# Redis笔记

# Redis能干吗?

1. **内存存储,持久化**
2. **效率高,可以用于高速缓存**
3. **发布订阅**
4. **地图信息分析**
5. **计时器,计数器(浏览量)**

# Redis在liunx环境下的安装

## 把我们下载好的Redis压缩包，上传到Linux的 /mnt/ 文件目录下

## 安装gcc环境

yum install gcc-c++

## 进入到  /usr/local/redis-x.x.x/ 目录下，进行编译与安装

//进入到/usr/local/redis-5.0.0/ 文件目录下

[root@localhost mnt]# cd /usr/local/redis-5.0.0/

[root@localhost redis-5.0.0]# make //对解压后的文件进行编译

[root@localhost redis-5.0.0]# cd ./src //进入到 redis-5.0.0/src 文件目录下

[root@localhost src]# make install //进行redis安装

## 进入到/usr/local/bin目录下

mkdir config

cd /usr/local/bin/config

cp /usr/local/redis-5.0.7/redis.conf redis.conf

vi redis.conf

修改daemonize为yes

## 启动Redis

cd /usr/local/bin

redis-server config/redis.conf

## 连接Redis(本机)

redis-cli -p 6379

## 关闭Redis服务

redis-cli连接好后直接执行shutdown

# Redis的基础命令

## 选择数据库(默认16个数据库)

select 0

## 清空当前数据库

flushdb

## 清空所有数据库

FLUSHALL

## 查看当前库所有的Key

keys \*

## 设置key多久时间过期

expire key 时间(秒)

expire name 10

## 查看key还要多久过期

ttl key

ttl name

## 判断当前是否存在key

exists key

exists name

## 移除当前的key

move key 1(代表当前库)

move key name 1

## 查看key的类型

type key

# 4.Redis的数据类型

## 1.String(字符串)

#####################################################################

127.0.0.1:6379> set name shibc #设置值

OK

127.0.0.1:6379> get name #获得值

"shibc"

127.0.0.1:6379> keys \* #获得所有的key

1) "name"

127.0.0.1:6379> append name hello #追加字符串,如果当前key不存在,就相当于setkey

(integer) 10

127.0.0.1:6379> get name

"shibchello"

127.0.0.1:6379> strlen name #获取字符串的长度

(integer) 10

127.0.0.1:6379>

#####################################################################

127.0.0.1:6379> set views 0 #初始浏览量

OK

127.0.0.1:6379> get views

"0"

127.0.0.1:6379> incr views #自增! 浏览量+1 如果当前key不存在,就相当于setkey

(integer) 1

127.0.0.1:6379> get views

"1"

127.0.0.1:6379> decr views #自减1 浏览量-1 (integer) 0

127.0.0.1:6379> get views

"0"

127.0.0.1:6379> incrby views 10 #可以设置步长 自定增量 +10

(integer) 10

127.0.0.1:6379> get views

"10"

127.0.0.1:6379> decrby views 5 #可以设置步长 自定减量 -5

(integer) 5

127.0.0.1:6379> get views

"5"

#####################################################################

#字符串范围 range

127.0.0.1:6379> set key hello,shibc

OK

127.0.0.1:6379> get key

"hello,shibc"

127.0.0.1:6379> getrange key 0 3 #截取字符串[0,3]

"hell"

127.0.0.1:6379> getrange key 0 -1 #获取全部的字符串

"hello,shibc"

#替换

127.0.0.1:6379> set key1 abcdefg

OK

127.0.0.1:6379> get key1

"abcdefg"

127.0.0.1:6379> setrange key1 1 xx #替换字符串

(integer) 7

127.0.0.1:6379> get key1

"axxdefg"

#####################################################################

#setex #设置过期时间

#setnx #不存在再设置(在分布式锁中会常常使用)

127.0.0.1:6379> setex key2 60 yaoguifeng #设置key2的值为yaoguifeng,60秒后过期

OK

127.0.0.1:6379> ttl key2

(integer) 55

127.0.0.1:6379> get key2

"yaoguifeng"

127.0.0.1:6379> setnx mykey redis #如果mykey不存在,创建mykey

(integer) 1

127.0.0.1:6379> get mykey

"redis"

127.0.0.1:6379> setnx mykey mysql #如果mykey存在,创建mykey失败

(integer) 0

127.0.0.1:6379> get mykey

"redis"

#####################################################################

#####################################################################

mset

mget

127.0.0.1:6379> mset key1 v1 key2 v2 key3 v3 #同时设置多个key

OK

127.0.0.1:6379> keys \*

1) "key2"

2) "key3"

3) "key1"

127.0.0.1:6379> mget key1 key2 key3 #同时获取多个key

1) "v1"

2) "v2"

3) "v3"

127.0.0.1:6379> msetnx key1 v1 key4 v4 #msetnx是一个原子性操作,要么一起成功,要么一起失败

(integer) 0

#####################################################################

#对象

set user:1 {name:zhangsan,age:3}

#这里的key是一个巧妙的设计: user:{id}:{filed} 如此设计在Redis中晚期ok!

127.0.0.1:6379> mset user:!:name zhangsan user:1:age 8

OK

127.0.0.1:6379> mget user:!:name user:1:age

1) "zhangsan"

2) "8"

#####################################################################

getset

127.0.0.1:6379> getset db redis #如果不存在值,则返回(nil)

(nil)

127.0.0.1:6379> getset db mongodb #如果存在值,获取原来的值.并设置新的值

"redis"

#####################################################################

## List(列表)

#####################################################################

127.0.0.1:6379> lpush list one two three #将一个值或者多个值插入列表头部(左)

(integer) 3

127.0.0.1:6379> lrange list 0 -1 #获取list中的值

1) "three"

2) "two"

3) "one"

127.0.0.1:6379> lrange list 0 1 #通过区间获取list中的值

127.0.0.1:6379> lpush list four

(integer) 4

127.0.0.1:6379> rpush list five #将一个值或者多个值插入列表尾部(右)

(integer) 5

127.0.0.1:6379> lrange list 0 -1

1) "four"

2) "three"

3) "two"

4) "one"

5) "five"

#####################################################################

lpop

rpop

127.0.0.1:6379> lrange list 0 -1

1) "four"

2) "three"

3) "two"

4) "one"

5) "five"

127.0.0.1:6379> lpop list #移除list的第一个元素

"four"

127.0.0.1:6379> rpop list #移除list的最后一个元素

"five"

127.0.0.1:6379> lrange list 0 -1

1) "three"

2) "two"

3) "one"

#####################################################################

lindex

127.0.0.1:6379> lrange list 0 -1

1) "three"

2) "two"

3) "one"

127.0.0.1:6379> lindex list 0 #通过下标获取list的某个值

"three"

127.0.0.1:6379> lindex list 1

"two"

127.0.0.1:6379>

#####################################################################

Llen

127.0.0.1:6379> llen list #获取list的长度

(integer) 3

#####################################################################

移除指定的值

Lrem

127.0.0.1:6379> lrange list 0 -1

1) "three"

2) "two"

3) "one"

127.0.0.1:6379> llen list

(integer) 3

127.0.0.1:6379> lrem list 1 one #移除list集合中指定个数的值,精确匹配

(integer) 1

127.0.0.1:6379> lrange list 0 -1

1) "three"

2) "two"

#####################################################################

trim 修剪

127.0.0.1:6379> lpush mylist hello1 hello 2 hello3 hello4

(integer) 5

127.0.0.1:6379> lrange mylist 0 -1

1) "hello4"

2) "hello3"

3) "2"

4) "hello"

5) "hello1"

127.0.0.1:6379> ltrim mylist 0 2 #通过下标截取指定的长度,这个list已经改变,只剩下截取的

OK

127.0.0.1:6379> lrange mylist 0 -1

1) "hello4"

2) "hello3"

3) "2"

#####################################################################

rpoplpush 移除列表的最后一个元素,将它移动到新的列表中

127.0.0.1:6379> lpush mylist hello1 hello2 hello3

(integer) 3

127.0.0.1:6379> lrange mylist 0 -1

1) "hello3"

2) "hello2"

3) "hello1"

127.0.0.1:6379> rpoplpush mylist otherlist #移除列表的最后一个元素,将它移动到新的列表中

"hello1"

127.0.0.1:6379> lrange mylist 0 -1

1) "hello3"

2) "hello2"

127.0.0.1:6379> lrange otherlist 0 -1

1) "hello1"

#####################################################################

lset 将列表中指定下标的值替换为另外一个值,更新操作

127.0.0.1:6379> lpush list hello

(integer) 1

127.0.0.1:6379> lrange list 0 -1

1) "hello"

127.0.0.1:6379> lset list 0 item #如果下标存在,则会更新当前下标的值

OK

127.0.0.1:6379> lrange list 0 -1

1) "item"

127.0.0.1:6379> lset list 1 item #如果下标不存在,更新会报错

(error) ERR index out of range

#####################################################################

linsert #将某个具体的值插入到列表中某个元素的前面或者后面

127.0.0.1:6379> lpush list hello word

(integer) 2

127.0.0.1:6379> lrange list 0 -1

1) "word"

2) "hello"

127.0.0.1:6379> linsert list before word other

(integer) 3

127.0.0.1:6379> lrange list 0 -1

1) "other"

2) "word"

3) "hello"

#####################################################################

## Set(集合)

#####################################################################

127.0.0.1:6379> sadd myset hello shibc yaogf #set集合中添加元素

(integer) 3

127.0.0.1:6379> sadd myset loveygf #set集合中添加元素

(integer) 1

127.0.0.1:6379> smembers myset #查看指定set的所有值

1) "loveygf"

2) "yaogf"

3) "hello"

4) "shibc"

127.0.0.1:6379> sismember myset hello #判断某一个值是不是在set集合中

(integer) 1

127.0.0.1:6379> sismember myset word

(integer) 0

#####################################################################

127.0.0.1:6379> scard myset #获取set集合中内容元素的个数

(integer) 4

#####################################################################

rem

127.0.0.1:6379> srem myset hello #移除set集合中的指定元素

(integer) 1

127.0.0.1:6379> smembers myset

1) "loveygf"

2) "yaogf"

3) "shibc"

#####################################################################

127.0.0.1:6379> SRANDMEMBER myset

"loveygf"

127.0.0.1:6379> SRANDMEMBER myset #随机抽取某个元素

"shibc"

#####################################################################

127.0.0.1:6379> spop myset #随机删除某个元素

"shibc"

127.0.0.1:6379> spop myset

"loveygf"

#####################################################################

将一个指定的值,移动到另外一个set集合

127.0.0.1:6379> sadd myset hello word shibc

(integer) 3

127.0.0.1:6379> smembers myset

1) "loveygf"

2) "word"

3) "hello"

4) "shibc"

127.0.0.1:6379> sadd myset1 loveshibc

(integer) 1

127.0.0.1:6379> smembers myset1

1) "loveshibc"

127.0.0.1:6379> smove myset myset1 loveygf #将一个指定的值,移动到另外一个set集合

(integer) 1

127.0.0.1:6379> smembers myset1

1) "loveygf"

2) "loveshibc"

127.0.0.1:6379> smembers myset

1) "word"

2) "hello"

3) "shibc"

#####################################################################

数字集合

1. 差集
2. 交集
3. 并集

127.0.0.1:6379> sadd key1 a b c

(integer) 3

127.0.0.1:6379> sadd key2 c d e

(integer) 3

127.0.0.1:6379> sdiff key1 key2 #差集

1) "a"

2) "b"

127.0.0.1:6379> sinter key1 key2 #交集

1) "c"

127.0.0.1:6379> sunion key1 key2 #并集

1) "a"

2) "b"

3) "c"

4) "d"

5) "e"

#####################################################################

## Hash(哈希)

#####################################################################

127.0.0.1:6379> hset myhash field1 hello field2 word #set 具体的key-value

(integer) 2

127.0.0.1:6379> hget myhash field1 #get 具体的key-value

"hello"

127.0.0.1:6379> hgetall myhash #get所有的key-value

1) "field1"

2) "hello"

3) "field2"

4) "word"

127.0.0.1:6379> hdel myhash field1 #删除某个key-value

(integer) 1

127.0.0.1:6379> hgetall myhash

1) "field2"

2) "word"

127.0.0.1:6379> hlen myhash #获取hash的字段数量

(integer) 1

127.0.0.1:6379> hexists myhash field2 #判断hash中指定字段是否存在

(integer) 1

127.0.0.1:6379> hexists myhash field3

(integer) 0

127.0.0.1:6379> hkeys myhash #获取所有的key

1) "field2"

127.0.0.1:6379> hvals myhash #获取所有的value

1) "word"

#####################################################################

127.0.0.1:6379> hset myhash field5 5

127.0.0.1:6379> hincrby myhash field5 1 #可以设置步长 自定增量 +1

127.0.0.1:6379> hsetnx myhash filed6 sss #如果不存在则set成功

(integer) 1

127.0.0.1:6379> hsetnx myhash filed6 bb #如果存在则set失败

(integer) 0

#####################################################################

## 5.Zset(有序集合)

127.0.0.1:6379> zadd myset 1 one #添加一个值

(integer) 1

127.0.0.1:6379> zadd myset 2 two 3 three #添加多个值

(integer) 2

127.0.0.1:6379> zrange myset 0 -1 #查看所有的值

1) "one"

2) "two"

3) "three"

#####################################################################

127.0.0.1:6379> zadd salary 500 shibicheng 1000 yaoguifeng 200000 zhangsan

(integer) 3

127.0.0.1:6379> zrangebyscore salary -inf +inf #显示全部的用户,从小到大

1) "shibicheng"

2) "yaoguifeng"

3) "zhangsan"

127.0.0.1:6379> zrevrange salary 0 -1 #显示全部的用户,从大到小

1) "yaoguifeng"

2) "shibicheng"

127.0.0.1:6379> zrangebyscore salary -inf +inf withscores #显示全部的用户且附带成绩

1) "shibicheng"

2) "500"

3) "yaoguifeng"

4) "1000"

5) "zhangsan"

6) "200000"

127.0.0.1:6379> zrangebyscore salary -inf 2500 withscores #显示工资小于2500的用户且附带成绩

1) "shibicheng"

2) "500"

3) "yaoguifeng"

4) "1000"

#####################################################################

#移除元素zrem

127.0.0.1:6379> zrem salary zhangsan #移除元素

(integer) 1

127.0.0.1:6379> zrange salary 0 -1

1) "shibicheng"

2) "yaoguifeng"

#####################################################################

127.0.0.1:6379> zcard salary #获取有序集合的个数

(integer) 2

127.0.0.1:6379> zadd myset 1 shibc 2 yaoguifeng 3 zhangsan

(integer) 3

127.0.0.1:6379> zcount myset 1 2 #获取指定区间的成员个数

(integer) 2

## 6.三种特殊类型

### geospatial 地理位置

geoadd

指定的地理空间位置（纬度、经度、名称）添加到指定的key中

#####################################################################

127.0.0.1:6379> geoadd city 116.40 39.90 beijing

(integer) 1

127.0.0.1:6379> geoadd city 117.19 39.01 tianjin

(integer) 1

127.0.0.1:6379> geoadd city 121.47 31.23 shanghai 118.77 32.04 nanjing 113.28 23.13 guangzhou 114.09 22.55 shenzhen

(integer) 4

#####################################################################

GEODIST

返回两个给定位置之间的距离，如果两个位置之间的其中一个不存在，那么命令返回空值。

可以指定单位，默认是米

\* m 米

\* km 千米

\* mi 英里

\* ft 英尺

#####################################################################

127.0.0.1:6379> geodist city beijing shanghai km #北京到上海的直线距离

"1067.3788"

#####################################################################

GEOPOS

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 city shanghai beijing shenzhen

1) 1) "121.47000163793563843"

2) "31.22999903975783553"

2) 1) "116.39999896287918091"

2) "39.90000009167092543"

3) 1) "114.09000009298324585"

2) "22.5500010475923105"

#####################################################################

GEORADIUS

以给定的经纬度为中心，返回键包含的位置元素当中，与中心的距离不超过给定最大距离的所有位置元素。

可以指定单位，默认是米

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。

应用：通过给定的半径距离来找附近的人等

#####################################################################

127.0.0.1:6379> georadius city 120 20 1000 km #以120,20为中心，寻找方圆1000米的城市

1) "shenzhen"

2) "guangzhou"

127.0.0.1:6379> georadius city 120 20 1000 km withdist #返回位置元素的同时，将位置元素和中心之间的距离一并返回

1) 1) "shenzhen"

2) "674.9271"

2) 1) "guangzhou"

2) "777.2656"

127.0.0.1:6379> georadius city 120 20 1000 km withcoord #将位置信息和经度纬度一并返回

1) 1) "shenzhen"

2) 1) "114.09000009298324585"

2) "22.5500010475923105"

2) 1) "guangzhou"

2) 1) "113.27999979257583618"

2) "23.13000101271457254"

127.0.0.1:6379> georadius city 120 20 1000 km withcoord count 1 #选取前1个匹配的元素

1) 1) "shenzhen"

2) 1) "114.09000009298324585"

2) "22.5500010475923105"

127.0.0.1:6379> georadius city 120 20 1000 km withcoord asc #从近到远

1) 1) "shenzhen"

2) 1) "114.09000009298324585"

2) "22.5500010475923105"

2) 1) "guangzhou"

2) 1) "113.27999979257583618"

2) "23.13000101271457254"

127.0.0.1:6379> georadius city 120 20 1000 km withcoord desc #从远到近

1) 1) "guangzhou"

2) 1) "113.27999979257583618"

2) "23.13000101271457254"

2) 1) "shenzhen"

2) 1) "114.09000009298324585"

2) "22.5500010475923105"

#####################################################################

GEORADIUSBYMEMBER

找出位于指定元素周围的其他元素

#####################################################################

127.0.0.1:6379> georadiusbymember city beijing 1000 km

1) "nanjing"

2) "tianjin"

3) "beijing"

127.0.0.1:6379> georadiusbymember city beijing 300 km

1) "tianjin"

2) "beijing"

#####################################################################

可以用Zset的命令来操作Geo

#####################################################################

127.0.0.1:6379> zrange city 0 -1

1) "shenzhen"

2) "guangzhou"

3) "shanghai"

4) "nanjing"

5) "tianjin"

6) "beijing"

127.0.0.1:6379> zrem city tianjin

(integer) 1

127.0.0.1:6379> zrange city 0 -1

1) "shenzhen"

2) "guangzhou"

3) "shanghai"

4) "nanjing"

5) "beijing"

#####################################################################

### Hyperloglog(数据结构)

#####################################################################

127.0.0.1:6379> pfadd mkey1 a b c d e f g h i j #创建第一组元素

(integer) 1

127.0.0.1:6379> pfcount mkey1 #统计mkey1的基数

(integer) 10

127.0.0.1:6379> pfadd mkey2 i j z x c v b n m #创建第二组元素

(integer) 1

127.0.0.1:6379> pfcount mkey2

(integer) 9

127.0.0.1:6379> pfmerge mkey3 mkey1 mkey2 #合并两组 mkey1 mkey2=>mkey3 并集

OK

127.0.0.1:6379> pfcount mkey3

(integer) 15

#####################################################################

### Bitmap(位图)

只有两个状态的都可以使用Bitmap 如(打卡,登录未登录等功能)

使用bitmaps来记录周一到周日是否打卡(0表示未打卡,1表示已打卡)

#####################################################################

127.0.0.1:6379> setbit sign 1 0 #添加值

(integer) 0

127.0.0.1:6379> setbit sign 2 1

(integer) 0

127.0.0.1:6379> setbit sign 3 1

(integer) 0

127.0.0.1:6379> setbit sign 4 0

(integer) 0

127.0.0.1:6379> setbit sign 5 1

(integer) 0

127.0.0.1:6379> setbit sign 6 0

(integer) 0

127.0.0.1:6379> setbit sign 7 0

(integer) 0

#####################################################################

查看某一天是否打卡

127.0.0.1:6379> getbit sign 5

(integer) 1

127.0.0.1:6379> getbit sign 6

(integer) 0

统计这周打卡的记录,就可以看到是否全勤

#####################################################################

127.0.0.1:6379> bitcount sign

(integer) 3

#####################################################################

# 5.事务

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

一次性,顺序性,排他性!执行一系列的命令

Redis单条命令是保证原子性的,但是事务不保证原子性!

Redis的事务没有隔离级别的概念!

Redis的事务:

开启事务(multi)

命令入队(...)

执行事务(exec)

#####################################################################

127.0.0.1:6379> multi #开启事务

OK

#命令入队

127.0.0.1:6379> set key1 vi

QUEUED

127.0.0.1:6379> set key2 v2

QUEUED

127.0.0.1:6379> get key2

QUEUED

127.0.0.1:6379> exec #执行事务

1) OK

2) OK

3) "v2"

#####################################################################

放弃事务

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> discard #取消事务,事务中的命令都不会执行

OK

127.0.0.1:6379> get k2

(nil)

#####################################################################

编译型异常(命令有错,事务中所有的命令都不会被执行)

#####################################################################

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> discard

OK

127.0.0.1:6379> get k2

(nil)

127.0.0.1:6379> flushdb

OK

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> getset v3 #错误的命令

(error) ERR wrong number of arguments for 'getset' command

127.0.0.1:6379> exec #执行事务的时候报错

(error) EXECABORT Transaction discarded because of previous errors.

127.0.0.1:6379> get key2

(nil)

#####################################################################

运行时异常(如果事务中存在语法性,那么执行命令的时候,其他命令可以正常的执行,错误的命令抛出异常)

#####################################################################

127.0.0.1:6379> set k1 k1

OK

127.0.0.1:6379> multi

OK

127.0.0.1:6379> incr k1 #执行的失败

QUEUED

127.0.0.1:6379> set k2 v2

QUEUED

127.0.0.1:6379> set k3 v3

QUEUED

127.0.0.1:6379> get k3

QUEUED

127.0.0.1:6379> exec #虽然第一条命令报错,但是依旧执行成功

1) (error) ERR value is not an integer or out of range

2) OK

3) OK

4) "v3"

127.0.0.1:6379> get k2

"v2"

#####################################################################

# 6.redis实现乐观锁

测试多线程修改值,使用watch实现redis的乐观锁

线程一

#####################################################################

127.0.0.1:6379> set money500

OK

127.0.0.1:6379> set out 200

OK

127.0.0.1:6379> watch money #watch监视money

OK

127.0.0.1:6379> multi

OK

127.0.0.1:6379> set money800

QUEUED

127.0.0.1:6379> set out 400

QUEUED

127.0.0.1:6379> exec #执行之前,线程二修改了值,这个时候会导致事务执行失败

(nil)

#####################################################################

线程二

#####################################################################

127.0.0.1:6379> set money99999

OK

#####################################################################

1. 如果发现事务执行失败,就先使用unwatch解锁
2. 再次使用watch监视,获取最新的值
3. 比对监视的值是否发生了变化,如果发生了变化事务就会执行失败,如果没有发生变化则事务执行成功

# 7.Redis的配置文件

## 1.单位

# 1k => 1000 bytes

# 1kb => 1024 bytes

# 1m => 1000000 bytes

# 1mb => 1024\*1024 bytes

# 1g => 1000000000 bytes

# 1gb => 1024\*1024\*1024 bytes

配置文件unit单位大小写不敏感

## 包含(INCLUDES)

# include /path/to/local.conf

# include /path/to/other.conf

可以包含多个配置文件,多个配置文件合成一个

## 网络(NETWORK)

bind 127.0.0.1 #绑定的IP

protected-mode no #保护模式

port 6379 #端口

## 通用(GENERAL)

daemonize yes #以守护进程的方式运行,默认为no,需要修改为yes

pidfile /var/run/redis\_6379.pid #如果以后台方式运行(daemonize为yes),需要指定一个pid文件

# Specify the server verbosity level.

# This can be one of:

# 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)

# warning (only very important / critical messages are logged)

loglevel notice #日志级别

logfile "" #生成日志位置的文件名

databases 16 #默认有16个数据库

## 快照(SNAPSHOTTING)

持久化,在规定的时间内,执行了多少次操作.则会持久化到文件.rdb或者.aof文件

持久化规则

#如果900内,至少有1个key进行了修改,就会进行持久化操作

save 900 1

#如果300内,至少有10个key进行了修改,就会进行持久化操作

save 300 10

#如果60内,至少有10000个key进行了修改,就会进行持久化操作

save 60 10000

stop-writes-on-bgsave-error yes #如果持久化出错,是否继续进行工作

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

## REPLICATION(复制)

配置主机的ip和端口

replicaof 127.0.0.1 6381

## 安全(SECURITY)

requirepass foobared #设置redis的密码,默认没有密码

## 限制(CLIENTS)

maxclients 10000 #设置连接redis最大客户端

maxmemory <bytes> #redis配置最大容量

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

****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

## 9.aof配置(APPEND ONLY MODE)

appendonly no #默认不开起aof模式,默认使用rdb模式持久化的,在大部分的情况下,rdb够用

appendfilename "appendonly.aof" #aof文件名

ppendfsync always #每次修改都会执行sync,消耗性能

appendfsync everysec #每秒执行sync,可能会丢失1s的数据

appendfsync no #不执行sync,这时候操作系统自己同步数据,速度最快

# 8.Redis的持久化

RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。

AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。

#### **RDB存在哪些优势呢？**

1). 一旦采用该方式，那么你的整个Redis数据库将只包含一个文件，这对于文件备份而言是非常完美的。比如，你可能打算每个小时归档一次最近24小时的数据，同时还要每天归档一次最近30天的数据。通过这样的备份策略，一旦系统出现灾难性故障，我们可以非常容易的进行恢复。

2). 对于灾难恢复而言，RDB是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介质上。

3). 性能最大化。对于Redis的服务进程而言，在开始持久化时，它唯一需要做的只是fork出子进程，之后再由子进程完成这些持久化的工作，这样就可以极大的避免服务进程执行IO操作了。

4). 相比于AOF机制，如果数据集很大，RDB的启动效率会更高。

RDB又存在哪些劣势呢？

1). 如果你想保证数据的高可用性，即最大限度的避免数据丢失，那么RDB将不是一个很好的选择。因为系统一旦在定时持久化之前出现宕机现象，此前没有来得及写入磁盘的数据都将丢失。

2). 由于RDB是通过fork子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，可能会导致整个服务器停止服务几百毫秒，甚至是1秒钟。

#### **AOF的优势有哪些呢？**

1). 该机制可以带来更高的数据安全性，即数据持久性。Redis中提供了3中同步策略，即每秒同步、每修改同步和不同步。事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。而每修改同步，我们可以将其视为同步持久化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是最低的。至于无同步，无需多言，我想大家都能正确的理解它。

2). 由于该机制对日志文件的写入操作采用的是append模式，因此在写入过程中即使出现宕机现象，也不会破坏日志文件中已经存在的内容。然而如果我们本次操作只是写入了一半数据就出现了系统崩溃问题，不用担心，在Redis下一次启动之前，我们可以通过redis-check-aof工具来帮助我们解决数据一致性的问题。

3). 如果日志过大，Redis可以自动启用rewrite机制。即Redis以append模式不断的将修改数据写入到老的磁盘文件中，同时Redis还会创建一个新的文件用于记录此期间有哪些修改命令被执行。因此在进行rewrite切换时可以更好的保证数据安全性。

4). AOF包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可以通过该文件完成数据的重建。

AOF的劣势有哪些呢？

1). 对于相同数量的数据集而言，AOF文件通常要大于RDB文件。RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快。

2). 根据同步策略的不同，AOF在运行效率上往往会慢于RDB。总之，每秒同步策略的效率是比较高的，同步禁用策略的效率和RDB一样高效。

二者选择的标准，就是看系统是愿意牺牲一些性能，换取更高的缓存一致性（aof），还是愿意写操作频繁的时候，不启用备份来换取更高的性能，待手动运行save的时候，再做备份（rdb）。rdb这个就更有些 eventually consistent的意思了。

#### **RDB持久化配置**

Redis会将数据集的快照dump到dump.rdb文件中。此外，我们也可以通过配置文件来修改Redis服务器dump快照的频率，在打开6379.conf文件之后，我们搜索save，可以看到下面的配置信息：

save 900 1              #在900秒(15分钟)之后，如果至少有1个key发生变化，则dump内存快照。

save 300 10            #在300秒(5分钟)之后，如果至少有10个key发生变化，则dump内存快照。

save 60 10000        #在60秒(1分钟)之后，如果至少有10000个key发生变化，则dump内存快照。

#### **AOF持久化配置**

在Redis的配置文件中存在三种同步方式，它们分别是：

appendfsync always     #每次有数据修改发生时都会写入AOF文件。

appendfsync everysec  #每秒钟同步一次，该策略为AOF的缺省策略。

appendfsync no          #从不同步。高效但是数据不会被持久化。

# Redis的发布订阅

|  |  |
| --- | --- |
| 1 | [PSUBSCRIBE pattern [pattern ...]](https://www.runoob.com/redis/pub-sub-psubscribe.html) 订阅一个或多个符合给定模式的频道。 |
| 2 | [PUBSUB subcommand [argument [argument ...]]](https://www.runoob.com/redis/pub-sub-pubsub.html) 查看订阅与发布系统状态。 |
| 3 | [PUBLISH channel message](https://www.runoob.com/redis/pub-sub-publish.html) 将信息发送到指定的频道。 |
| 4 | [PUNSUBSCRIBE [pattern [pattern ...]]](https://www.runoob.com/redis/pub-sub-punsubscribe.html) 退订所有给定模式的频道。 |
| 5 | [SUBSCRIBE channel [channel ...]](https://www.runoob.com/redis/pub-sub-subscribe.html) 订阅给定的一个或多个频道的信息。 |
| 6 | [UNSUBSCRIBE [channel [channel ...]]](https://www.runoob.com/redis/pub-sub-unsubscribe.html) 指退订给定的频道。 |
| 1 | [PSUBSCRIBE pattern [pattern ...]](https://www.runoob.com/redis/pub-sub-psubscribe.html) 订阅一个或多个符合给定模式的频道。 |
| 2 | [PUBSUB subcommand [argument [argument ...]]](https://www.runoob.com/redis/pub-sub-pubsub.html) 查看订阅与发布系统状态。 |
| 3 | [PUBLISH channel message](https://www.runoob.com/redis/pub-sub-publish.html) 将信息发送到指定的频道。 |
| 4 | [PUNSUBSCRIBE [pattern [pattern ...]]](https://www.runoob.com/redis/pub-sub-punsubscribe.html) 退订所有给定模式的频道。 |
| 5 | [SUBSCRIBE channel [channel ...]](https://www.runoob.com/redis/pub-sub-subscribe.html) 订阅给定的一个或多个频道的信息。 |
| 6 | [UNSUBSCRIBE [channel [channel ...]]](https://www.runoob.com/redis/pub-sub-unsubscribe.html) 指退订给定的频道。 |

# 10.Redis的主从复制

**一、主从复制概述**

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)，后者称为从节点(slave)；数据的复制是单向的，只能由主节点到从节点。

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

**环境配置**

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

# Replication

role:master #角色

connected\_slaves:0 #没有从机

master\_replid:71b90c5e9ad003f1ce53bc40f44d0b91517bd1ed

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:0

second\_repl\_offset:-1

repl\_backlog\_active:0

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:0

repl\_backlog\_histlen:0

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

**一主二从**

默认情况下，每台Redis服务器都是主节点；

**1.命令配置(不会写入配置文件)**

127.0.0.1:6380> SLAVEOF 127.0.0.1 6379 #SLAVEOF host 6379 找主机

OK

127.0.0.1:6380> 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:9

master\_sync\_in\_progress:0

slave\_repl\_offset:28

slave\_priority:100

slave\_read\_only:1

connected\_slaves:0

master\_replid:65765656833b6ab554dd60221d7f8c4b1366a20a

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:28

second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:1

repl\_backlog\_histlen:28

1. **配置文件(只需配置从机的配置文件)**

replicaof 127.0.0.1 6379 #主机的ip 和端口

# If the master is password protected (using the "requirepass" configuration

# directive below) it is possible to tell the replica to authenticate before

# starting the replication synchronization process, otherwise the master will

# refuse the replica request.

#

# masterauth <master-password> #如果有密码配置密码

**主机可以写,但是从机不能写只能读**

测试:主机断开连接,从机依旧连接到主机的,但是没有写的操作,这个时候,主机如果回来了,立马就会从主机中获取值

**复制原理**

主从节点之间的连接建立以后，便可以开始进行数据同步，该阶段可以理解为从节点数据的初始化。具体执行的方式是：从节点向主节点发送psync命令（Redis2.8以前是sync命令），开始同步。

**全量复制**：用于初次复制或其他无法进行部分复制的情况，将主节点中的所有数据都发送给从节点，是一个非常重型的操作。

**部分复制**：用于网络中断等情况后的复制，只将中断期间主节点执行的写命令发送给从节点，与全量复制相比更加高效。需要注意的是，如果网络中断时间过长，导致主节点没有能够完整地保存中断期间执行的写命令，则无法进行部分复制，仍使用全量复制。

**哨兵模式**

1. **配置哨兵配置文件**

#sentinel monitor 被监控的名称 host prot 1

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

sentinel monitor myredis 127.0.0.1 6379 1

启动哨兵

redis-sentinel config/sentinel.conf

如果主节点断开,这个时候就会从从机中选出一个主机

如果此时主机(宕机)回来了,此时只能当从机不能当主机

**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,系统的高可用性就会更好
3. 哨兵模式就是主从模式的升级,手动到自动,更加健壮

**缺点:**

1. redis不好在线扩容,集群容量一旦达到上限,在线扩容十分麻烦
2. 实现哨兵模式的配置很麻烦,里面有很多选择

# 11.redis的缓存穿透和雪崩

**缓存穿透解决方案**

一个一定不存在缓存及查询不到的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。

有很多种方法可以有效地解决缓存穿透问题，最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被 这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。另外也有一个更为简单粗暴的方法（我们采用的就是这种），如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

### **缓存击穿解决方案**

key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题。

**使用互斥锁(mutex key)**

### **缓存雪崩解决方案**

**加锁排队**