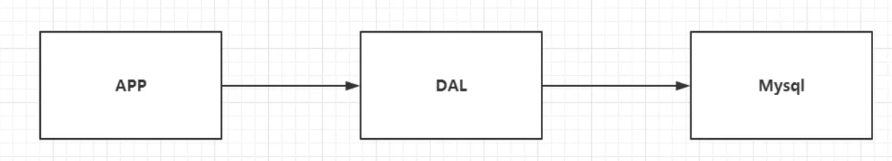
**（虚拟机的redis存放路径/usr/local/bin）**

**Nosql概述：**

为什么要用nosql：我们现在处理什么年代2020年，大数据时代;大数据一般的数据库无法进行分析处理了!

1. **单机MgSQL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 ！

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~

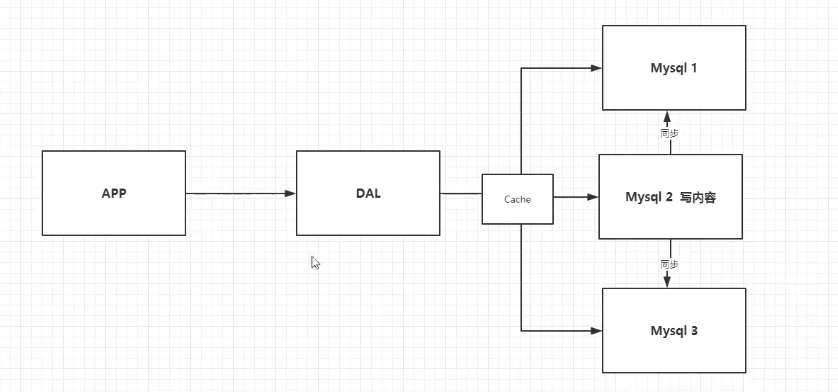
只要你开始出现以上的三种情况之一，那么你就必须要晋级!

1. **Memcached (缓存）+ MySQL＋垂直拆分（读写分离）**

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

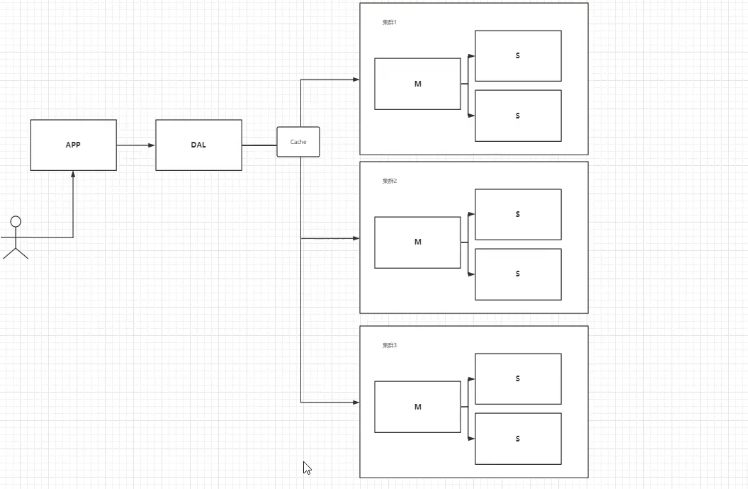
Mysql1和mysql2负责读取，mysql2负责写入，做到读写分离，然后cache是中间的缓存区域。

发展过程︰优化数据结构和索引-->文件缓存(IO ) ---> Memcached(当时最热门的技术! )



1. **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MylSAM:表锁，十分影响效率!高并发下就会出现严重的锁问题。

转战Innodb :行锁。

慢慢的就开始使用分库分表来解决写的压力! MySQL在哪个年代推出了表分区!这个并没有多少公司使用!MySQL的集群，很好满足哪个年代的所有需求!|

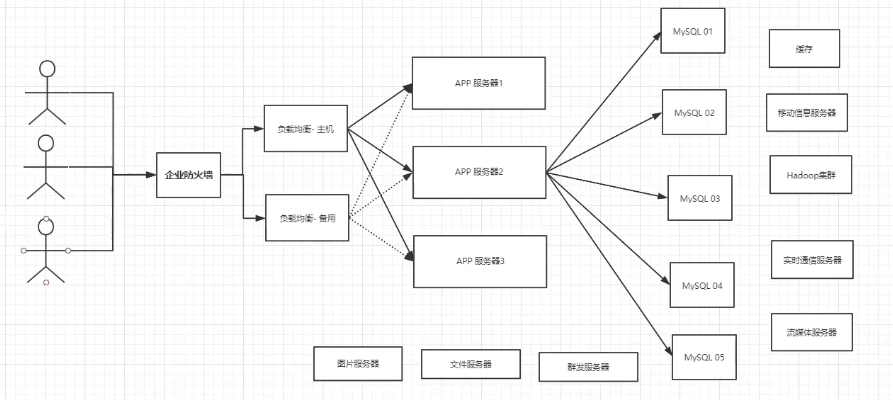
4、如今最近的年代

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)

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!

目前的一个基本互联网项目

**为什么要用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：并不是不是sql，而是Not Only SQL，不仅仅只是sql

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，No5QL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!

NoSQL特点：

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! )

4、传统RDBMS（关系型数据库）和NoSQL

**传统的RDBMS**

1.结构化组织

2.SQL

3.数据和关系都存在单独的表中

4.操作操作，数据定义语言

5.严格的一致性

6.基础事务

**NOSQL**

1.不仅仅是数据

2.没有固定的查询语言

3.键值对存储，列存储，文档存储，图形数据库（社交关系)-最终一致性，

4.CAP定理和BAS(异地多活)

5.高性能，高可用，高可扩

了解:3V+3高:

大数据时代的3V︰

主要是描述问题的

1.海量Volume

2.多样Variety

3.实时Velocity

大数据时代的3高︰

主要是对程序的要求

1.高并发

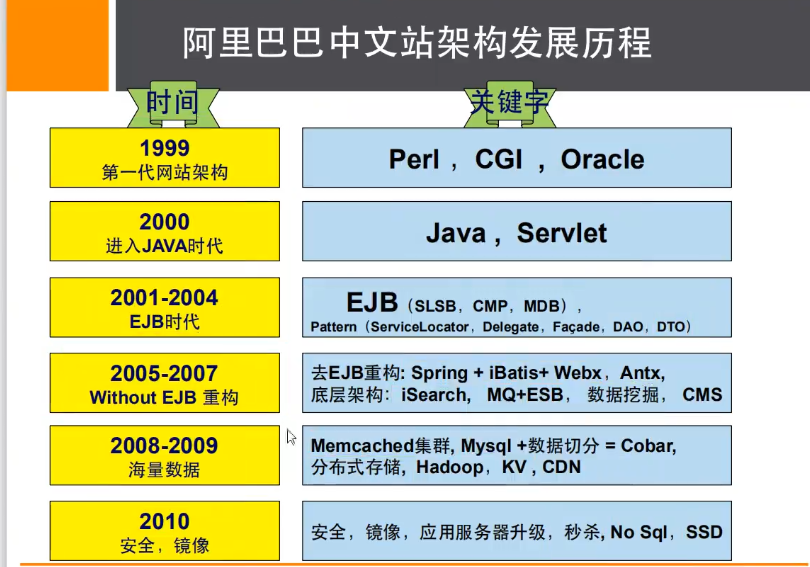
2.高可拓

3.高性能

真正在公司中的实践:NoSQL +RDBMS一起使用才是最强的，阿里巴巴的架构演进!

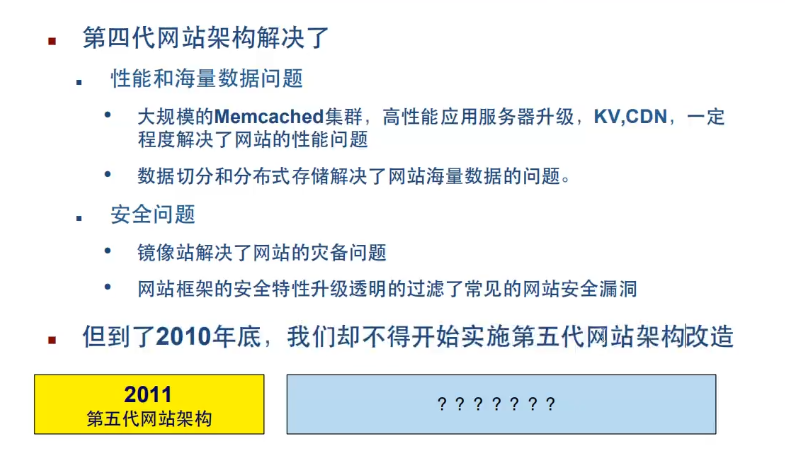
技术没有高低之分，就看你如何去使用!（提升内功，思维的提高!)

**阿里巴巴演进分析：**



技术急不得，越是慢慢学，才能越扎实!

第四代网站架构解决的问题：



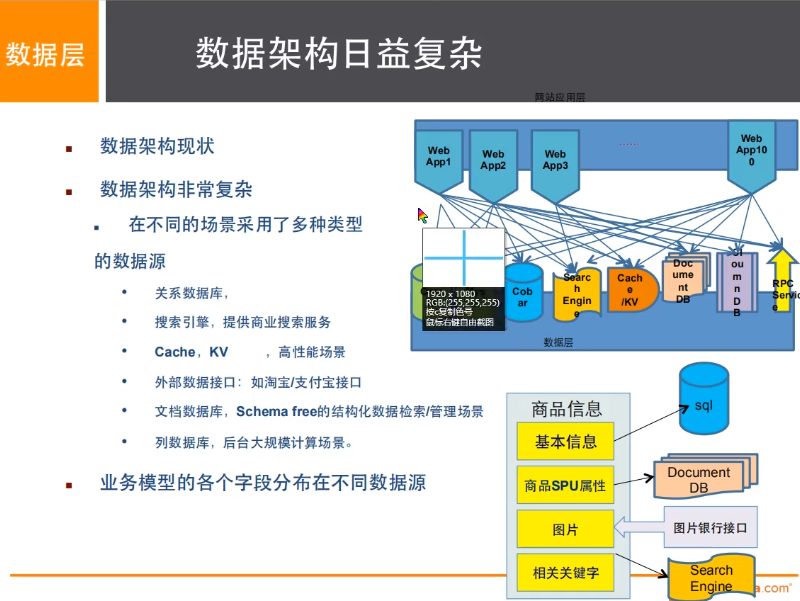
敏捷开发、极限编程：

开源才是技术的王道!

任何一家互联网的公司，都不可能只是简简单单让用户能用就好了!大量公司做的都是相同的业务;(竞品协议)

随着这样的竞争，业务是越来越完善，然后对于开发者的要求也是越来越高!高中，大一开始就应该是认真的学习了!





如果你未来相当一个架构师:没有什么是加一层解决不了的!

#1、商品的基本信息

名称、价格、商家信息;关系型数据库就可以解决了! MysQL / oracle（淘宝早年就去ToE了! -王坚:推荐文章:阿里云的这群疯子:40分钟重要!)

淘宝内部的 MysQL不是大家用的MySQL

#2、商品的描述、评论(文字比较多)

文档型数据库中，MongoDB

#3、图片

分布式文件系统FastDFS

淘宝自己的TFS

Gooale的 GFS

Hadoop HDFS

#4、商品的关键字(搜索)

搜索引擎solr elasticsearch

阿里用的Iserach:多隆

所有牛逼的人都有一段苦逼的岁月!但是你只要像SB一样的去坚持，终将牛逼!

#5、商品热门的波段信息

内存数据库

Redis Tair、Memache. . .

#6、商品的交易，外部的支付接口

三方应用

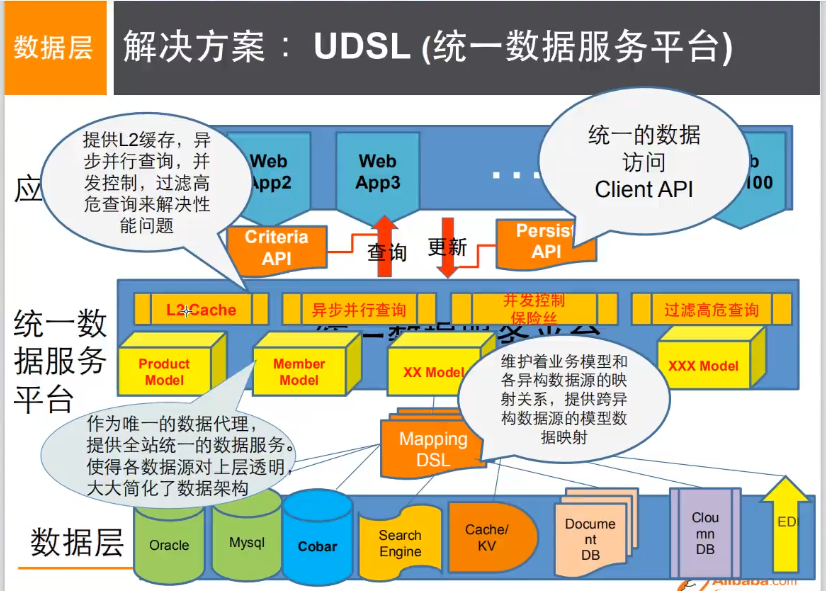
大型互联网应用问题:

·数据类型太多了!

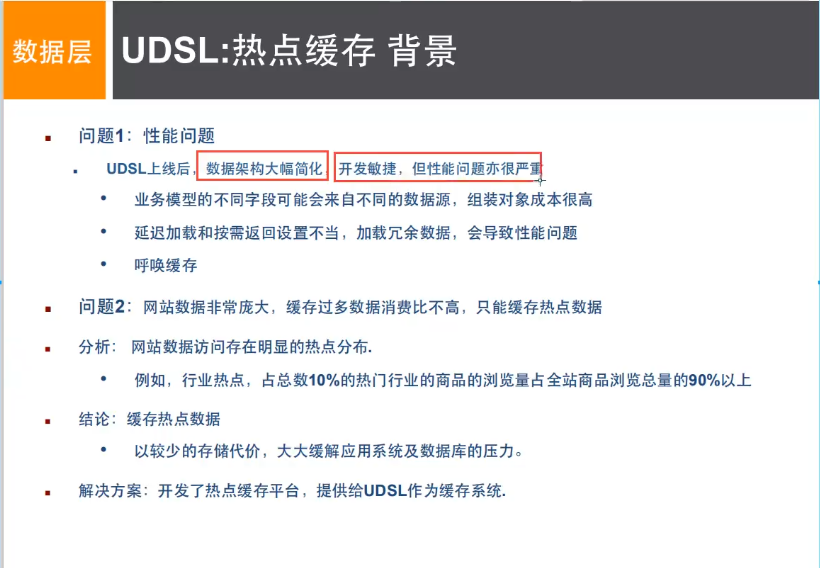
·数据源繁多，经常重构!

·数据要改造，大面积改造?

解决问题的方法:



热点缓存：



以上都是NoSQL的概述。

**NoSQL的四大分类**

KV键值对∶

·新浪:Redis（学）

·美团:Redis + Tair

·阿里、百度:Redis + memcache

**文档型数据库( bson格式和json一样）∶**

MongoDB(必须要掌握)

·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，最像关系型数据库的!

ConthDB

**列存储数据库**

·HBase（学）

·分布式文件系统

图形关系数据库

·他不是存图形的，放的是关系：比如朋友圈，广告推荐

·Neo4j（学），infoGrid；

NoSQL的四大分类示例图：四者对比



敬畏之心可以使人进步!宇宙!

活着的意义﹖追求幸福（帮助他人，感恩之心），探索未知（努力的学习，不要这个社会抛弃)

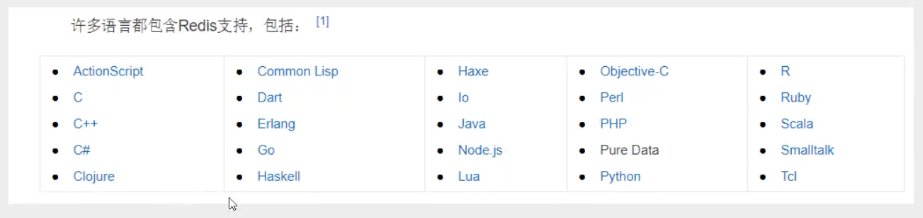
**Redis入门**

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-value数据库，并提供多种语言的API)

支持的语言：



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：

·多样数据类型

·持久化

·集群

·事务

学习中要用到的东西：

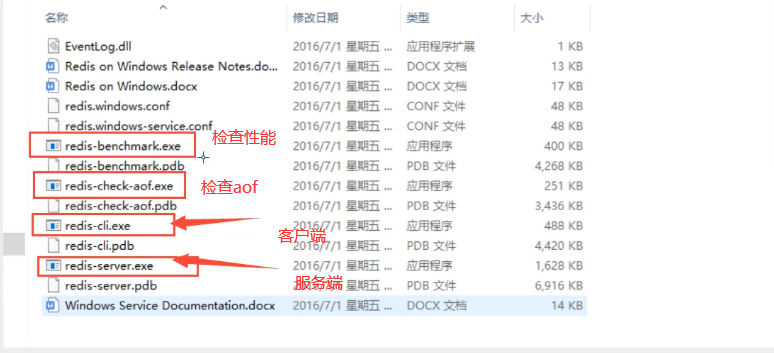
1. 狂神的公众号∶狂神说
2. 官网: [https://redis.io/](https://redis.iol)
3. 中文网: <http://www.redis.cn/>

注意:Wdinow在Github上下载(停更很久了! )

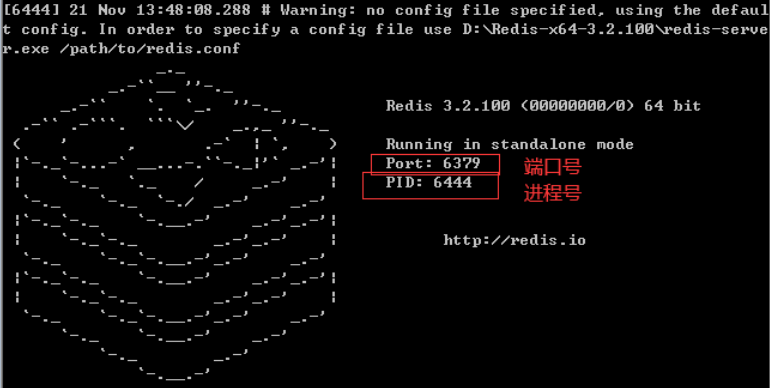
Redis推荐都是在Linux服务器上搭建的，我们是基于Linux学习!

**Windows安装：**

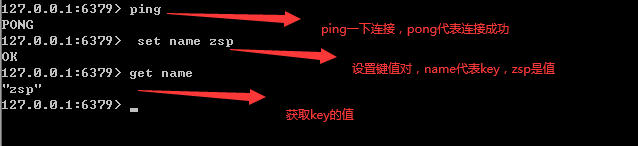
1. 下载安装包:https://github.com/dmajkic/redis
2. 下载完毕得到压缩包︰



1. 开启Redis，双击运行服务即可!点击redis-server



1. 连接redis，点击cli即可

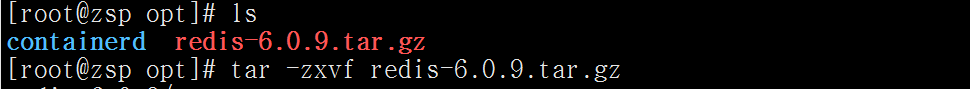


记住一句话，Window下使用确实简单，但是Redis推荐我们使用Linux去开发使用!

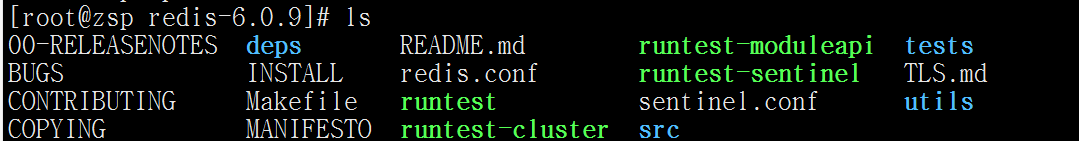
**Linux安装:**

1. 下载安装包! redis-5.0.8.tar.gz
2. 解压Redis的安装包!

使用tar -zxvf redis-6.0.9.tar.gz解压文件

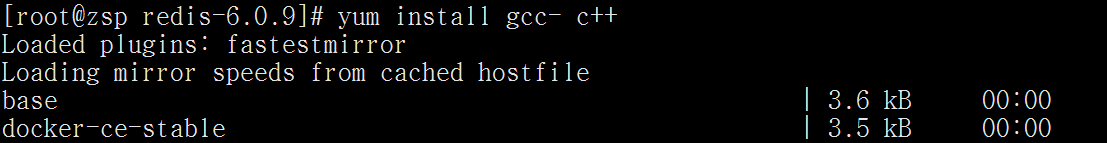


1. 进入解压后的文件，可以看到redis的配置文件



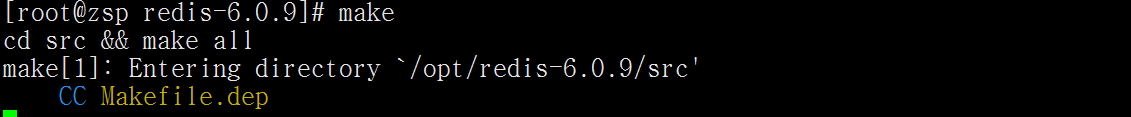
1. 配置redis运行所需环境

使用yum install gcc- c++安装c++环境

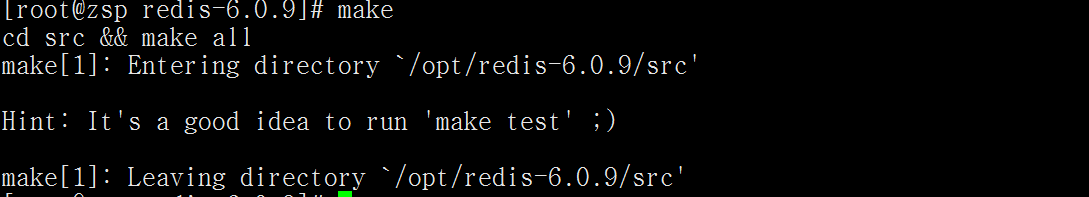


1. 执行安装redis，到redis解压文件夹内直接输入make（注意高版本的可能要升级gcc）

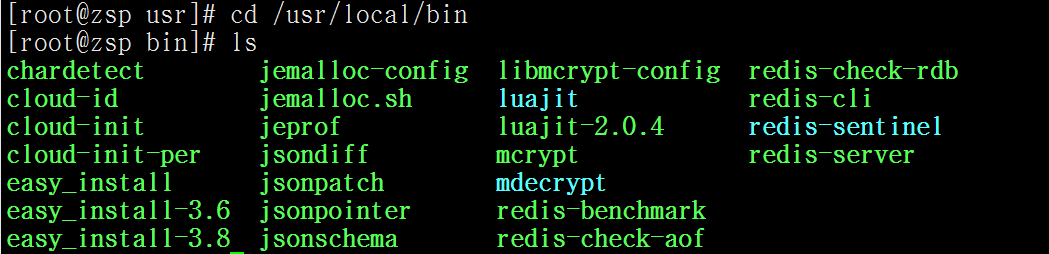
踩坑地址：<https://mp.csdn.net/console/home?spm=1001.2101.3001.4503>



再make一次，出现以下情况，说明完成：



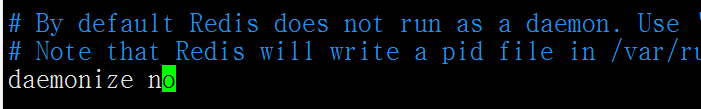
1. redis的默认安装路径/usr/local/bin （redis存放路径）



1. 将redis配置文件复制到我们当前目录下（之后我们会用这个配置文件启动



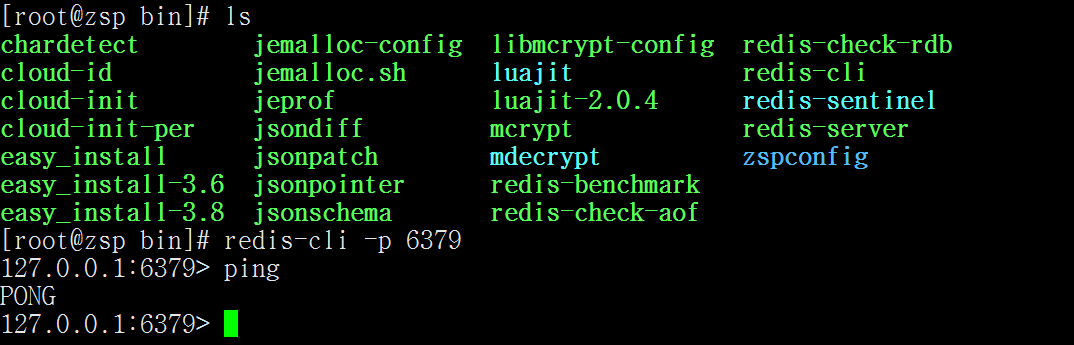
1. 修改redis的配置文件，设置后台启动（把daemonize状态改成yes



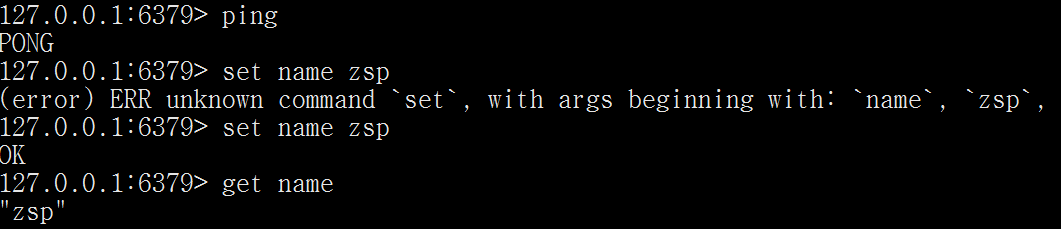
1. 启动redis服务（使用redis-server 路径 启动redis
   1. redis-server zspconfig/redis.conf



1. 使用redis-cli -p 端口号
   1. redis-cli -p 6379

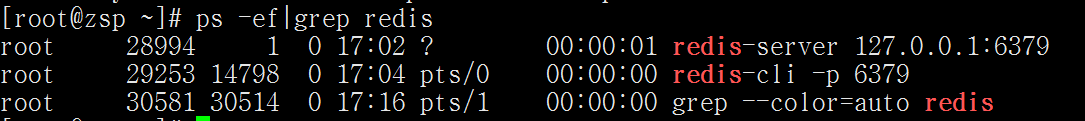


1. 测试一下

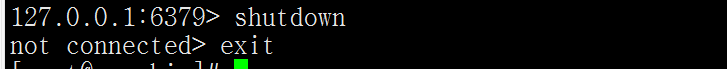


如果想查询所有的键值对：输入keys \*

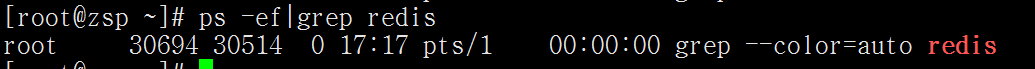
1. 查看redis的进程是否开启和关闭redis：
   1. ps -ef|grep redis



* 1. 输入shutdown即可关闭



查看后发现进程不存在了：



后面会使用单机多集群redis。

redis的存放路径：/usr/local/bin

redis配置的存放路径：/usr/local/bin/zspconfig

在docker上使用redis：

1. 拉取redis镜像

docker pull redis

1. 使用docker镜像创造容器
   1. 不使用配置文件创建

docker run --name redis01 -p 6379:6379 -d redis

* 1. 使用配置文件创建

·这里需要先去官方下载配置文件，详细参考博客

<https://blog.csdn.net/weixin_42456466/article/details/87270959>

·按着博客配置完配置文件后，接下来启动redis容器，以主机的6380启动redis

docker run -p 6380:6379 --name redis01 -v /home/redis/conf/redis.conf:/etc/redis/redis.conf -v /home/redis/data:/data -d redis redis-server /etc/redis/redis.conf --appendonly yes

关键词解释：

-p 6379:6379 端口映射：前表示主机部分，：后表示容器部分。

--name redis01 指定该容器名称，查看和进行操作都比较方便。

-v 挂载目录，规则与端口映射相同。

为什么需要挂载目录：个人认为docker是个沙箱隔离级别的容器，这个是它的特点及安全机制，不能随便访问外部（主机）资源目录，所以需要这个挂载目录机制。

-d redis 表示后台启动redis

redis-server /etc/redis/redis.conf  以配置文件启动redis，加载容器内的conf文件，最终找到的是挂载的目录/usr/local/docker/redis.conf

1. 使用redis容器或者进入docker容器
   1. 使用redis容器

docker exec -it redis01 redis-cli

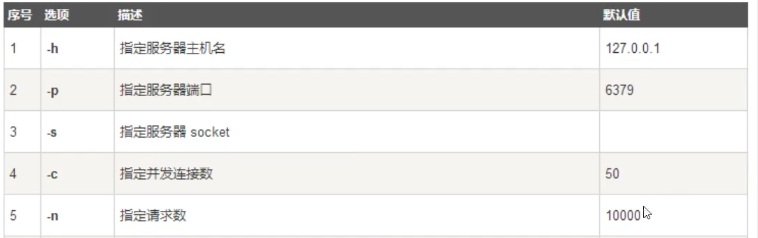
* 1. 进入redis容器

docker exec -it redis01 /bin/bash

**性能测试工具Redis-benchmark：**

简介：是一个压力测试工具

使用参数：



使用benchmark测试redis：对端口6379的redis进行100个并发发起1万个请求

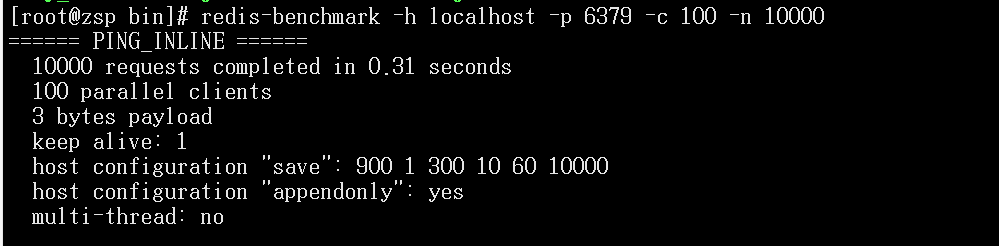
·redis-benchmark -h localhost -p 6379 -c 100 -n 10000

-h 指定服务器主机名

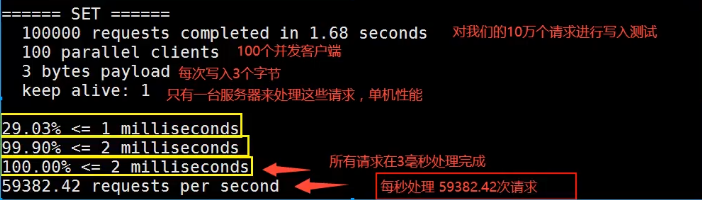
-p 指定服务端口

-c 指定并发数量

-n 指定每个并发发起请求



查看数据：



思考：为什么redis默认端口号是6379！

6379在是手机按键上MERZ对应的号码，而MERZ取自意大利歌女[Alessia Merz](http://it.wikipedia.org/wiki/Alessia_Merz)的名字。MERZ长期以来被antirez及其朋友当作愚蠢的代名词。[Redis](http://blog.nosqlfan.com/tags/redis" \o "查看 Redis 的全部文章)作者antirez同学在twitter上说将在下一篇博文中向大家解释为什么他选择[6379](http://blog.nosqlfan.com/tags/6379" \o "查看 6379 的全部文章)作为[默认](http://blog.nosqlfan.com/tags/%E9%BB%98%E8%AE%A4" \o "查看 默认 的全部文章)[端口](http://blog.nosqlfan.com/tags/%E7%AB%AF%E5%8F%A3" \o "查看 端口 的全部文章)号。而现在这篇博文出炉，在解释了Redis的LRU机制之后，向大家解释了采用6379作为默认端口的原因。

redis在redis6以前还是单线程的，到6以后就有多线程的了，官方解释是：

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，Redis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!

Redis为什么单线程还这么快?

1、误区1∶高性能的服务器一定是多线程的?

2、误区2∶多线程一定比单线程效率高!（如果cpu只有单核，CPU上下文会切换!会导致运行变慢)

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!

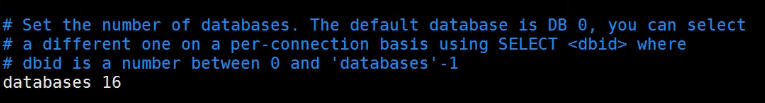
**Redis的基本命令：**

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 )。

**基础的知识：**

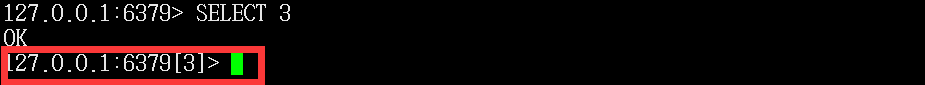
1. redis默认有16个数据库



默认使用的是第0个，可以选择切换数据库

2、redis基本操作指令

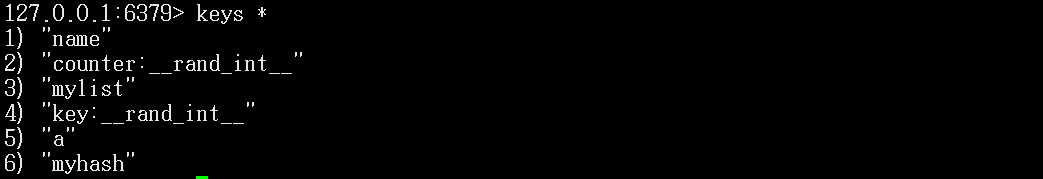
1. 切换数据库指令select



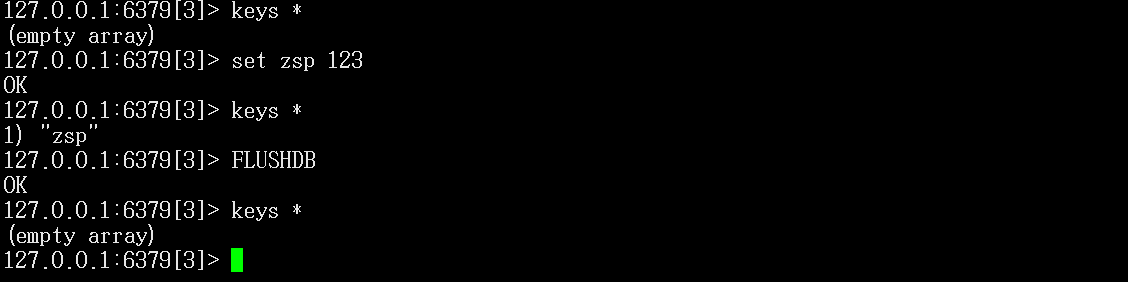
1. 查看内存大小DBSIZE



1. 查看所有的key，keys \*



1. 清除当前数据库FLUSHDB（小心用）



1. 清空所有数据库FLUSHALL

**Redis-key：**

1. key是否存在（查找当前库）

·EXISTS key



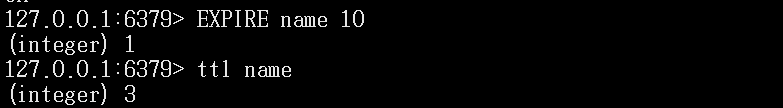
1. 移动当前库的key到其他库（当前库的key

·move key 1 （把key移动到库1

1. 设置key-v值过期（过期后是直接删除的

·EXPIRE key 10（设置多少秒后，对应的key-v过期

·ttl key （查看对应的key剩余过期时间



1. 查看key对应的v的数据类型

·type key

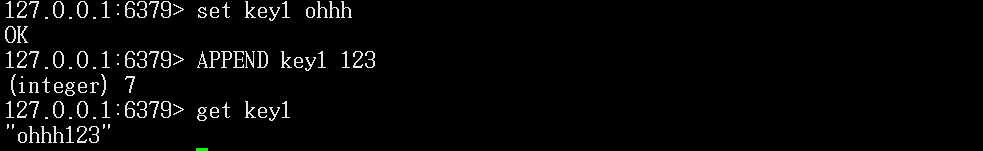


后面如果遇到不会的命令，可以在官网查看帮助文档!

**String（字符串）类型：**

1. 追加字符串

·APPEND key1 123（key1是要追加进去的键，123是追加值）



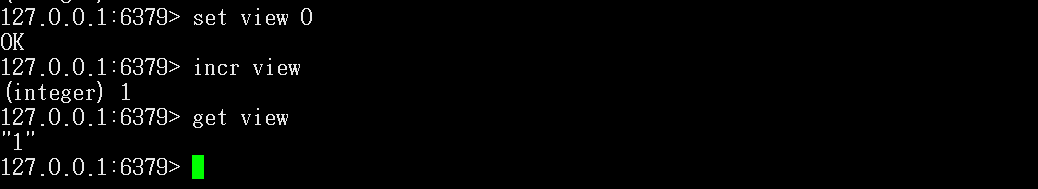
1. 查看key对应的字符串长度

·STRLEN key



1. 数量增加

·incr view（i++）

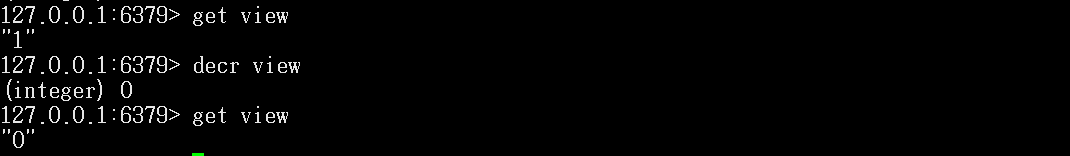


·incrby view 10（增加10



1. 数量减少

·decr view（i--）

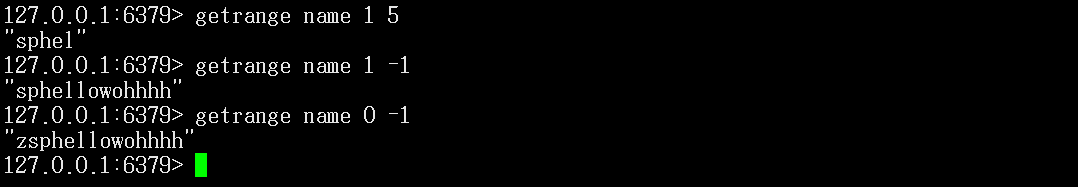


·decrby view 10（数量减少10）



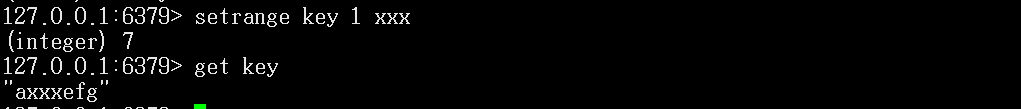
1. 获取字符串指定范围

·getrange key 1 5（-1 表示到完）



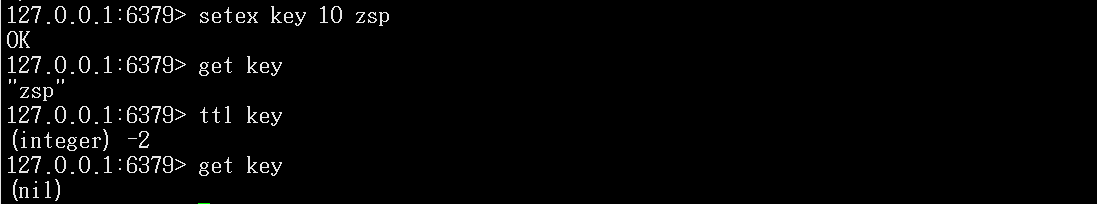
1. 对字符串指定范围进行替换

·setrange key 1 xxx



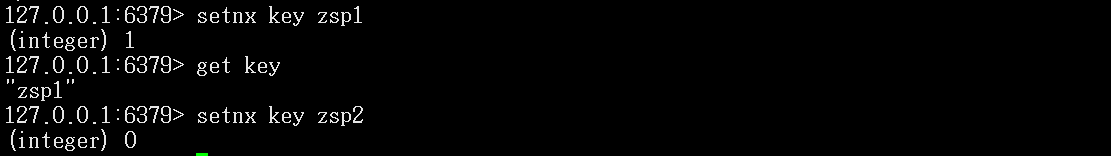
1. 设置过期时间

·setex key 10 zsp (set with expire) （设置过期时间，这个方法会覆盖原本的值



1. **键值对**不存在才创建**键值对**

·setnx key zsp (set if not exist，只有不存在键值对才创建，创建成功返回1，失败返回0

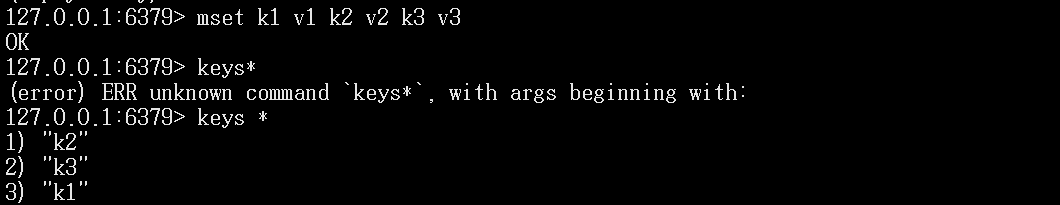


ex:设置过期时间

nx:存在则不创建

1. 批量处理

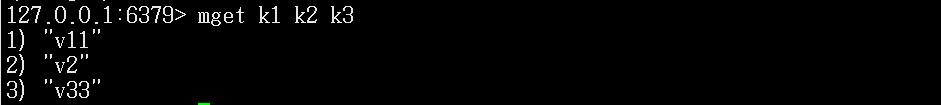
·mset k1 v1 k2 v2 k3 v3 (批量设置存储,如果key已经存在，会覆盖掉key的value值



·msetnx k1 v1 k2 v2(批量设置存储，如果key存在则失败



·mget k1 k2 k3 (批量获取key的值



注：msetnx是一个原子性的操作，要么一起成功要么一起失败~

10、存储对象

·使用json

存储：set user:1 {name:zhangsan,age:3}

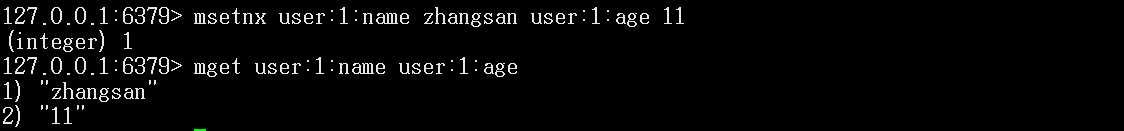
取值：get user:1



·自定义存储

存储：msetnx user:1:name zhangsan user:1:age 11

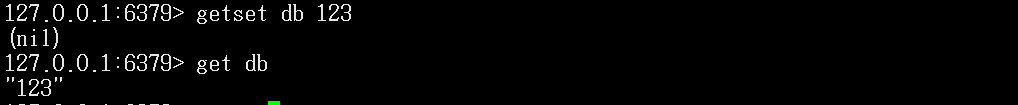
获取：mget user:1:name user:1:age



1. 先获取然后再修改值

（可以用于原子性操作，比如和原本的值对比，如果相同进行交换，类似于乐观锁的version

·getset（注意！这个操作会覆盖掉原本的值



数据结构是相同的!

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

·统计多单位的数量

·粉丝数

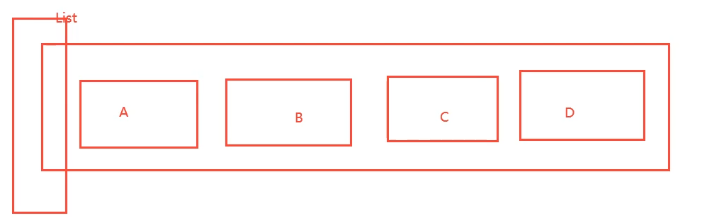
·对象缓存存储

**List：**

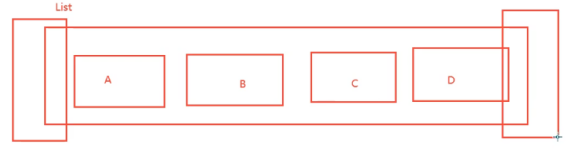
list充当队列：从左边进右边出



List充当栈：先进的后出只有一个出口



List充当阻塞队列：从两边取



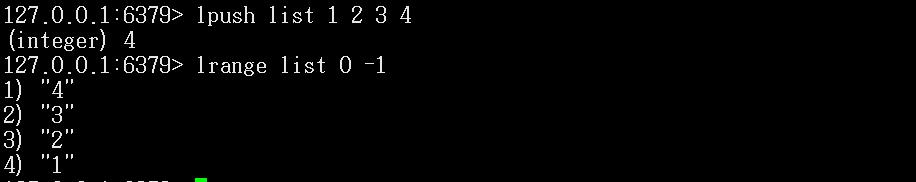
在redis里面，我们可以把list玩成，栈、队列、阻塞队列!

所有的list命令都是用开头的。

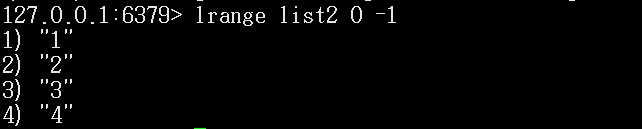
**List的命令：**

1. 存储list命令

·lpush list 1 2 3 4（将列表的值放到头部，先放进去的会在最后（左边）



·rpush list 5 （将列表的值放到尾部，先放进去的在前面（右边）

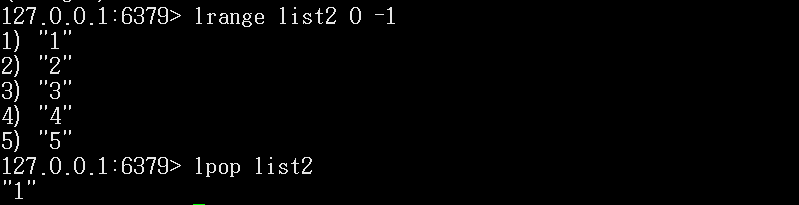


1. 获取list的范围值

·lrange list 0 -1 （获取指定范围的list的值，-1表示最大值

1. 移除当前队列的值

·lpop list2（移除左边队列的第一个值



·rpop list2



1. 获取下标值

·lindex list2 1（起始下标是0）



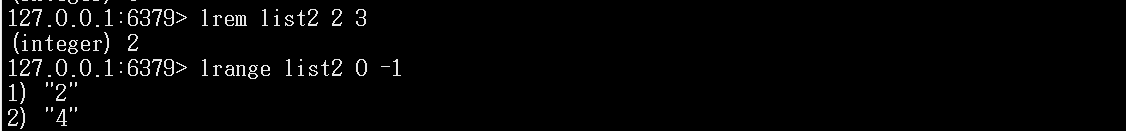
1. 获取列表长度

·llen list2



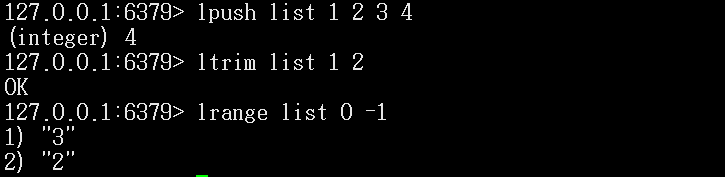
1. 移除指定的值

·lrem list2 2 3（从list2里面移除2个为3的值



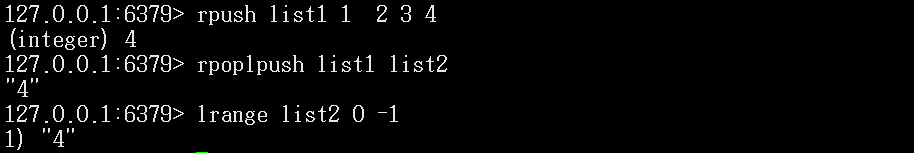
1. 截取功能

·ltrim list 1 2



1. 组合命令

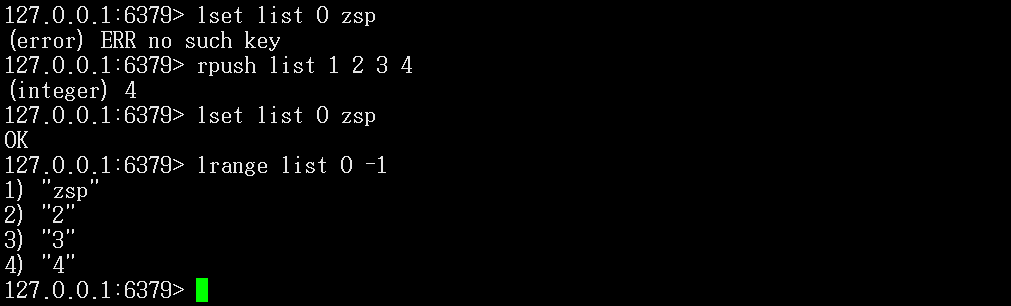
·rpoplpush list1 list2（从list1中弹出元素加入到list2



1. 替换指定下标的值

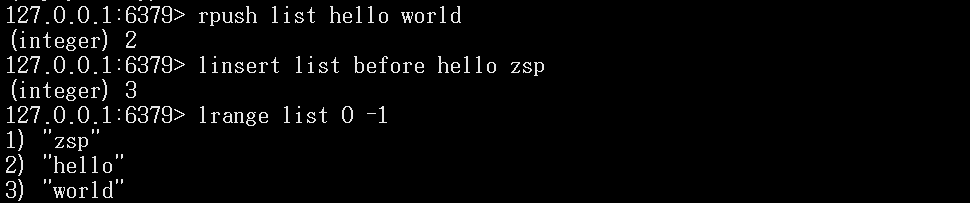
·lset list 0 zsp

注意如果指定的列表或下标不存在，则无法使用



1. 在指定的元素前或者后添加数值

·linsert list before hello zsp



小结：

·他实际上是一个链表，before Node after , left , right 都可以插入值

·如果key不存在，创建新的链表

·如果key存在，新增内容

·如果移除了所有值，空链表，也代表不存在!

·在两边插入或者改动值，效率最高!中间元素，相对来说效率会低一点~

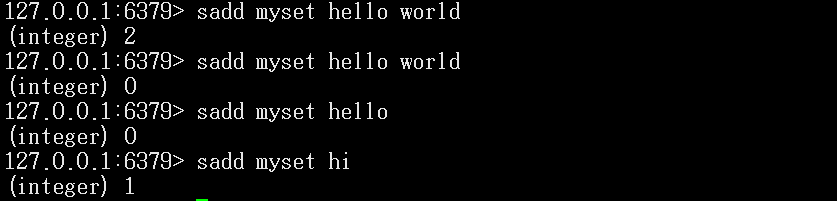
可以实现消息排队!消息队列( Lpush Rpop ) ，栈(Lpush Lpop ) !

**Set（集合，这里用S开头代表set）：**

Set是一个无序不重复集合

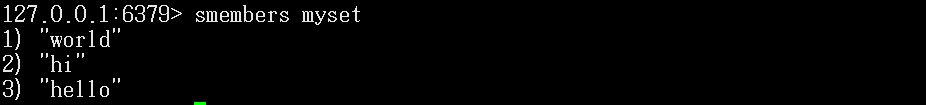
1. 创建集合，给集合添加元素

·sadd myset hello world（添加元素，如果元素已存在则添加失败



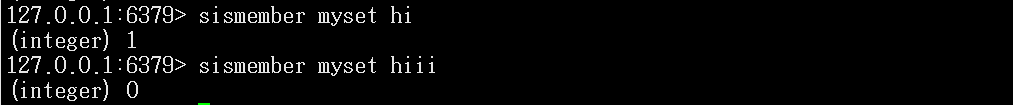
1. 查看set里面的所有值

·smembers myset



3、判断元素是否在集合里

·sismember myset hi （set is member ，1表示存在，0表示不存在



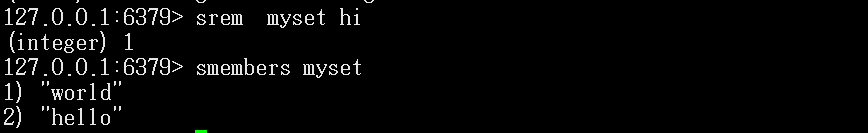
4、获取集合里面的个数

·scard myset



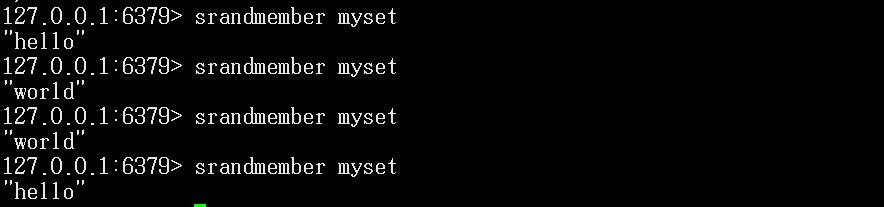
1. 移除set集合中的一个具体的值

·srem myset hi



1. 获取集合里随机一个元素

·Srandmember myset



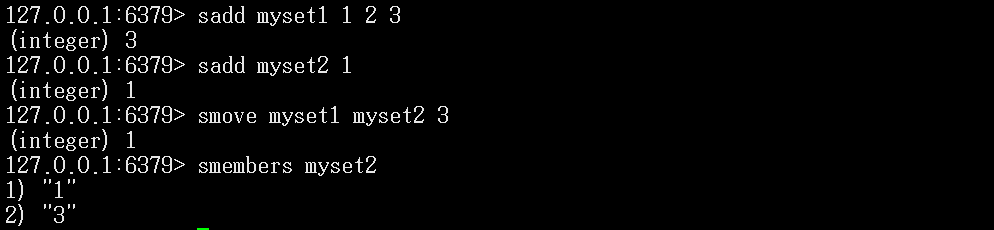
1. 随机删除一个元素

·spop myset



1. 从一个集合里面移动一个元素到另一个集合里

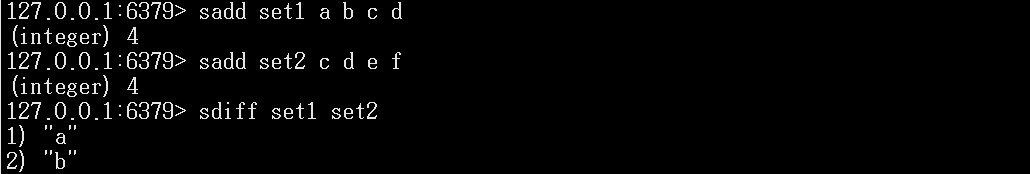
·smove myset1 myset2 3



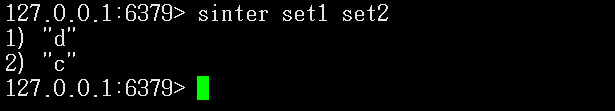
1. 集合运算

设置set1 set2两个集合

·sdiff set1 set2（差集



·sinter set1 set2（交集



·sunion set1 set2



微博，A用户将所有关注的人放在一个set集合中!将它的粉丝也放在一个集合中!

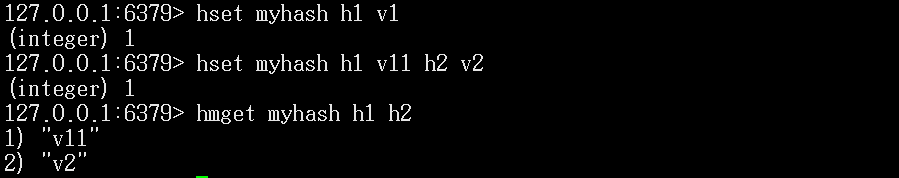
共同关注，共同爱好，二度好友，推荐好友!(六度分割理论）

**Hash（哈希，h开头）**

Map集合，key-map！时候这个值是一个map集合：本质和string类型没有太大区别

1. 创建哈希表或存入哈希值

·hset myhash h1 v11 h2 v2 （可以存入一个，也可以存入多个



·hmset h1 v1 h2 v2 h3 v3 （一次存入多个值



·hsetnx myhash h1 v111（不存在则键入值，存在则不键入

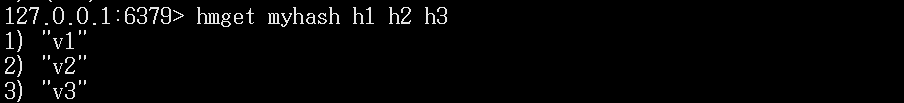


1. 获取哈希对应的值

· hget myhash h1（只能获取一个值



·hmget myhash h1 h2 h3（批量获取哈希表的值

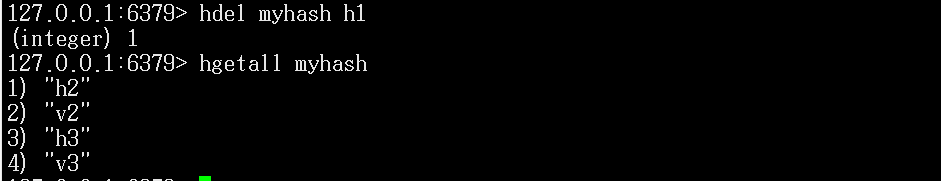


·获取所有的哈希值，一个对应一个hash值



1. 删除哈希指定的key字段

·hdel myhash h1（可以批量删除



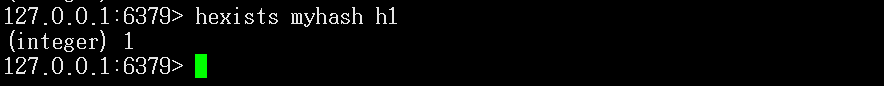
1. 查看哈希表长度

·hlen myhash



1. 判断哈希表是否存在指定字段

·hexists myhash h1



1. 增加值或者减去值

·hincrby myhash h3 1 （1是增量）

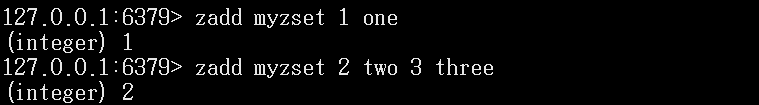


hash变更的数据user name age,尤其是是用户信息之类的，经常变动的信息! hash更适合于对象的存储，String更加适合字符串存储!

**Zset(有序集合，代表符号为Z)**

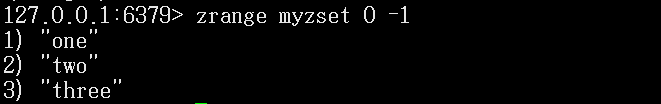
在set的基础上增加了一个值，原本是set k1 v1 后面变成zset k1 score1 v1  
1、添加或创建zset集合

·zadd myzset 1 one（中间的数字称之为score，即 zadd myzset score one



2、展示范围内的zset集合

·zrange myzset 0 -1

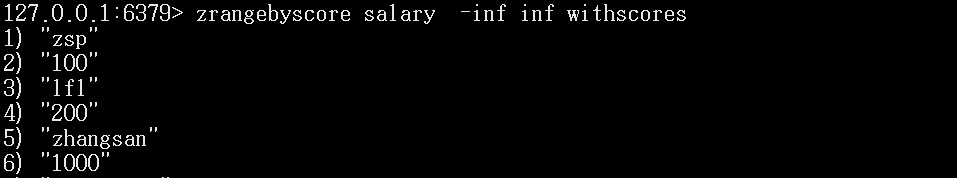


3、通过score升序排序(语法是最小值到最大值

·zrangebyscore salary -inf inf（inf代表无穷的意思



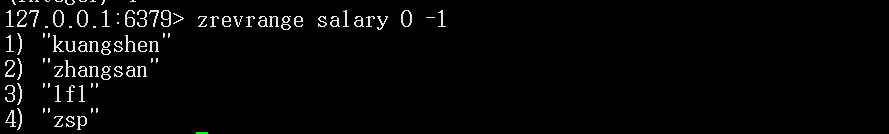
·zrangebyscore salary -inf inf withscores（可以自定义的加上参数



·zrangebyscore salary 0 (2500 (小于等于可以用（表示

4、通过score进行降序排序

·zrevrange salary 0 -1



4、移除zset里面的元素

·zrem myzset two



5、查看有多少元素

·zcard salary（查看所有的元素



·zcount salary 100 300（这里的排序不是下标而是score



可以用zset实现的功能：

set 排序存储班级成绩表，工资表排序!

普通消息，1，重要消息2，带权重进行判断!

排行榜应用实现，取Top N测试!

**三种特殊类型**

**geospatial地理位置：（代表字符geo）**

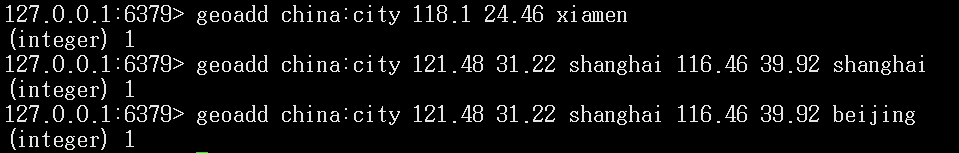
朋友的定位，附近的人，打车距离计算?

Redis的Geo在redis3.2版本就有了！这个功能可以推算地理位置的信息，两地之间的距离。

1. 创建或添加地理位置

·geoadd china:city 118.1 24.46 xiamen

·参数（key 经度，纬度，名称

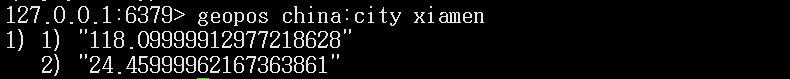
#有效的经度从-180度到180度。

#有效的纬度从-85.05112878度到85.05112878度。

#坐标位置超出上述指定范围时，该命令将会返回一个错误。

1. 获取城市对应的经纬度

·geopos china:city xiamen（输入key和你的城市名）



1. 计算两地的距离

·geodist china:city xiamen beijing km（后面能加上单位例如 km



4、以经纬度为中心用距离获取元素

附近的人?（获得所有附近的人的地址，定位!)通过半径来查询!

·georadius china:city 110 37 1000 km （可以加上withdist显示直线距离

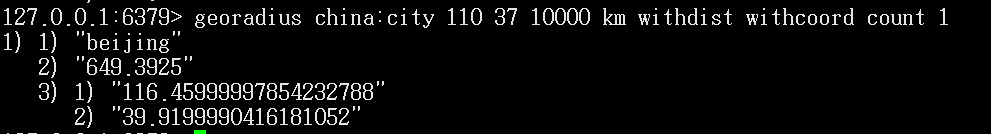


·加上参数withdist 可以显示直线距离

·加上参数withcoord可以带上找到的经纬度

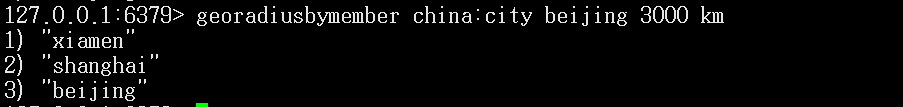
·加上参数count 1 可以只找到一个

·georadius china:city 110 37 10000 km withdist withcoord count 1 （全部参数带上



5、以城市为中心用距离获取元素

·georadiusbymember china:city beijing 3000 km

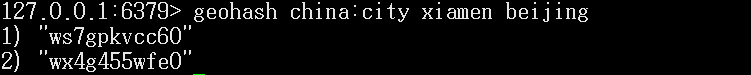


#找出位于指定元素周围的其他元素!

6、返回一个或者多个位置元素的哈希值

该命令将返回11个代表地理位置的Geohash字符串!

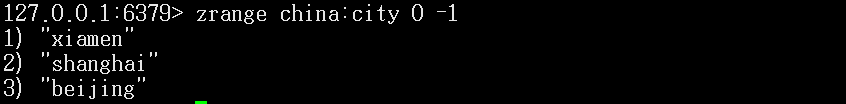
·geohash china:city xiamen beijing



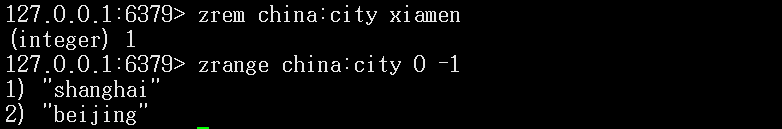
#将二维的经纬度转换为一维的字符串

**GEO底层原理是Zset！我们可以用Zset命令来操作geo！**

·zrange china:city 0 -1（查询geo里面的所有元素



·zrem china:city xiamen（移除元素



**Hyperloglog：（代表符号是pf）**

什么是基数？

A{1,3,5,7,9,7}

B{1,3,5,7,9}

基数（不重复的元素）=5，可以接受误差！

Redis 2.8.9版本就更新了Hyperloglog 数据结构!Redis Hyperloglog基数统计的算法!

优点∶占用的内存是固定，2^64不同的元素的计数，只需要废12KB内存!如果要从内存角度来比较的话Hyperloglog 首选!网页的UV(一个人访问一个网站多次，但是还是算作一个人! )

传统的方式，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!

这个方式如果保存大量的用户id，就会比较麻烦!我们的目的是为了计数，而不是保存用户id ;0.81%错误率!统计UV任务，可以忽略不计的!

基本命令：

1. 创建或者添加

·pfadd mypf a b c d e f g



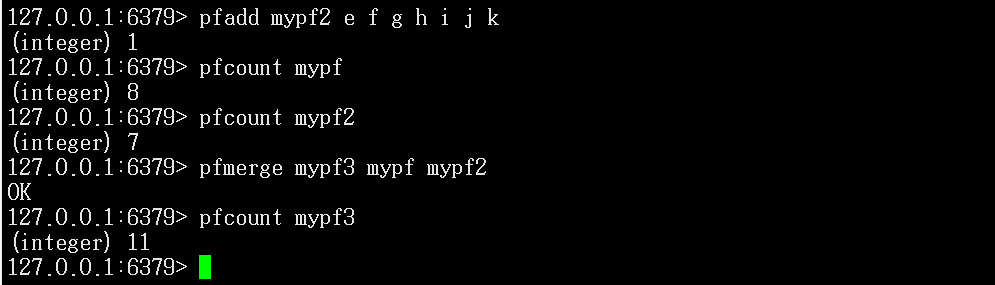
1. 查看基数

·pfcount mypf



1. 合并基数

·pfmerge mypf3 mypf mypf2



如果允许容错，那么一定可以使用Hyperloglog !

如果不允许容错，就使用set 或者自己的数据类型即可!

#使用场景主要在于统计用户数量，或者访问数这种！

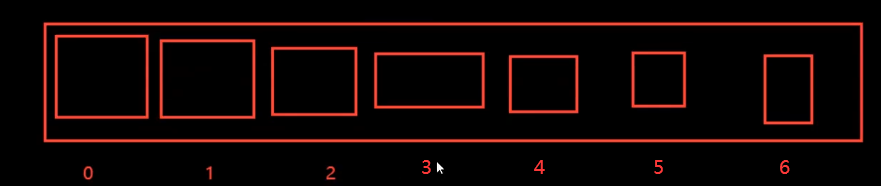
**Bitmaps**

位存储：

统计用户信息，活跃，不活跃!登录、未登录!打卡，365打卡!两个状态的，都可以使用Bitmaps !Bitmaps拉图，数据结构!都是操作二进制位来进行记录，就只有0和1两个状态!

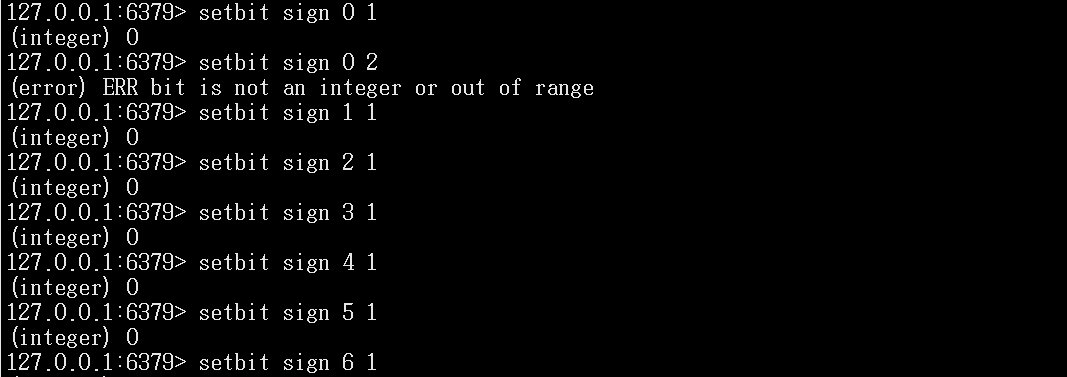
365天=365 bit1字节= 8bit46个字节左右!

示例图：表示一周七天



操作命令：

1. 设置一周七天签到，从0到6代表一周七天

·setbit sign 0 1 （前一个数字代表星期几，后一个数字代表状态，后一位数字只有0和1  


1. 查看某一天是否打卡

·getbit sign 

1. 统计打卡天数

· bitcount sign



·后面可以追加参数，例如

**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，在事务执行过程的中，会按照顺序执行!一次性、顺序性、排他性!执行一些列的命令!

·Redis事务没有没有隔离级别的概念!

·所有的命令在事务中，并没有直接被执行!只有发起执行命令（Exec）的时候才会执行!

·Redis单条命令式保证原子性的，但是事务不保证原子性!

Redis的执行事务的过程：

1、开启事务（multi）

2、命令入队（要使用的命令）

3、执行事务（exec）

正常执行事务操作：

1. 开启事务（multi）

·multi

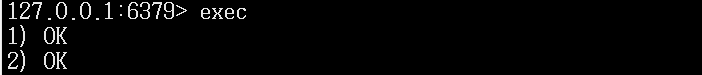


1. 使用命令（此时命令会进入队列）



1. 执行事务（exec）

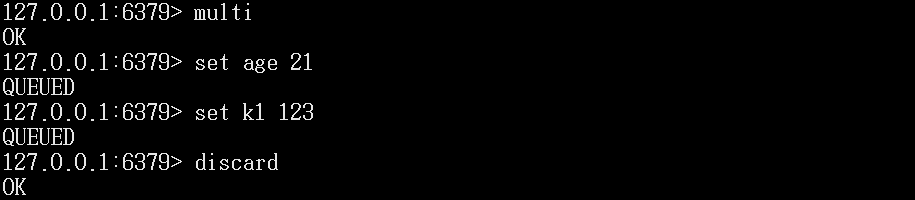
·exec



放弃事务：

1. 使用discard放弃事务即可

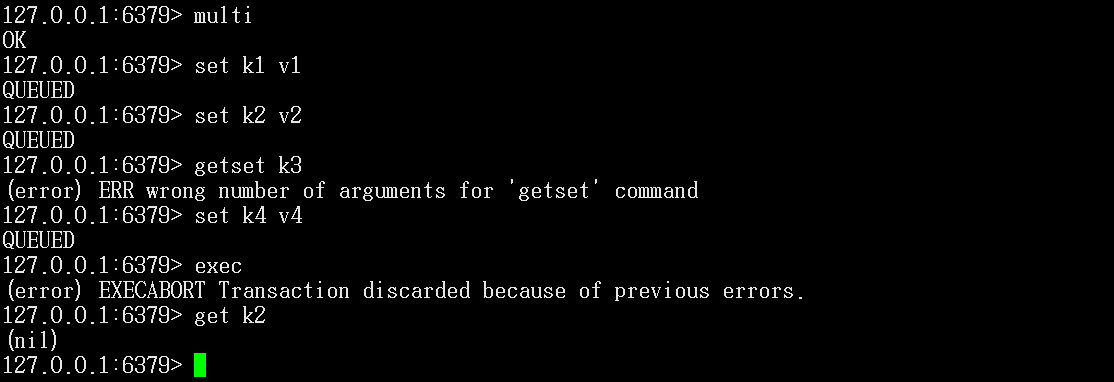
·discard



放弃事务的话，所有命令不会被执行。

编译型异常（代码有问题！命令有错），事务中所有的命令都不会被执行：

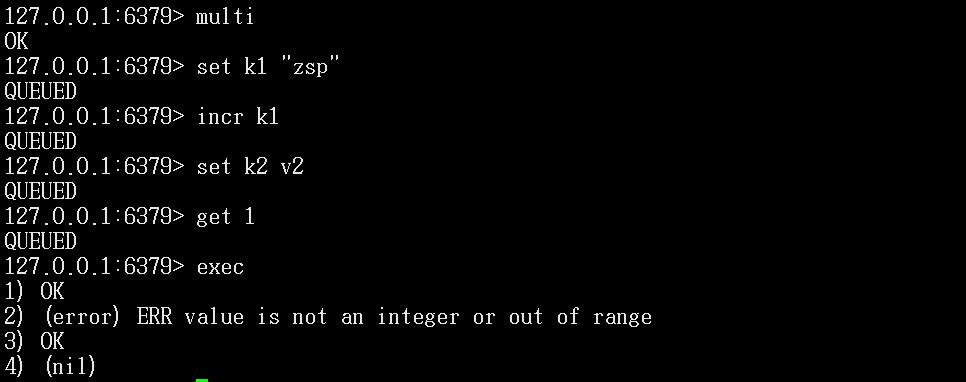
·可以发现整条代码都没有执行



运行时异常（1/0），如果事务队列中存在语法性，那么执行命令的时候，其他命令是可以正常执行的（这里诠释了为什么redis的事务没有原子性）

（举例：A转账给B,B发生了错误没收到钱，A却少了钱）

·会发现，虽然2出现了错误，但是其他的操作还是能进行



**监控! Watch**

·使用unwatch可以解锁

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：

·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

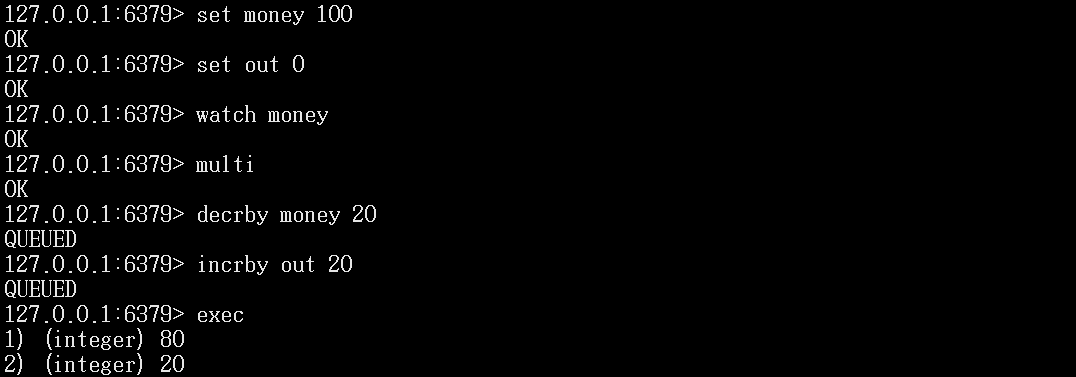
·获取version

·更新的时候比较version

Redis监视测试：这个监视只持续一次操作。

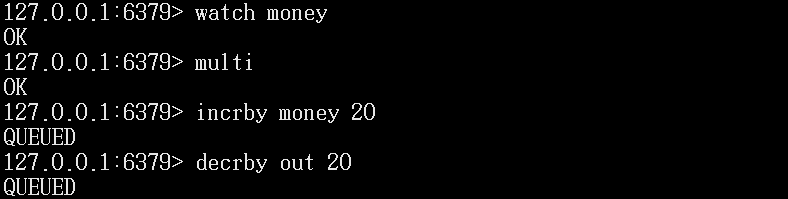
1、正常情况下

·watch money



2、测试多线程修改值，使用watch可以充当redis的乐观锁操作

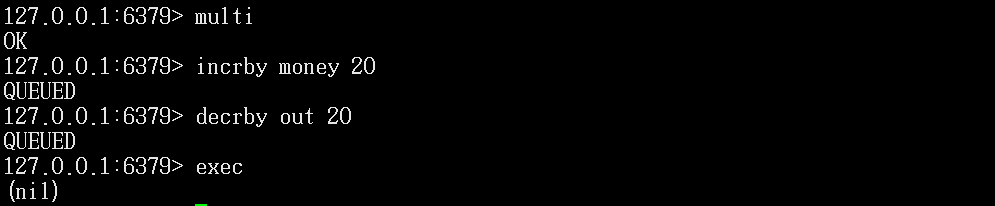
·首先客户端1使用watch监视操作，但并没有提交



·客户端2对监视的对象进行修改



·客户端1的提交失败，说明监视类似于乐观锁



#在一次事务结束完毕后就会自动解锁watch

#发生了discard也会解锁

#使用unwatch也可以解锁

**Jedis**

我们要使用Java来操作Redis，知其然并知其所以然，授人以渔!学习不能急躁，慢慢来会很快!

什么是jedis ？是Redis官方推荐的java连接开发工具!使用Java操作Redis 中间件!如果你要使用java操作redis，那么一定要对Jedis十分的熟悉!

测试：

1. 导入对应的包

<dependency>  
 <groupId>redis.clients</groupId>  
 <artifactId>jedis</artifactId>  
 <version>3.3.0</version>  
</dependency>  
<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.60</version>  
</dependency>

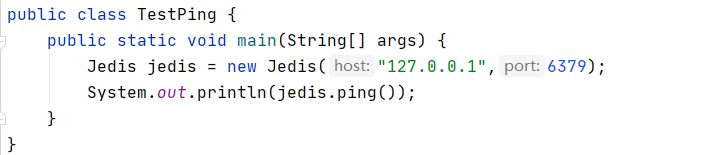
1. 编码测试

·连接数据库

Jedis jedis = new Jedis("127.0.0.1",6379);

·操作命令

jedis.ping()



·断开连接

**常用的API（跟前面学的一样）**

string

List

Set

Hash

Zset

**事务**

1. 先连接上我们的redis

Jedis jedis = new Jedis("127.0.0.1", 6379);

1. 编写json

JSONObject jsonObject = new JSONObject();  
jsonObject.put("name","zsp");  
jsonObject.put("age","21");

1. 开启事务

Transaction multi = jedis.multi();

1. 使用事务

multi.set("user1",result);  
multi.set("user2",result);  
multi.exec();

1. 最后关闭

jedis.close();

全部代码展示：

package com.zsp;  
  
import com.alibaba.fastjson.JSONObject;  
import redis.clients.jedis.Jedis;  
import redis.clients.jedis.Transaction;  
  
public class TestMulti {  
 public static void main(String[] args) {  
 Jedis jedis = new Jedis("127.0.0.1", 6379);  
 JSONObject jsonObject = new JSONObject();  
 jsonObject.put("name","zsp");  
 jsonObject.put("age","21");  
 Transaction multi = jedis.multi();  
 String result = jsonObject.toJSONString();  
 try {  
 multi.set("user1",result);  
 multi.set("user2",result);  
 multi.exec();  
 } catch (Exception e) {  
 multi.discard();  
 } finally {  
 System.out.println(jedis.get("user1"));  
 System.out.println(jedis.get("user2"));  
 jedis.close();  
 }  
 }  
}

**springboot整合redis：**

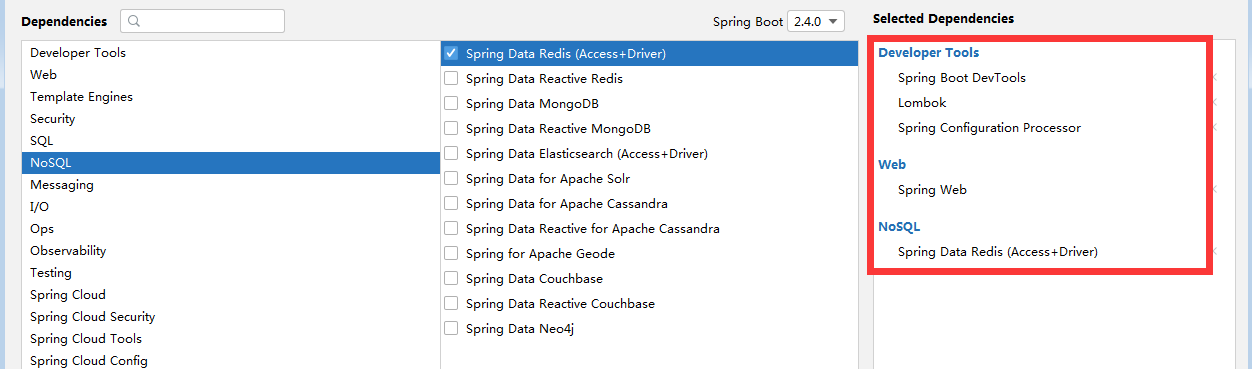
SpringBoot操作数据:spring-data jpa jdbc mongodb redis !SpringData也是和SpringBoot齐名的项目!

说明︰在SpringBoot2.x之后，原来使用的jedis被替换为了lettuce?

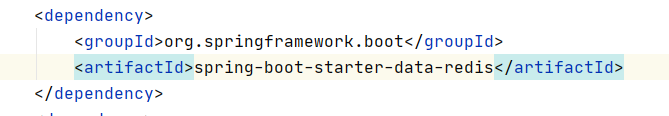
jedis :采用的直连，多个线程操作的话，是不安全的，如果想要避免不安全的，使用jedis pool连接池!更像BIO模式

lettuce :采用netty，实例可以再多个线程中进行共享，不存在线程不安全的情况!可以减少线程数据了，更像NIO模式

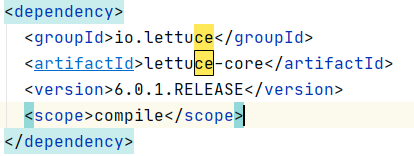
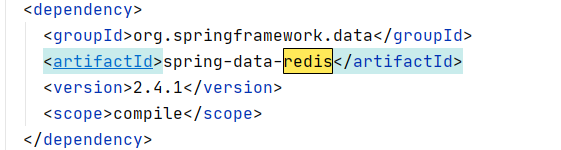
1. 创建boot项目导入依赖：



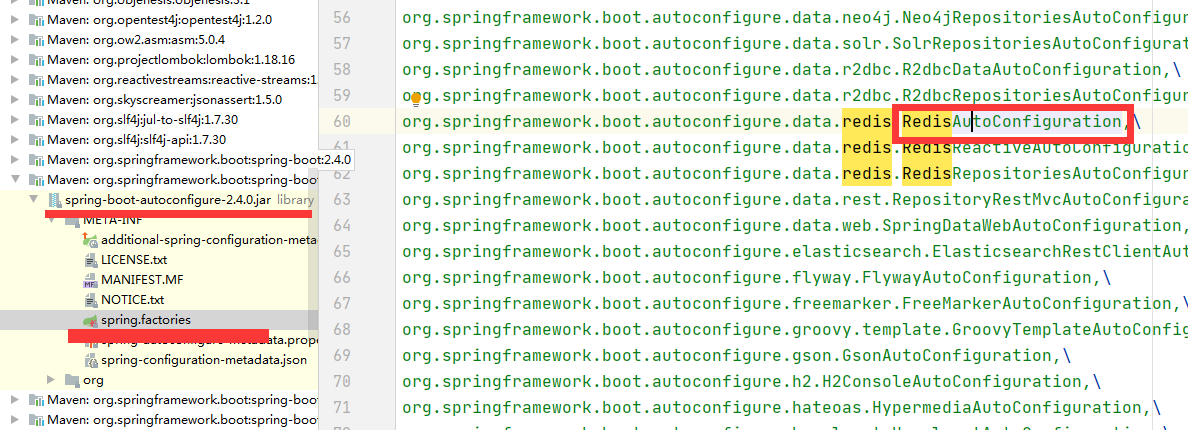
·查看依赖



·进入底层后发现还是引用的spring-data-redis和lettuce



1. 进入源码查看怎么书写配置



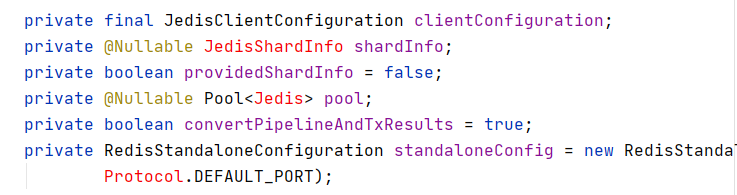


进入后查看可以配置的属性。

1. 根据配置类书写配置文件



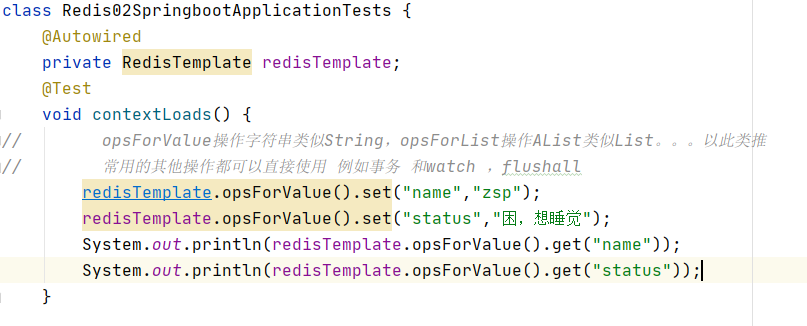
注意：连接池记得配lettuce的，因为阅读源码后发现jedis根本没有导入



1. 测试：

opsForValue操作字符串类似String，opsForList操作AList类似List。。。以此类推

常用的其他操作都可以直接使用 例如事务 和watch ，flushall



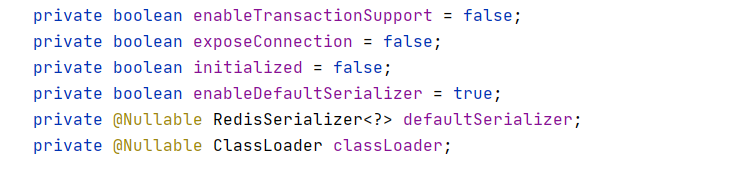
**Redis序列化**

为什么要序列化？

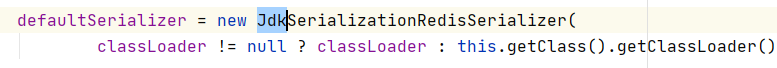
·因为序列化后可以存储对象，不序列化的话，保存对象会报错

·在企业中，pojo类都会序列化

进入RedisTemplate，查看序列化



使用的是默认的jdk序列化方式：

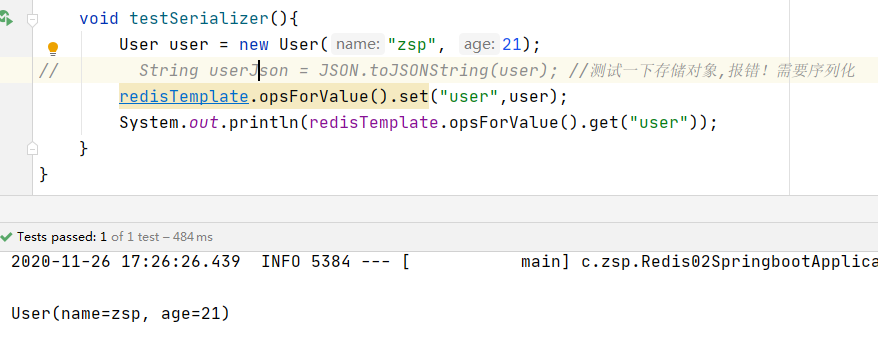


自己实现序列化：

·没有序列化：传入对象会报错



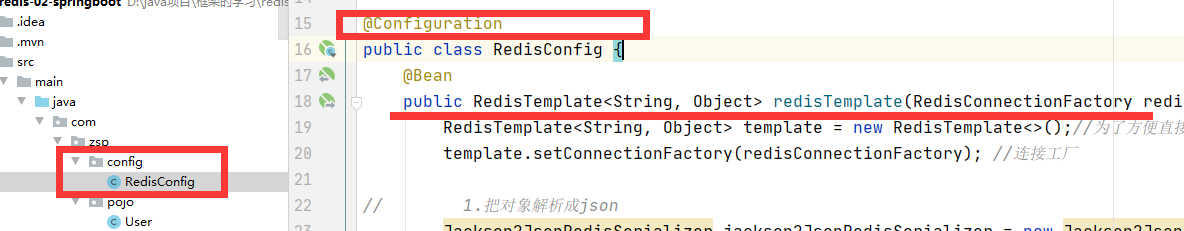
·给pojo类实现 序列化(Serializable)后，发现成功：



在企业中，pojo类都会序列化

·编写一个自己的redisTemplate，实现序列化

1、创建一个redis的配置类，从源码中获取redisTemplate的方法，重写他（存在配置文件了



2、使用它



·发现是可以的，再去查看一下redis的保存方式。



是用键值对形式保存下来的。

Redis工具类的使用：

1. 导入redis工具类（工具类我写到配置文件里了）



2、使用它

所有的redis操作，其实对于java开发人员来说，十分的简单，更重要是要去理解redis的思想和每一种数据结构的用处和作用场景!（工具类我写到配置文件里了）

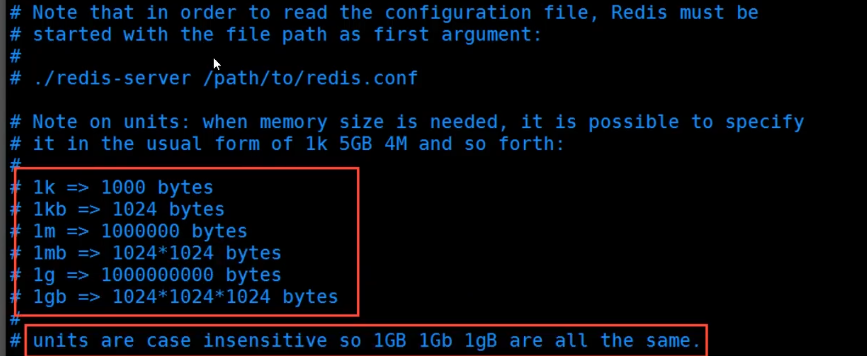


Redis配置文件详解：

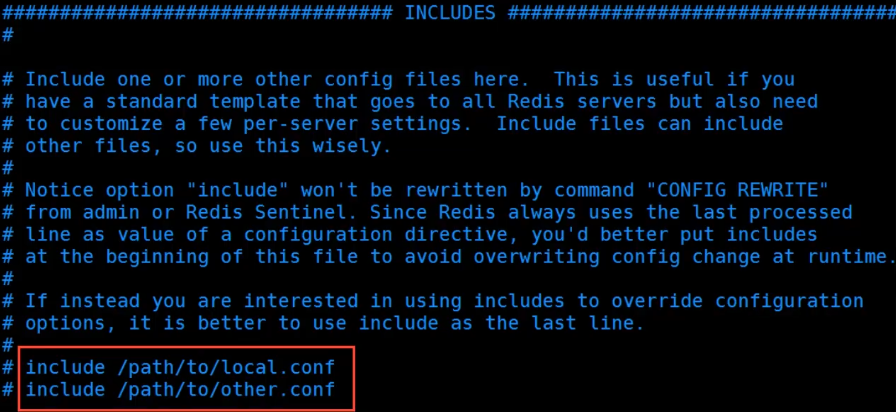
·启动的时候，就通过配置文件来启动!

·工作中，一些小小的配置，可以让你脱颖而出!

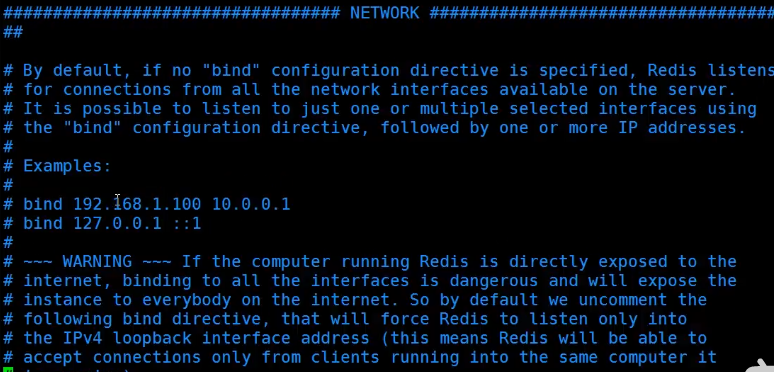
1、大小写不敏感



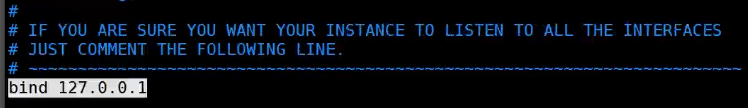
2、可以去包含其他的配置文件，就好比spring里的import



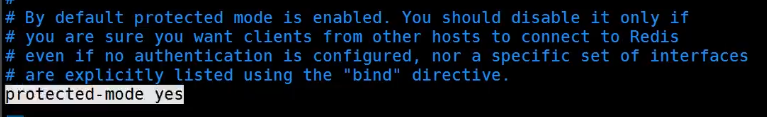
3、网络配置



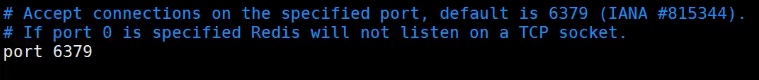
·绑定的ip



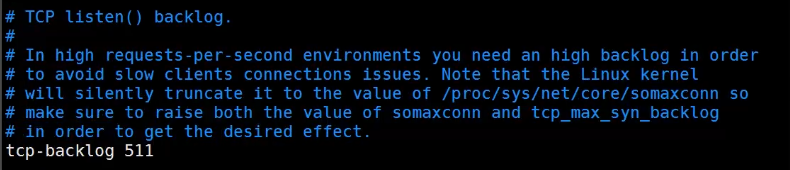
·是否受保护姿态：（默认开启）



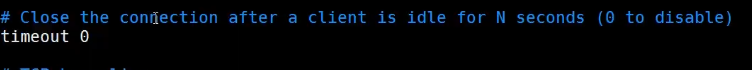
·设置端口



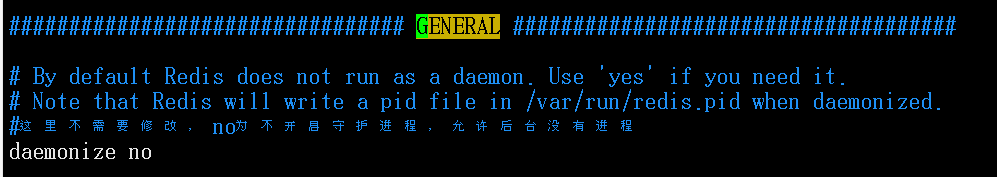
·Tcp连接（不是重点）



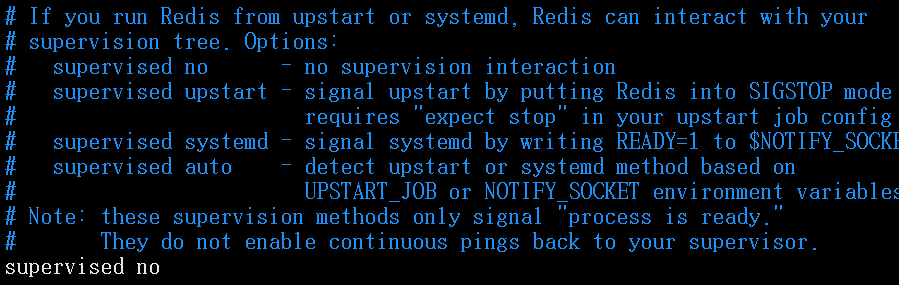
·超时判断或设置：



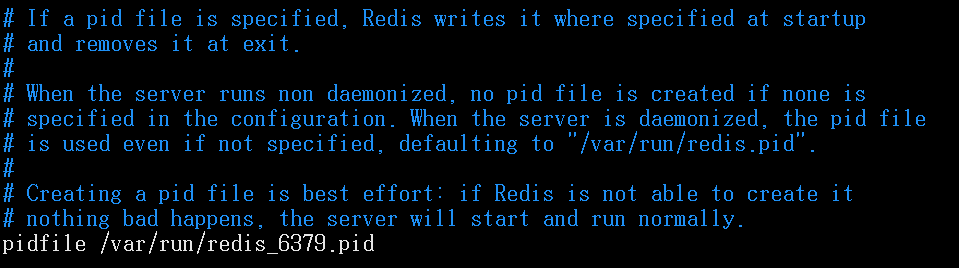
4、通用配置：general

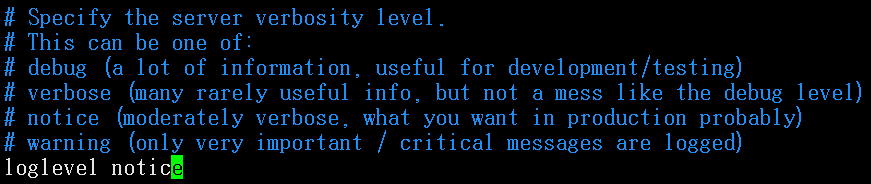
·守护进程关闭，不然后台没有进程就会推出

·管理守护进程（默认是no，不要动

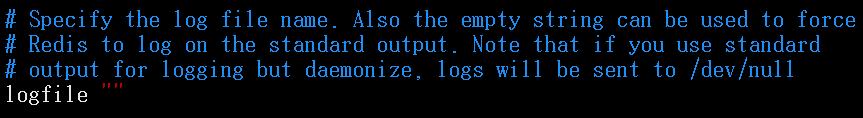


·如果以后台方式运行，redis要指定一个pid文件，是一个进程文件

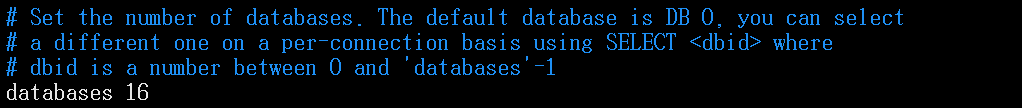


·日志文件：底下是各个警告的级别

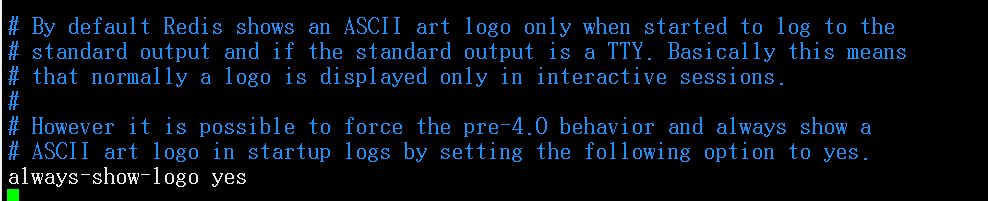
·生成的日志文件位置名



·数据库数量，默认16个



·是否显示logo

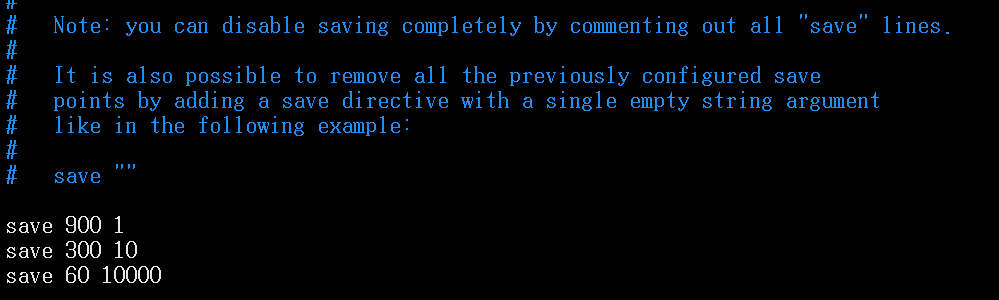


5、快照：

(1)持久化，在规定时间内，执行了多少次操作，则会持久化到文件，即.rdb和.aof文件

(2)Redis是内存数据库，如果没有持久化，那么数据就会断电即失。

·持久化规则



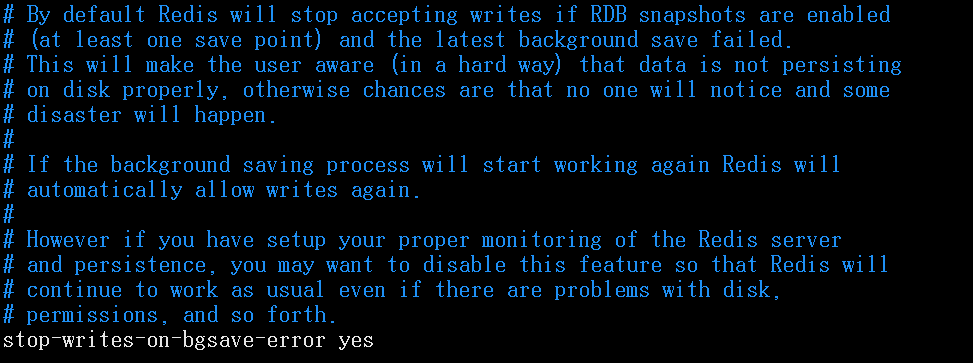
save 900 1 //如果进行1次操作将会激活900秒内，进行一次持久化操作

save 300 10//如果进行10次操作将会激活300秒内，进行一次持久化操作

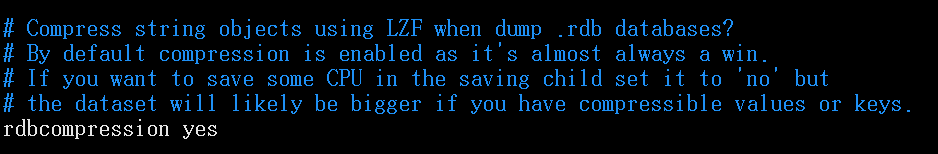
save 60 10000//如果进行10000次操作将会激活60秒内，进行一次持久化操作

#我们之后学习持久化，会自已定义这个测试!

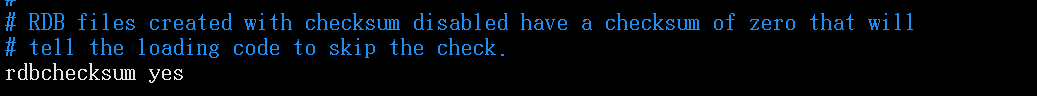
·持久化后是否继续工作：



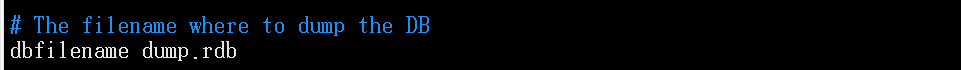
·是否压缩rdb文件（持久化文件），需要消耗一些cpu资源



·rdb文件校验：



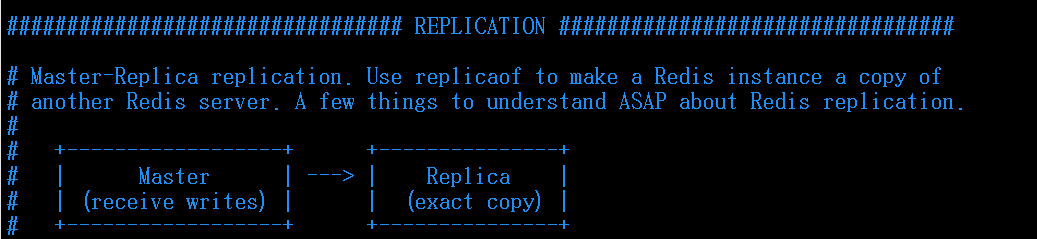
·存放rdb的文件夹名字



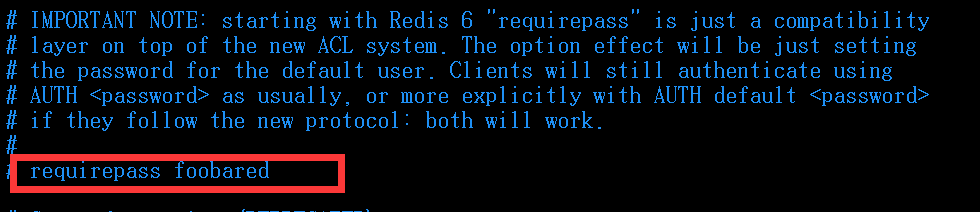
·rdb文件存储目录



6、REPLICATION 复制，我们后面讲解主从复制的，时候再进行讲解



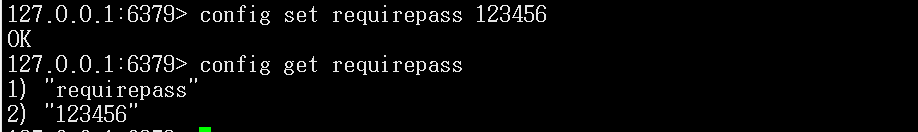
7、redis密码设置



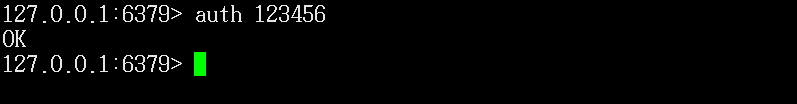
·方法1：在配置文件设置



·方法2：在redis内部写（常用）



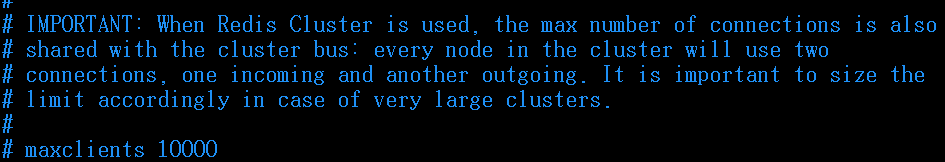
·设置完毕密码后，要进行操作必须使用auth 然后输入密码



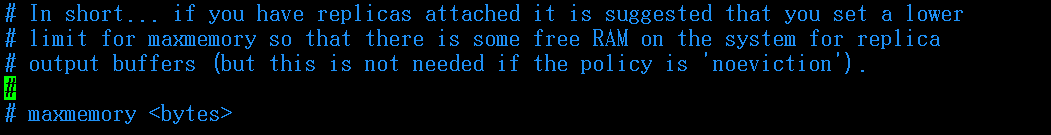
8、客户端



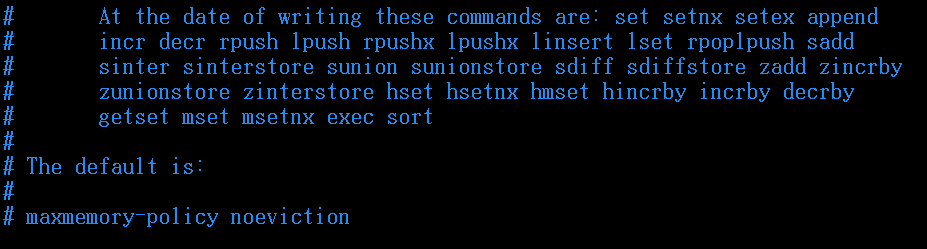
·客户端连接最大数目maxclients



·redis配置最大内存容量maxmemory <bytes>



·内存达到上限后的处理策略 maxmemory-policy noeviction



maxmemory-policy 六种方式

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

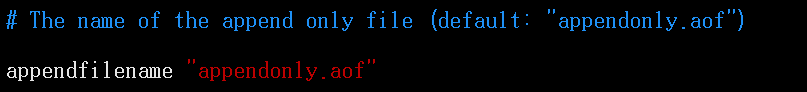
6、noeviction ： 永不过期，返回错误

9、APPEND ONLY 模式aof配置

·aof默认不开启，因为默认使用的是rdb方式持久化，大部分情况下够用了



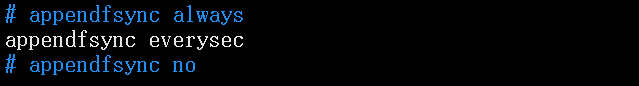
·aof文件的名字appendfilename "appendonly.aof"



·每次都执行一次同步：appendfsync always

·每秒执行一次同步：appendfsync everysec ，一秒内的可能丢失

·不执行同步：appendfsync no，操作系统会自己同步



**Redis持久化：**

·面试和工作，持久化都是重点!

·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!

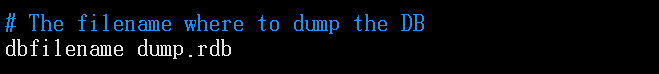
·因为断电即失的存在，所以不得不把东西保存到硬盘上

·在主从复制中rdb就是备用的！从机上面的！

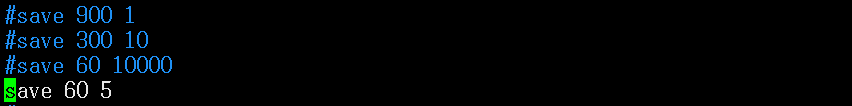
RDB ( Redis DataBase )：

在指定的时间间隔内将内存中的数据集快照写入磁盘，也就是行话讲的Snapshot快照，它恢复时是将快照文件直接读到内存里。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!‘’

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!



1、修改rdb保存规则：



2、在redis里面触发rdb规则次数，生成了dump.db文件

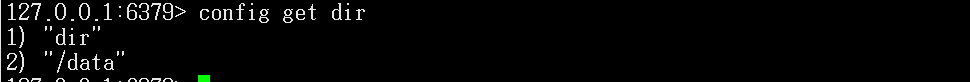


触发机制：

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！

如何恢复rdb文件？

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dump.rdb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!

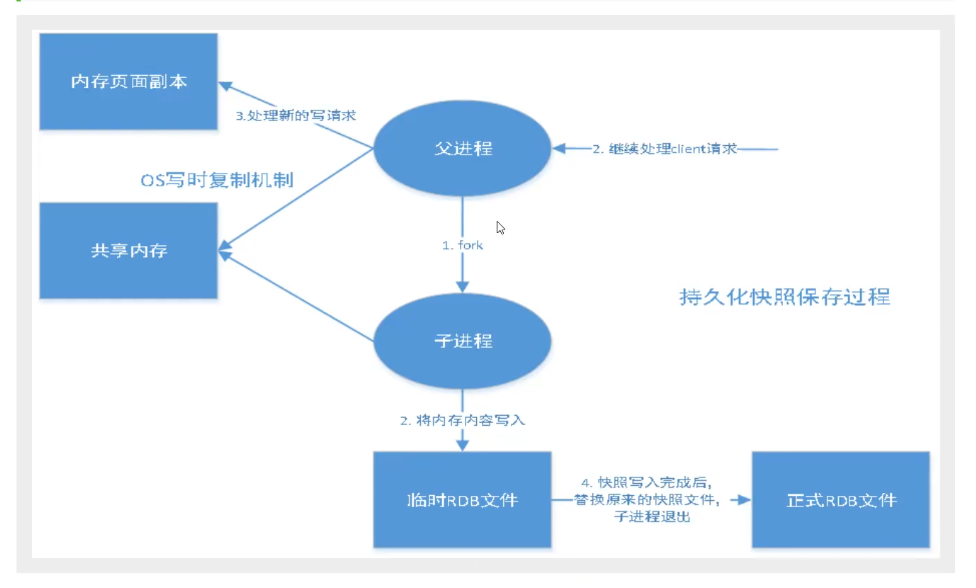
**优点：**

1. 适合大规模的数据恢复！Dump.rdb
2. 对数据的完整性要求不高

**缺点：**

1. 需要一定的时间间隔进行进程操作，如果此时redis宕机了，那最后一次修改的数据可能就丢失了
2. Fork进程的时候，会占用一定进程空间

图解rdb：

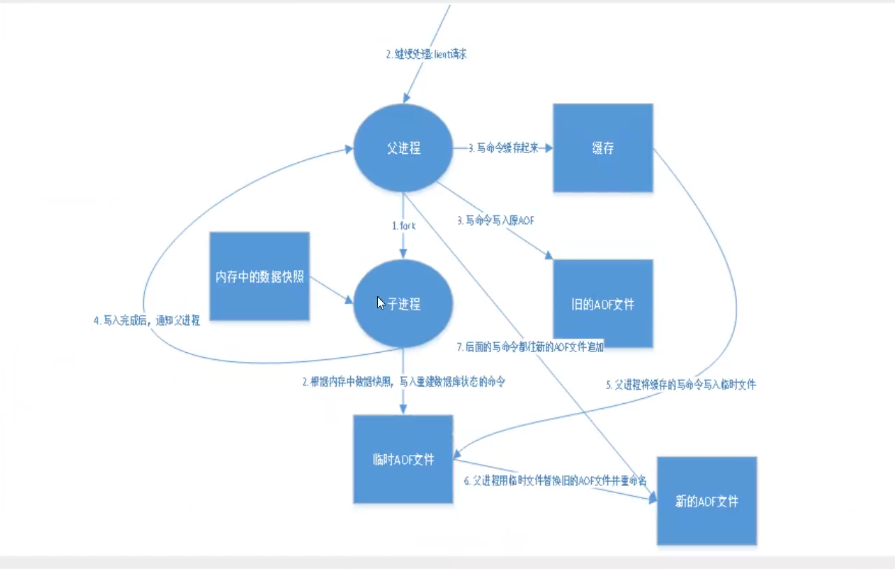


AOF( Append Only File )：

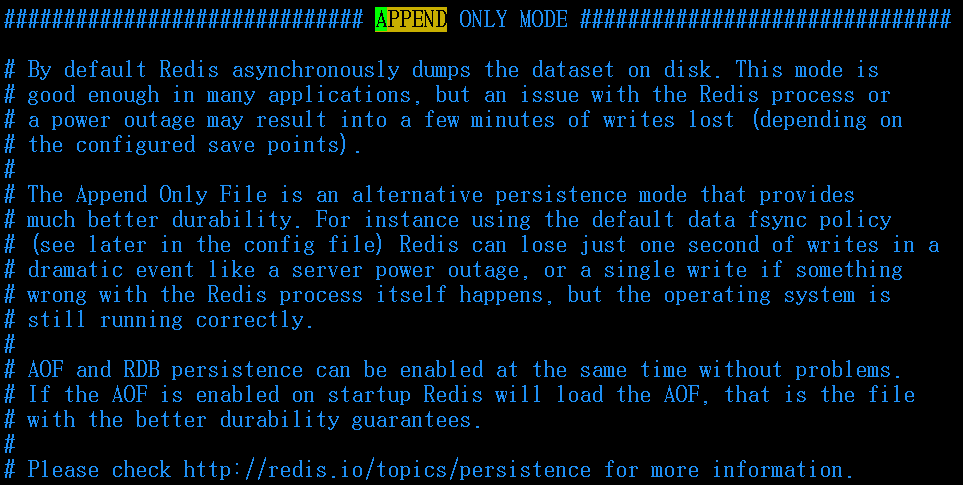
将我们的所有操作命令都记录下来，history，恢复的时候就把这个文件全部在执行一遍！

Aof保存的是appendonly.aof 文件

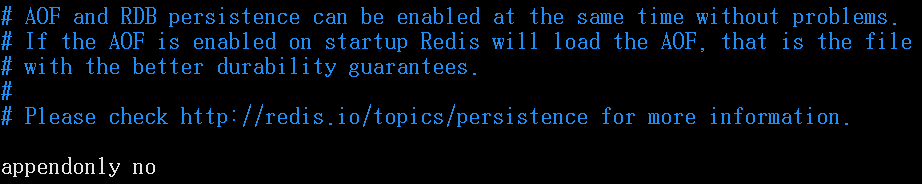
图解aof：



配置保存位置：

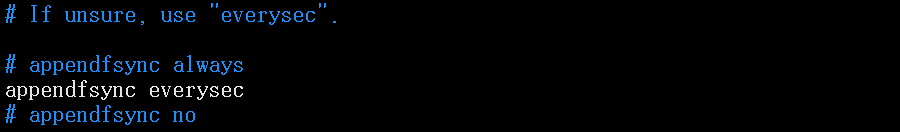


·开启状态：默认不开启

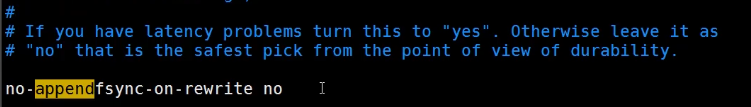


·文件名

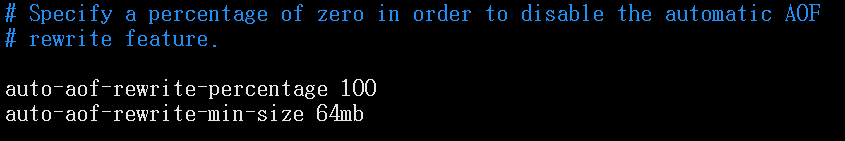


·每秒进行一次同步

·是否进行重写



·重写的规则是100条内，超过64m会再生成一个文件

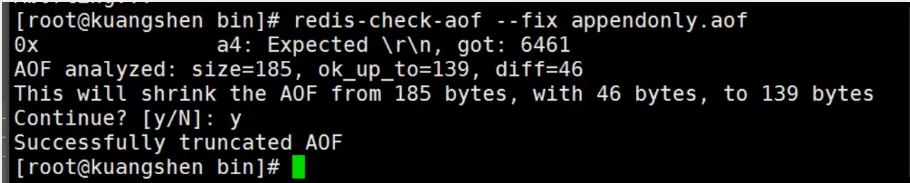


·默认是不开启的，我们需要手动进行配置!我们只需要将appendonly改为yes就开启了aof !重启，redis 就可以生效了!

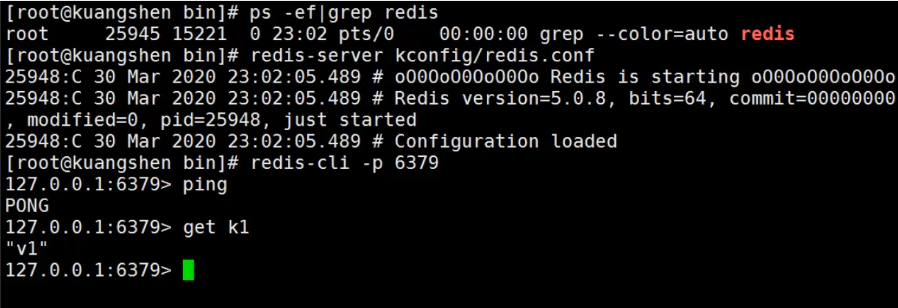
·如果这个aof文件有错位，这时候redis是启动不起来的吗，我们需要修复这个aof文件redis 给我们提供了一个工具redis-check-aof--fix



·使用redis-check-aof--fix恢复文件



·如果文件正常，重启就可以直接恢复了!



优点和缺点!

**优点：**

1. 每一次修改都同步，文件的完整性会更好！
2. 每秒同步一次，可能会丢失一秒的数据
3. 从不同步效率是最高的

**缺点：**

1、相对于数据文件来说aof远远大于rdb，修复速度也比rdb慢！

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

拓展：

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 $001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。

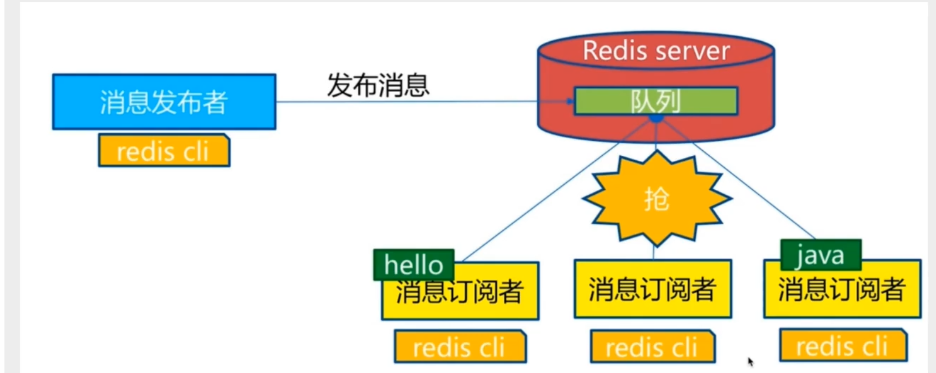
·如果不Enable AOF，仅靠Master-Slave Repllcation实现高可用性也可以，能省掉一大笔l0，也减少了rewrite时带来的系统波动。代价是如果Master/Slave同时倒掉，会丢失十几分钟的数据，启动脚本也要比较两个Master/Slave 中的 RDB文件，载入较新的那个，微博就是这种架构。

**Redis发布订阅：**

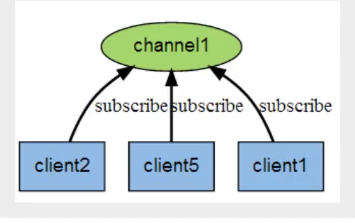
Redis 发布订阅(pub/sub)是一种消息通信模式︰发送者(pub)发送消息，订阅者(sub)接收消息。Redis客户端可以订阅任意数量的频道。也可以使用消息队列，rabbitMQ

订阅/发布消息图︰

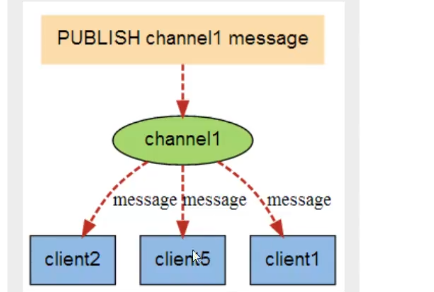
第一个:消息发送者，第二个:频道第三个∶消息订阅者!



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channer.，这个消息就会被发送给订阅它的三个客户端:



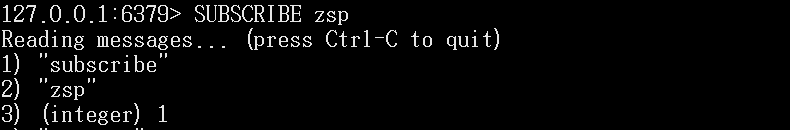
命令：这些命令被广泛用于构建即时通信应用，比如网络聊天室(chatroom)和实时广播、实时提醒等。



测试：

1. 订阅频道

·SUBSCRIBE zsp



1. 从订阅频道发送消息

·publish zsp "hello"



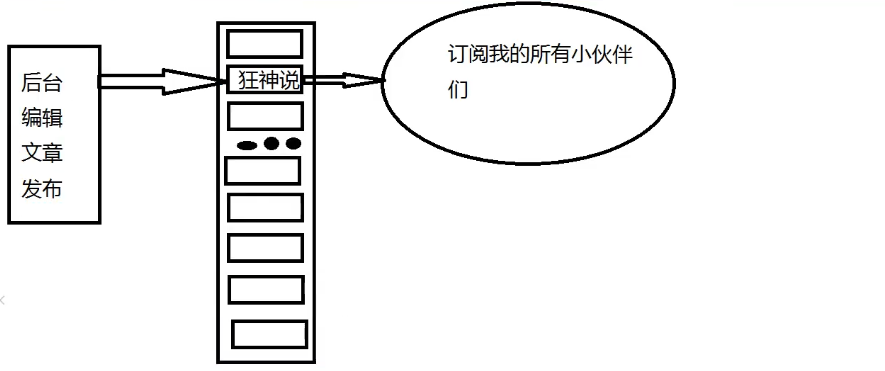
原理：

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

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channel，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。

图解过程：此时订阅的号，就类似于中间的列表



**Redis主从复制：**

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。

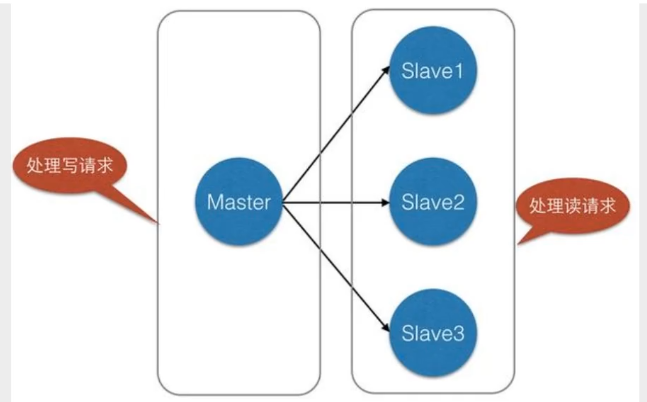
4、高可用基石︰除了上述作用以外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是Redis高可用的基础。

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∶



主从复制，读写分离!80%的情况下都是在进行读操作!减缓服务器的压力!架构中经常使用!最低配：1主3从。

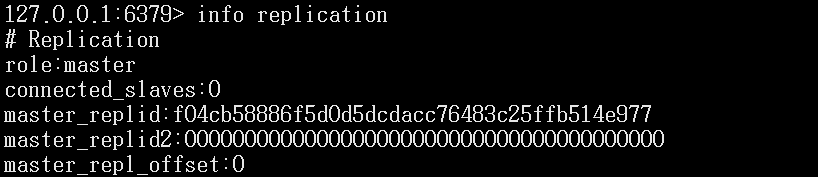
只要在公司中，主从复制就是必须要使用的，因为在真实的项目中不可能单机使用Redis ! )

**Linx下redis集群环境配置：**

·只需要配置从库，不需要配置主库！

·查看当前redis的信息：

info replication

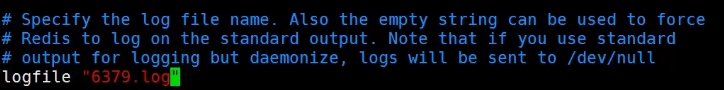


先复制几份配置文件，用于我们使用从机：



主机修改配置文件：

1. 修改日志文件即可

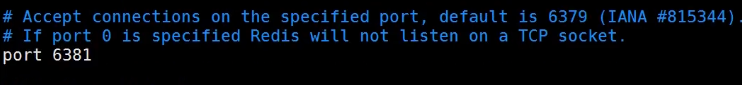


1. 修改db文件名字

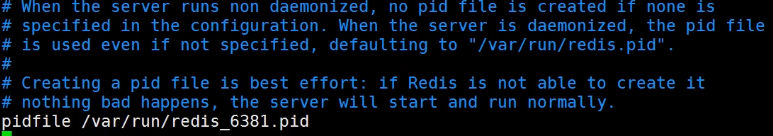


从机修改配置文件：

1、修改端口号port



2、修改后台进程pid文件



3、日志文件名配置log文件名字



4、rdb文件配置



5、启动redis服务（到/usr/local/bin路径 启动redis

redis-server zspconfig/redis.conf



**一主二从（主从配置，俗称认老大）**

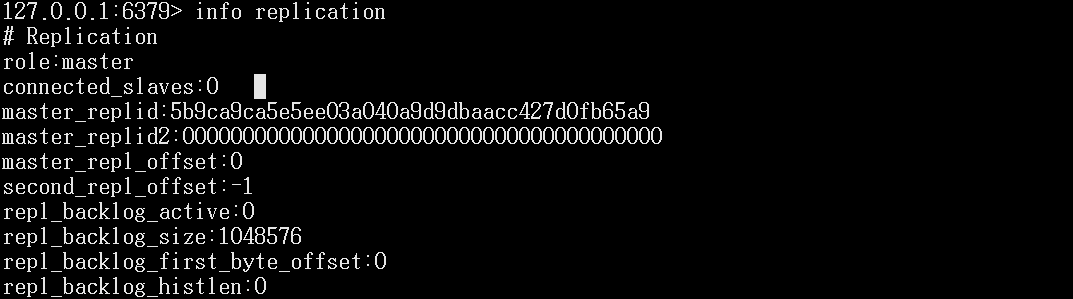
1、进入redis配置从机即可：进入redis的语句

redis-cli -p 6380



2、查看当前主机连接信息

info replication



3、默认情况下，每台Redis服务器都是主节点;我们一般情况下只用配置从机就好了!认老大!一主（79）二从( 80， 81 )：

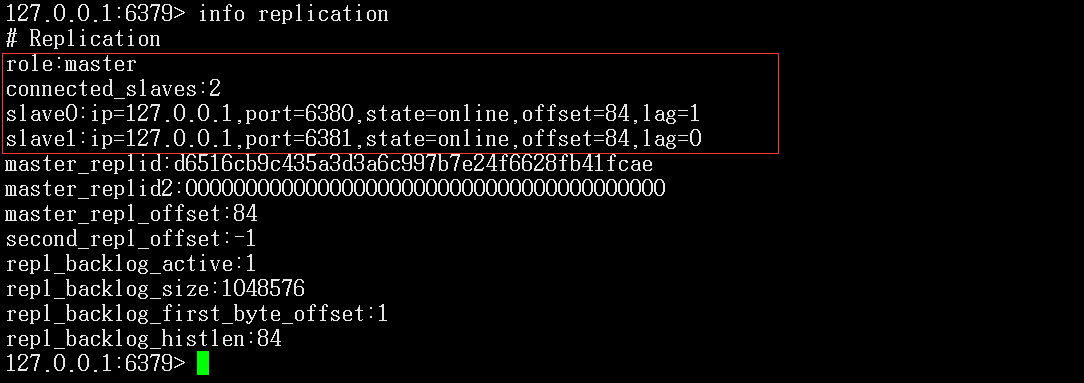
SLAVEOF 127.0.0.1 6379



·取消成为从机

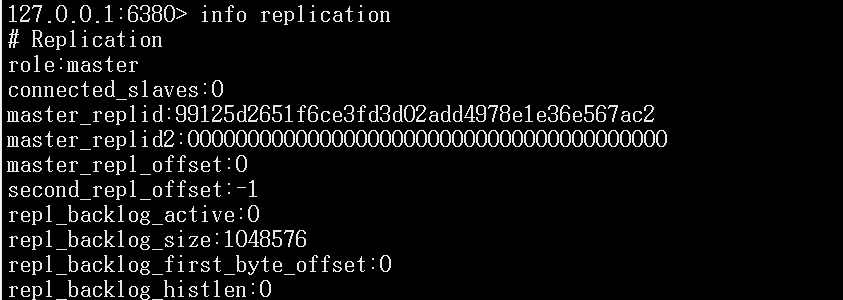
SLAVEOF no one

4、再次查看主机信息：发现多了两台从机配置成功

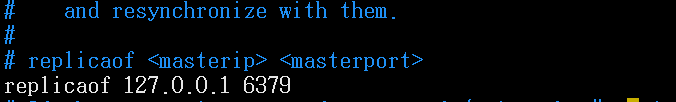


·真实的从主配置应该在配置文件中配置，这样的话是永久的，我们这里使用的是命令，暂时的!

·重启从机6380后发现并不能保存以前的主从关系



·如果要重启后让主机配置还在，必须去配置文件redis80.config里的REPLICATIO里面进行配置：



·再次启动发现已经是从机了



**Docker下配置redis：**

1、

**操作主从复制效果：**

·主从复制：主机负责写入，从机负责查询

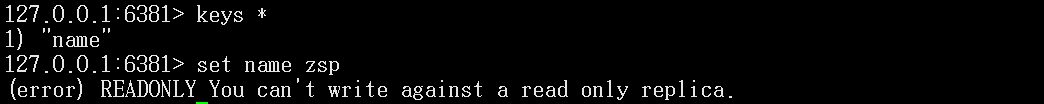
1. 主机放入一个键值对



2、从机可以获取



3、但是从机不可以set值



模拟主机断连：

主机断开连接，从机依旧连接到主机的，但是没有写操作，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!之前写入的信息依旧可以得到



模拟从机断连后，主机放入值：（使用配置文件配置了主机，如果没有则不行）

1、从机断连



1. 主机放入值
2. 从机重新连接，连接后还是可以得到主机放入的值

**复制原理**：

Slave启动成功连接到master后会发送一个sync命令

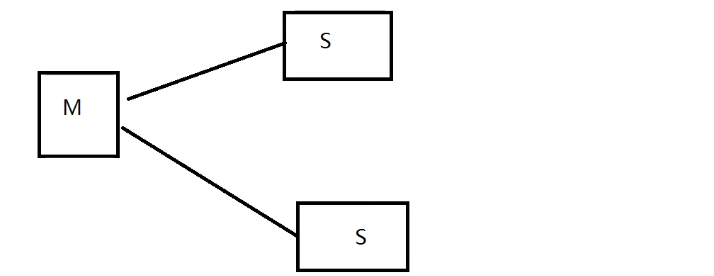
Master接到命令，启动后台的存盘进程，同时收集所有接收到的用于修改数据集命令，在后台进程执行完毕之后，master将传送整个数据文件到slave，**并完成一次完全同步**。

**全量复制**: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。

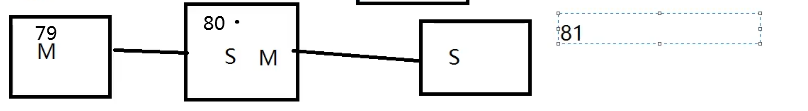
**增量复制**:Master继续将新的所有收集到的修改命令依次传给slave，完成同步但是只要是重新连接master，一次完全同步（全量复制)将被自动执行，我们的数据一定可以在从机中看到。

主从复制示意图：以下两种，工作中都不会使用

1. 第一种效果图：一主多从



1. 层层链路集群：第一个是主节点，第二个是从节点也是第三个的主节点，类似于链表



·此时的80依旧是从节点

层层链路集群实现：

1. 先把6380挂在6379上



1. 把6381挂载在6380上

  
3、检测下主机追加字段从机可不可以获得值

·主机追加



·从机获取





4、尝试6380追加字段：失败



5、关闭6379主机后，6380追加字段：依旧失败



**如果主机宕机了，我们可以手动选取一个从机成为主机：**

·取消成为从机，变回主机，其他从机再连接该从机

SLAVEOF no one



**哨兵模式：(自动选举老大的模式)**

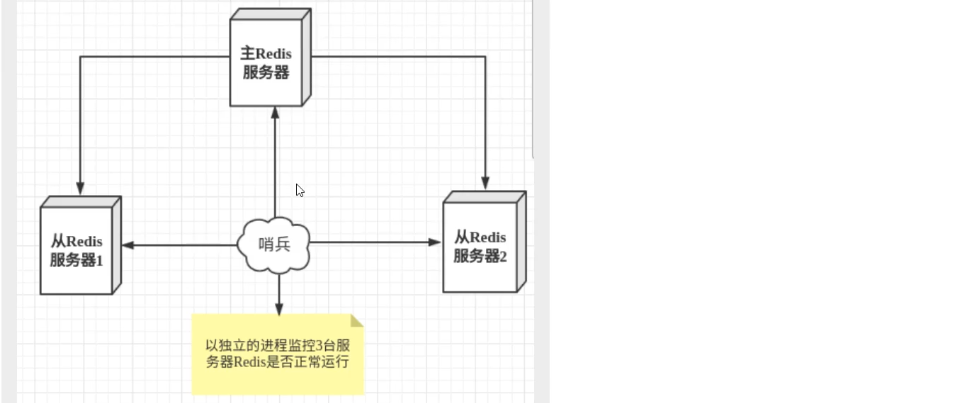
概述：

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。

哨兵模式示意图：



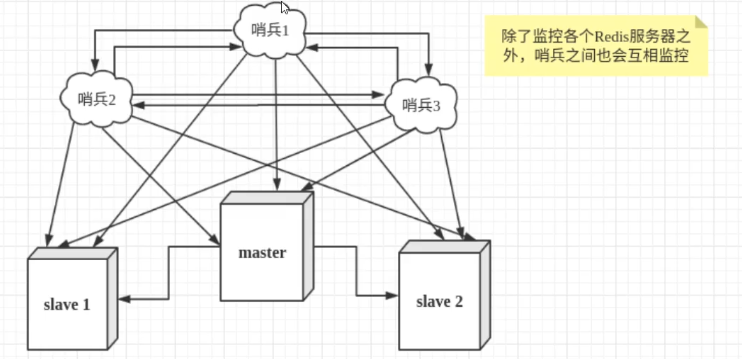
这里的哨兵有两个作用：

·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。

#但一个哨兵进程对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。

哨兵集群：多哨兵检测一个主机的示意图

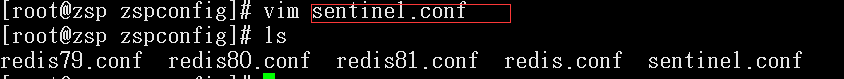


假设主服务器宕机，哨兵1先检测到这个结果，系统并不会马上进行failover（选举）过程，仅仅是哨兵1主观的认为主服务器不可用，这个现象成为**主观下线**。当后面的哨兵也检测到主服务器不可用，并且数量达到一定值时，那么哨兵之间就会进行一次投票，投票的结果由一个哨兵发起，进行failover[故障转移]操作。切换成功后，就会通过发布订阅模式，让各个哨兵把自己监控的从服务器实现切换主机，这个过程称为**客观下线**。

测试使用哨兵模式：

1. 写哨兵模式的配置文件

·创建文件sentinel.conf



·写配置文件

sentinel monitor myredis 127.0.0.1 6379 1



·Sentinel monitor：代表监视谁

·Myredis：代表监视的名称

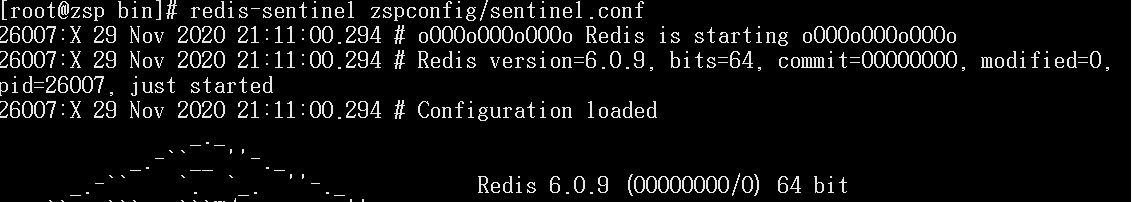
·127.0.0.1：代表ip地址

·6379：代表端口号port

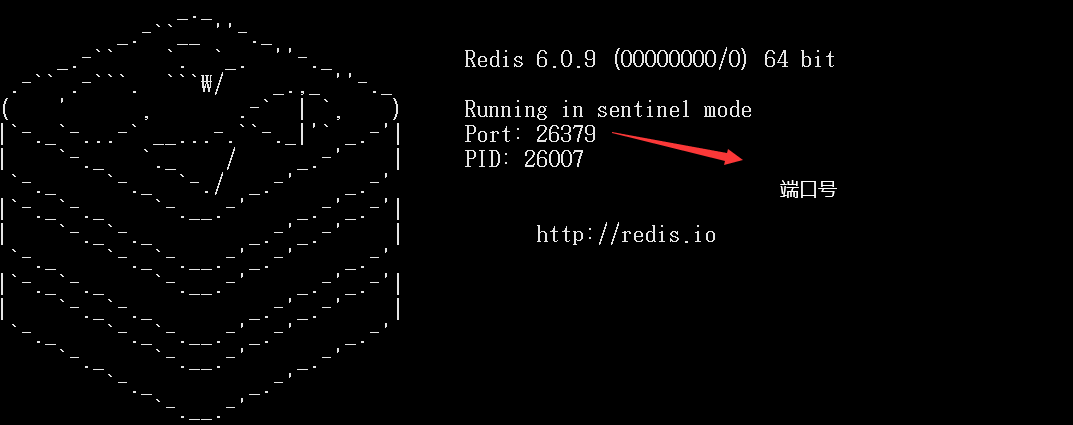
·1：后面的这个数字1，代表主机挂了，slave投票看让谁接替成为主机，票数最多的，就会成为主机!

1. 启动哨兵

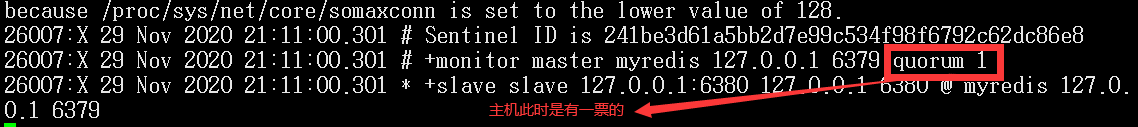
redis-sentinel zspconfig/sentinel.conf



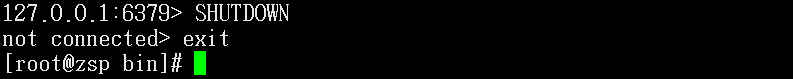
#在这里可以看到端口信息

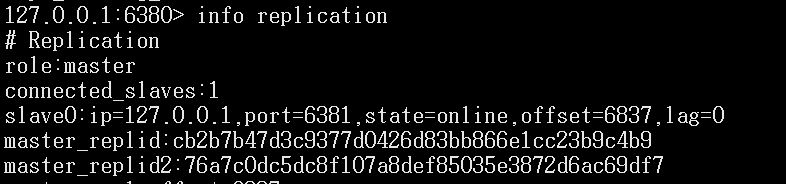


#投票信息

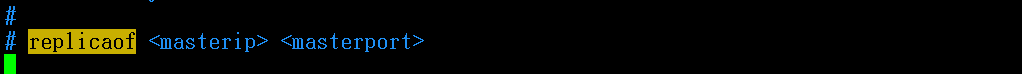


1. 测试一下哨兵模式的选举：
   1. 首先关闭我们的主机

 (2) 查看从机信息：6380成为了新的一个主机



(3) 查看哨兵的信息：哨兵投票给6380成为了新的一个主机

(4)查看一下6380的配置文件，先前我们绑定了6379是主机

#所以说明，哨兵模式是可以选举新主机的，并且，新一任主机的配置文件会被修改

Redis使用注解生成缓存：

@Cacheable（{“name”}）:会先去查询缓存，缓存没有的话会把方法查询下来的数据存到redis中。

@CacheEvict(value=“name”):缓存失效模式，调用方法会去删除缓存。

举例~

查询方法：

@GetMapping("/member/list")  
@Cacheable(value = {"memberList"},key = "#root.methodName")  
public R getMemberList(){}

增删改方法：记得如果写名字要用单引号！！双引号里加单引号再放名字

@GetMapping("/member/add/{userName}/{userPassword}")  
@CacheEvict(value = "memberList",key = "'getMemberList'")