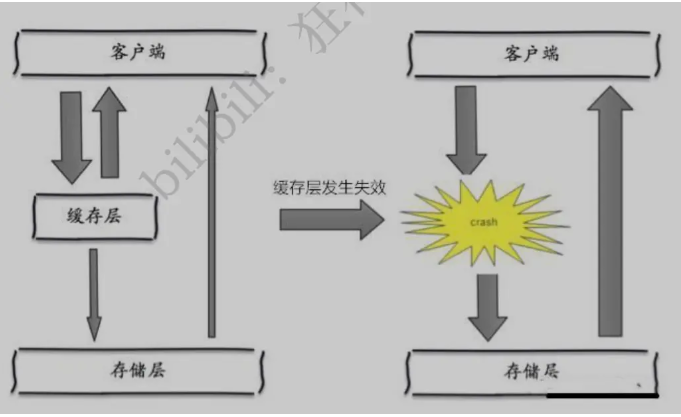
设置热点数据永不过期从缓存层面来看，没有设置过期时间，所以不会出现热点 key 过期后产生的问题。

加互斥锁

分布式锁：使用分布式锁，保证对于每个key同时只有一个线程去查询后端服务，其他线程没有获得分布式锁的权限，因此只需要等待即可。这种方式将高并发的压力转移到了分布式锁，因此对分布式锁的考验很大。



缓存雪崩：是指在某一个时间段，缓存集中过期失效。Redis 宕机！产生雪崩的原因之一，比如在写本文的时候，马上就要到双十二零点，很快就会迎来一波抢购，这波商品时间比较集中的放入了缓存，假设缓存一个小时。那么到了凌晨一点钟的时候，这批商品的缓存就都过期了。而对这批商品的访问查询，都落到了数据库上，对于数据库而言，就会产生周期性的压力波峰。于是所有的请求都会达到存储层，存储层的调用量会暴增，造成存储层也会挂掉的情况。

其实集中过期，倒不是非常致命，比较致命的缓存雪崩，是缓存服务器某个节点宕机或断网。因为自然形成的缓存雪崩，一定是在某个时间段集中创建缓存，这个时候，数据库也是可以顶住压力的。无非就是对数据库产生周期性的压力而已。而缓存服务节点的宕机，对数据库服务器造成的压力是不可预知的，很有可能瞬间就把数据库压垮。

**解决方案：**

**Redis的高可用**

这个思想的含义是，既然redis有可能挂掉，那我多增设几台redis，这样一台挂掉之后其他的还可以继续工作，其实就是搭建的集群。（异地多活！）

**限流降级：**

这个解决方案的思想是，在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。

**数据预热：**

数据加热的含义就是在正式部署之前，我先把可能的数据先预先访问一遍，这样部分可能大量访问的数据就会加载到缓存中。在即将发生大并发访问前手动触发加载缓存不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀。