# Redis简介

Redis是Remote Dictionary Server的缩写，是一种高性能的Key-value存储系统，类似于Memcached，不过Redis支持存储的value类型更多，包括String，List，Hash，Set和Sorted Set，以及对这些数据结构的操作支持。和Memcached一样，为了保证效率，Redis的数据都缓存在内存中，但是Redis会周期性地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同步。

# 安装

$ wget http://download.redis.io/releases/redis-3.0.0.tar.gz

$ tar xzf redis-3.0.0.tar.gz

$ cp -r redis-3.0.0 /usr/local/redis

$ cd /usr/local/redis

$ make

# 编译好的文件将被复制到/usr/local/bin下

#redis-server：Redis 服务器的 daemon启动程序

#redis-cli：Redis 命令行操作工具

#redis-benchmark：Redis 性能测试工具，测试 Redis 在你的系统及你的配置下的读写性能

$redis-benchmark -n 100000 –c 50 #模拟同时由50个客户端发送100000 个SETs/GETs 查询

#redis-check-aof：更新日志检查

#redis-check-dump：本地数据库检查

## 配置

修改配置文件，并将其复制到etc 目录下

$cp redis.conf /etc/redis.conf

配置文件基本说明

daemonize： #是否以后台守护进程方式运行

pidfile： #pid 文件位置

port： #监听的端口号

timeout： #请求超时时间

loglevel： #log 信息级别，总共支持四个级别：debug、verbose、notice、warning，默认为verbose

logfile： #默认为标准输出（stdout），如果配置为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null

databases： #开启数据库的数量。使用“SELECT 库ID”方式切换操作各个数据库

save \* \*： #保存快照的频率，第一个\*表示多长时间，第二个\*表示执行多少次写操

作。在一定时间内执行一定数量的写操作时，自动保存快照。可设置多个条件。

rdbcompression：#保存快照是否使用压缩

dbfilename： #数据快照文件名（只是文件名，不包括目录）。默认值为dump.rdb

dir： #数据快照的保存目录（这个是目录）

requirepass： #设置Redis 连接密码，如果配置了连接密码，客户端在连接Redis 时需要通过AUTH <password>命令提供密码，默认关闭。

## 启动redis

$redis-server /etc/redis.conf

## 关闭redis

$ redis-cli shutdown

#关闭指定端口的redis-server

$redis-cli -p 6379 shutdown

## Redis主从复制

只需要修改slave的配置redis.conf

修改如下内容：

Port 6380

slaveof 192.168.220.125 6379 # slaveof master的ip master的端口

主从测试：

在master set

127.0.0.1:6379> set testslave haha

OK

在slave get

127.0.0.1:6380> get testslave

"haha"

# 简单测试

$redis-server /etc/redis.conf

$ redis-cli [-p 6379]

127.0.0.1:6379> set foo bar

OK

127.0.0.1:6379> get foo

"bar"

# Redis数据类型

## String类型及操作

1. Set

设置key对应的值为string类型的value

例如，添加一个name=test的键值对，可以这样做：

127.0.0.1:6379> set name test

OK

1. Setnx

设置key对应的值为string类型的value。如果key已经存在，则返回0，nx是not exist的意思。

例如，添加一个name=test\_new的键值对，可以这样做：

127.0.0.1:6379> get name

“test”

127.0.0.1:6379> setnx name test\_new

(integer) 0

127.0.0.1:6379> get name

“test”

1. Setex

设置key对应的值为string类型的value，并指定此键值对应的有效期。

例如，添加一个sex=male的键值对，并指定其有效期是10秒，可以这样做：

127.0.0.1:6379> setex sex 10 male

127.0.0.1:6379> get sex

“male”

127.0.0.1:6379> get sex

(nil)

最后一次调用的时间已超过10秒范围，因此无法取得sex键对应的值。

1. Setrange

设置指定key的value值的子字符串。

例如，将Test的126邮箱替换为gmail邮箱，可以这样做：

127.0.0.1:6379> get name

“Test@126.com”

127.0.0.1:6379> setrange name 8 gmail.com

“Test@gmail.com”

其中8是指从下标为8（包含8）的字符开始替换。

1. mset

一次设置多个key的值，成功后返回OK，表示所有的值都设置完毕；失败则返回0，表示没有任何值被设置。

127.0.0.1:6379> mset key1 Test1 key2 Test2

OK

127.0.0.1:6379> get key1

“Test1”

127.0.0.1:6379> get key2

“Test2”

1. msetnx

一次设置多个key的值，成功后返回OK，表示所有的值都设置完毕；失败则返回0，表示没有任何值被设置，**但是不会覆盖已经存在的key**。

127.0.0.1:6379> get key1

“Test1”

127.0.0.1:6379> get key2

“Test2”

127.0.0.1:6379> msetnx key2 Test2\_new key3 Test3

(integer) 0

127.0.0.1:6379> get key2

“Test2”

127.0.0.1:6379> get key3

(nil)

可以看出，如果这条命令返回0，那么内部的操作都会回滚，都不会被执行。

1. Get

获取key对应的string值，如果key不存在则返回nil。

例如，获取一个库中存在的键name：

127.0.0.1:6379> get name

“Test”

获取一个库中不存在的键name1，那么会返回一个nil以表示无此键值对。

127.0.0.1:6379> get name1

(nil)

1. getset

设置key的值，并返回key的旧值。

127.0.0.1:6379> get name

“Test”

127.0.0.1:6379> getset name Test\_new

“Test”

127.0.0.1:6379> get name

“Test\_new”

如果key不存在会出现什么状况呢？

127.0.0.1:6379> getset name1 haha

(nil)

1. getrange

获取指定key的value值的子字符串。

127.0.0.1:6379> get name

“Test@126.com”

127.0.0.1:6379> getrange name 0 6

“Test@12”

字符串左面下标是从0开始的。

127.0.0.1:6379> getrange name -7 -1

“126.com”

字符串右面下标是从-1开始的。

127.0.0.1:6379> getrange name 7 100

“6.com”

当下标超出字符串长度时，将默认为是同方向的最大下标。

1. mget

一次获取多个key的值，如果对应的key不存在，则对应返回nil。

127.0.0.1:6379> mget key1 key2 key3

1) “Test1”

2) “Test2”

3) (nil)

1. incr

对key的值进行加加操作，并返回新的值。注意：incr不是int的value会返回错误信息，incr一个不存在的key，则将该key设置为1。

127.0.0.1:6379> set age 20

OK

127.0.0.1:6379> incr age

(integer) 21

127.0.0.1:6379> get age

“21”

1. incrby

同incr类似，与指定值相加，key不存在的时候会设置key，并认为原来的value为0。

127.0.0.1:6379> get age

“21”

127.0.0.1:6379> incrby age 5

(integer) 26

127.0.0.1:6379> get age

“26”

1. decr

对key的值进行减操作，decr一个不存在的key，则设置该key为-1。

127.0.0.1:6379> get age

“26”

127.0.0.1:6379> decr age

(integer) 25

127.0.0.1:6379> get age

“25”

1. decrby

同decr类似，与指定值相减。

127.0.0.1:6379> get age

“25”

127.0.0.1:6379> decrby age 5

“20”

Decrby完全是为了增加代码的可读性，完全可以通过incrby一个负值来实现同样的效果；

127.0.0.1:6379> get age

“20”

127.0.0.1:6379> incrby age -5

(integer) 15

1. append

给指定key的字符串值追加value，返回新字符串值的长度，例如，向name的值追加一个@126.com的字符串，那么可以这样做：

127.0.0.1:6379> append name @126.com

(integer) 15

127.0.0.1:6379> get name

“Test@126.com”

1. strlen

取指定key的value值的长度。

127.0.0.1:6379> get name

“Test@126.com”

127.0.0.1:6379> strlen name

(integer) 12

127.0.0.1:6379> get age

“15”

127.0.0.1:6379> strlen age

(integer) 2

## hash

Redis的hash是一个string类型的field和value的映射表。

1. hset

设置hash field为指定值，如果key不存在，则先创建。

127.0.0.1:6379> hset myhash field1 Hello

(integer) 1

1. hsetnx

设置hash field为指定值，如果key不存在，则先创建。如果field已经存在，则返回0值。nx是not exist的意思。

127.0.0.1:6379> hsetnx myhash field Hello

(integer) 1

127.0.0.1:6379> hsetnx myhash field Hello

(integer) 0

1. hmset

同时设置hash的多个field。

127.0.0.1:6379> hmset myhash field1 Hello field2 World

OK

1. hget

获取指定的hash field。

127.0.0.1:6379> hget myhash field1

“Hello”

127.0.0.1:6379> hget myhash field2

“World”

1. hmget

获取全部指定的hash field。

127.0.0.1:6379> hget myhash field1 field2 field3

1. “Hello”
2. “World”
3. (nil)
4. hincrby

为指定的hash field加上给定值。

127.0.0.1:6379> hset myhash field3 20

(integer) 1

127.0.0.1:6379> hget myhash field3

“20”

127.0.0.1:6379> hincrby myhash field3 -8

(integer) 12

127.0.0.1:6379> hget myhash field3

“12”

1. hexists

测试指定field是否存在。

127.0.0.1:6379> hexists myhash field1

(integer) 1

127.0.0.1:6379> hexists myhash field9

(integer) 0

1. hlen

返回指定hash的field数量。

127.0.0.1:6379> hlen myhash

(integer) 4

可以看到，myhash库中有4个field。

1. hdel

返回指定hash的field数量。

127.0.0.1:6379> hlen myhash

(integer) 4

127.0.0.1:6379> hdel myhash field1

(integer) 1

127.0.0.1:6379> hlen myhash

(integer) 3

1. hkeys

返回hash的所有key。

127.0.0.1:6379> hkeys myhash

1. “key1”
2. “key2”
3. “key3”
4. hvals

返回hash的所有value。

127.0.0.1:6379> hvals myhash

1. “World”
2. “Hello”
3. “12”
4. hgetall

获取某个hash中全部的field及value。

127.0.0.1:6379> hgetall myhash

1. “key1”
2. “World”
3. “key2”
4. “Hello”
5. “key3”
6. “12”

## List

1. lpush

在与key对应的list的头部添加字符串元素。

127.0.0.1:6379> lpush mylist world

(integer) 1

127.0.0.1:6379> lpush mylist hello

(integer) 2

127.0.0.1:6379> lrange mylist 0 -1

1) “hello”

2) “world”

此处先插入一个world，然后在world的头部插入一个hello。其中lrange指令用于获取mylist的内容。

1. rpush

在与key对应的list的尾部添加字符串元素。

127.0.0.1:6379> rpush mylist2 hello

(integer) 1

127.0.0.1:6379> rpush mylist2 world

(integer) 2

127.0.0.1:6379> lrange mylist2 0 -1

1) “hello”

2) “world”

1. linsert

在与key对应的list的特定位置（之前或之后）添加字符串元素。

127.0.0.1:6379> rpush mylist3 hello

(integer) 1

127.0.0.1:6379> rpush mylist3 world

(integer) 2

127.0.0.1:6379> linsert mylist3 before world there

(integer) 3

127.0.0.1:6379> lrange mylist3 0 -1

1) “hello”

2) “there”

3) “world”

先插入一个hello，然后在hello的尾部插入一个world，然后又在world的前面插入了there。

1. lset

设置list中指定下标的元素值（从0开始）

127.0.0.1:6379> rpush mylist4 one

(integer) 1

127.0.0.1:6379> rpush mylist4 two

(integer) 2

127.0.0.1:6379> rpush mylist4 three

(integer) 3

127.0.0.1:6379> lset mylist4 0 four

OK

127.0.0.1:6379> lset mylist4 -2 five

OK

127.0.0.1:6379> lrange mylist4 0 -1

1) “four”

2) “five”

3) “three”

此处依次插入了one，two，three，然后将下标为0的值设置为four，再将下标为-2的值设置为five。

1. lrem

从与key对应的list中删除count个和value相同的元素，使用count查询。

127.0.0.1:6379> rpush mylist5 hello

(integer) 1

127.0.0.1:6379> rpush mylist5 hello

(integer) 2

127.0.0.1:6379> rpush mylist5 foo

(integer) 3

127.0.0.1:6379> rpush mylist5 hello

(integer) 4

127.0.0.1:6379> lrem mylist5 2 hello

(integer) 2

127.0.0.1:6379> lrange mylist5 0 -1

1) “foo”

2) “hello”

count < 0 时，按从尾到头的顺序删除，具体代码如下：

127.0.0.1:6379> rpush mylist6 hello

(integer) 1

127.0.0.1:6379> rpush mylist6 hello

(integer) 2

127.0.0.1:6379> rpush mylist6 foo

(integer) 3

127.0.0.1:6379> rpush mylist6 hello

(integer) 4

127.0.0.1:6379> lrem mylist6 -2 hello

(integer) 2

127.0.0.1:6379> lrange mylist6 0 -1

1) “hello”

2) “foo”

count = 0 时，删除全部，具体代码如下：

127.0.0.1:6379> rpush mylist7 hello

(integer) 1

127.0.0.1:6379> rpush mylist7 hello

(integer) 2

127.0.0.1:6379> rpush mylist7 foo

(integer) 3

127.0.0.1:6379> rpush mylist7 hello

(integer) 4

127.0.0.1:6379> lrem mylist7 0 hello

(integer) 2

127.0.0.1:6379> lrange mylist7 0 -1

1) “foo”

1. ltrim

保留指定key值范围内的数据：

127.0.0.1:6379> rpush mylist8 one

(integer) 1

127.0.0.1:6379> rpush mylist8 two

(integer) 2

127.0.0.1:6379> rpush mylist8 three

(integer) 3

127.0.0.1:6379> rpush mylist8 four

(integer) 4

127.0.0.1:6379> ltrim mylist8 1 -1

(integer) 2

127.0.0.1:6379> lrange mylist8 0 -1

1) “two”

2) “three”

3) “four”

1. lpop

从list的头部删除元素，并返回删除元素的内容：

127.0.0.1:6379> lrange mylist 0 -1

1) “hello”

2) “world”

127.0.0.1:6379> lpop mylist

“hello”

127.0.0.1:6379> lrange mylist 0 -1

1) “world”

1. rpop

从list的尾部删除元素，并返回删除元素的内容：

127.0.0.1:6379> lrange mylist2 0 -1

1) “hello”

2) “world”

127.0.0.1:6379> rpop mylist2

“world”

127.0.0.1:6379> lrange mylist2 0 -1

1) “hello”

1. rpoplpush

从第一个list的尾部移除元素并将其添加到第二个list的头部，最后返回被移除的元素值，整个操作是原子性的。如果第一个list为空或者不存在，则返回nil。

127.0.0.1:6379> lrange mylist5 0 -1

1) “three”

2) “foo”

2) “hello”

127.0.0.1:6379> lrange mylist6 0 -1

1) “hello”

2) “foo”

127.0.0.1:6379> rpoplpush mylist5 mylist6

“hello”

127.0.0.1:6379> lrange mylist5 0 -1

1) “three”

2) “foo”

127.0.0.1:6379> lrange mylist6 0 -1

1) “hello”

2) “hello”

3) “foo”

1. lindex

返回名称为key的list中index位置的元素。

127.0.0.1:6379> lrange mylist5 0 -1

1) “three”

2) “foo”

127.0.0.1:6379> lindex mylist5 0

1) “three”

127.0.0.1:6379> lindex mylist5 1

1) “foo”

1. llen

返回与key对应的list的长度。

127.0.0.1:6379> llen mylist5

(integer) 2

## Set

Redis的set是string类型的无序集合，set元素最多可以包含232个元素。关于set集合类型除了基本的添加，删除操作，其他操作还包括对集合取并集（union）、交集（intersection）、差集（difference）。

1. sadd

向名称为key的set中添加元素。

127.0.0.1:6379> sadd myset hello

(integer) 1

127.0.0.1:6379> sadd myset world

(integer) 1

127.0.0.1:6379> sadd myset world

(integer) 0

127.0.0.1:6379> smembers myset

1) “world”

2) “hello”

此处向myset中添加了3个元素，但由于第3个元素跟第2个元素相同，所以第3个元素没有添加成功。最后用smembers来查看myset中的所有元素。

1. srem

删除名称为key的set中的元素member。

127.0.0.1:6379> sadd myset2 one

(integer) 1

127.0.0.1:6379> sadd myset2 two

(integer) 1

127.0.0.1:6379> sadd myset2 three

(integer) 1

127.0.0.1:6379> srem myset2 one

(integer) 1

127.0.0.1:6379> srem myset2 four

(integer) 0

127.0.0.1:6379> smembers myset2

1) “three”

2) “two”

此处向myset2中添加了3个元素后，再调用srem来删除one和four，但由于元素中没有four，所以srem命令执行失败。

1. spop

随机返回并删除名称为key的set中的一个元素。

127.0.0.1:6379> smembers myset2

1) “two”

2) “three”

3) “one”

127.0.0.1:6379> spop myset2

"three"

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

1. sdiff

返回所有给定key与第一个key的差集。  
127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> sdiff myset2 myset3

1) "one"

可以看出，myset2和myset3中不同的元素只是“one”所以只有one出现在查询结果中。

也可以将myset2和myset3交换顺序后再查看结果。

127.0.0.1:6379> sdiff myset3 myset2

1) "three"

1. sdiffstore

返回所有给定key与第一个key的差集，并将结果保存为另一个key。

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> sdiffstore myset4 myset2 myset3

(integer) 1

127.0.0.1:6379> smembers myset4

1) “one”

1. sinter

返回所有给定key的交集。

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> sinter myset2 myset3

1) “two”

1. sinterstore

返回所有给定key的交集，并将结果存为另一个key。

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> sinterstore myset5 myset2 myset3

(integer) 1

127.0.0.1:6379> smembers myset5

1) “two”

1. sunion

返回所有给定key的并集。

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> sunion myset2 myset3

1) “two”

2) “three”

3) “one”

1. sunionstore

返回所有给定key的并集，并将结果保存为另一个key。

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> sunionstore myset6 myset2 myset3

(integer) 3

127.0.0.1:6379> smembers myset6

1) “two”

2) “three”

3) “one”

1. smove

从第一个key对应的set中移除member并将其添加到第二个对应的set中。

127.0.0.1:6379> smembers myset2

1) “two”

2) “one”

127.0.0.1:6379> smembers myset3

1) “two”

2) “three”

127.0.0.1:6379> smove myset2 myset7 one

(integer) 1

127.0.0.1:6379> smembers myset7

1) “one”

127.0.0.1:6379> smembers myset2

1) “two”

可以看出myset2的成员one被移到myset7中了。

1. scard

返回名称为key的set的元素个数。

127.0.0.1:6379> scard myset2

(integer) 1

1. sismember

测试member是否为名称为key的set中的元素。

127.0.0.1:6379> smembers myset2

1) “two”

127.0.0.1:6379> sismember myset2 two

(integer) 1

127.0.0.1:6379> sismember myset2 one

(integer) 0

1. srandmember

随机返回名称为key的set中的一个元素，但不删除该元素。

127.0.0.1:6379> smembers myset3

1) “two”

2) “one”

127.0.0.1:6379> srandmember myset3

“two”

127.0.0.1:6379> srandmember myset3

“one”

## Sorted sets

和set一样，sorted set也是string类型元素的集合，不同的是每个元素都会关联一个double类型的score。

1. zadd

向名称为key的zset中添加元素(member)，并使用score指令进行排序。如果该元素已经存在，则更新该元素的顺序。

127.0.0.1:6379> zadd myzset 1 one

(integer) 1

127.0.0.1:6379> zadd myzset 2 two

(integer) 1

127.0.0.1:6379> zadd myzset 3 two

(integer) 0

127.0.0.1:6379> zrange myzset 0 -1 withscores

1) “one”

2) “1”

3) “two”

2) “3”

此处向myzset中添加了one和two成员，并且two被设置了2次，那么将以最后一次的设置为准，最后将所有元素按顺序显示出来。

1. zrem

删除名称为key的zset中的元素（member）。

127.0.0.1:6379> zadd myzset 0 -1 withscores

1) “one”

2) “1”

3) “two”

2) “3”

127.0.0.1:6379> zrem myzset two

(integer) 1

127.0.0.1:6379> zrange myzset 0 -1 withscores

1) “one”

2) “1”

1. zincrby

如果在名称为key的zset中已经存在元素（member），则该元素的score增加（increment）；

否则向集合中添加该元素，其score的值为increment。

127.0.0.1:6379> zadd myzset2 1 one

(integer) 1

127.0.0.1:6379> zadd myzset2 2 two

(integer) 1

127.0.0.1:6379> zincrby myzset2 2 one

“3”

127.0.0.1:6379> zrange myzset2 0 -1 withscores

1) “two”

2) “2”

3) “one”

4) “1”

这里将元素one的score从1增加了2，即增加到了3。

1. zrank

返回名称为key的zset中member的排名（按score从小到大排序），排名以 0 为底，也就是说， score 值最小的成员排名为 0 。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zrank myzset3 two

(integer) 1

1. zrevrank

返回名称为key的zset中member的排名（按score从大到小排序）。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zrevrank myzset3 two

(integer) 2

1. zrevrange

在名称为key的zset中，检索从开始到结束的所有元素，按score从大到小进行排序，再取出全部元素。

127.0.0.1:6379> zrevrange myzset3 0 -1 withscores

1) “five”

2) “5”

3) “three”

4) “3”

5) “two”

6) “2”

7) “one”

8) “1”

1. zrangebyscore

返回集合中score在给定区间的元素。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zrangebyscore myzset3 2 3 withscores

1) “two”

2) “2”

3) “three”

4) “3”

1. zcount

返回集合中score在给定区间的数量。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zcount myzset3 2 3

(integer) 2

1. zcard

返回集合中元素的个数。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zcard myzset3

(integer) 4

1. zscore

返回给定元素对应的score。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zscore myzset3 two

“2”

1. zremrangebyrank

删除集合中排名在给定区间的元素。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

7) “five”

8) “5”

127.0.0.1:6379> zremrangebyrank myzset3 3 3

(integer) 1

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

1. zremrangebyscore

删除集合中排名在给定区间的元素。

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “one”

2) “1”

3) “two”

4) “2”

5) “three”

6) “3”

127.0.0.1:6379> zremrangebyscore myzset3 1 2

(integer) 1

127.0.0.1:6379> zrange myzset3 0 -1 withscores

1) “three”

2) “3”

# 持久化

通常Redis将数据存储在内存中或虚拟内存中，它是通过以下两种方式实现对数据的持久化。在实际运营中，注意备份持久化文件以防止数据丢失。

## 快照方式：（默认持久化方式）

这种方式就是将内存中数据以快照的方式写入到二进制文件中，默认的文件名为dump.rdb。

客户端也可以使用save 或者bgsave 命令通知redis 做一次快照持久化。save 操作是在主线程中保存快照的，由于redis 是用一个主线程来处理所有客户端的请求，这种方式会阻塞所有客户端请求。所以不推荐使用。另一点需要注意的是，每次快照持久化都是将内存数据完整写入到磁盘一次，并不是增量的只同步增量数据。如果数据量大的话，写操作会比较多，必然会引起大量的磁盘IO 操作，可能会严重影响性能。

注意：由于快照方式是在一定间隔时间做一次的，所以如果redis 意外当机的话，就会丢失最后一次快照后的所有数据修改。

* 数据快照配置：

save 900 1

save 300 10

save 60 10000

以上在redis.conf中的配置指出在多长时间内,有多少次更新操作,就将数据同步到数据文件,这个可以多个条件配合。上面的含义是900秒后有一个key发生改变就执行save,300秒后有10个key发生改变执行save,60秒有10000个key发生改变执行save。

## 日志追加方式

这种方式redis 会将每一个收到的写命令都通过write 函数追加到文件中(默认appendonly.aof)。当redis 重启时会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。当然由于操作系统会在内核中缓存write做的修改，所以可能不是立即写到磁盘上。这样的持久化还是有可能会丢失部分修改。不过我们可以通过配置文件告诉redis 我们想要通过fsync 函数强制操作系统写入到磁盘的时机。有三种方式如下（默认是：每秒fsync 一次）

appendonly yes //启用日志追加持久化方式

#appendfsync always //每次收到写命令就立即强制写入磁盘，最慢的，但是保证完全的持久化，不推荐使用

appendfsync everysec //每秒钟强制写入磁盘一次，在性能和持久化方面做了很好的折中，推荐

#appendfsync no //完全依赖操作系统，性能最好,持久化没保证

日志追加方式同时带来了另一个问题。持久化文件会变的越来越大。例如我们调用incr test 命令100次，文件中必须保存全部100 条命令，其实有99 条都是多余的。因为要恢复

数据库状态其实文件中保存一条set test 100 就够了。为了压缩这种持久化方式的日志文件。

redis 提供了**bgrewriteaof**命令。收到此命令 redis 将使用与快照类似的方式将内存中的数据以命令的方式保存到临时文件中，最后替换原来的持久化日志文件。

# 集群

Redis 集群是一个可以在多个 Redis 节点之间进行数据共享的设施，Redis 集群使用数据分片（sharding）而非一致性哈希（consistency hashing）来实现： 一个 Redis 集群包含 16384 个哈希槽（hash slot）， 数据库中的每个键都属于这 16384 个哈希槽的其中一个，集群提供了以下两个好处：

* 将数据自动切分（split）到多个节点的能力。
* 当集群中的一部分节点失效或者无法进行通讯时， 仍然可以继续处理命令请求的能力。

下面就来着手创建一个Redis集群。

## 集群配置

以下是一个包含了最少选项的集群配置文件示例：

port 7000

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

appendonly yes

文件中的cluster-enabled选项用于打开实例的集群模式，而cluster-conf-file选项则设定了保存节点配置文件的路径，默认值为nodes.conf。节点配置文件无须人为修改，它由 Redis集群在启动时创建，并在有需要时自动进行更新。

## 创建集群实例

首先进入一个新目录，并创建六个以端口号为名字的子目录，稍后我们在将每个目录中运行一个Redis实例：

mkdir cluster-test

cd cluster-test

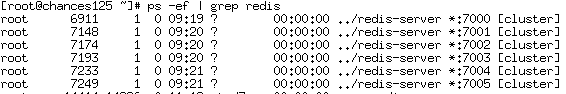
mkdir 7000 7001 7002 7003 7004 7005

* 在文件夹7000至7005中，各创建一个redis.conf文件，文件的内容可以使用上面的示例配置文件，但记得将配置中的端口号改为与文件夹名字相同的号码。
* 复制redis-server 文件到cluster-test 文件夹， 然后使用类似以下命令， 在每个标签页中打开一个实例：

cd 7000

../redis-server ./redis.conf

至此，6个集群实例已全部启动，如下图所示：



## 配置实例通讯

现在我们已经有了6个Redis 实例，并以Redis Cluster的方式运行着，但并没有自动构建集群，彼此都还是独立的节点，验证如下：

./redis-cli -p 7000 cluster info

cluster\_state:fail

cluster\_slots\_assigned:0

cluster\_slots\_ok:0

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:1

cluster\_size:0

cluster\_current\_epoch:0

cluster\_my\_epoch:0

cluster\_stats\_messages\_sent:0

cluster\_stats\_messages\_received:0

接下来我们来创建集群，让6个实例互相通讯：

./redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005

/usr/local/rvm/rubies/ruby-2.0.0-p451/lib/ruby/site\_ruby/2.0.0/rubygems/core\_ext/kernel\_require.rb:55:in `require': cannot load such file -- redis (LoadError)

from /usr/local/rvm/rubies/ruby-2.0.0-p451/lib/ruby/site\_ruby/2.0.0/rubygems/core\_ext/kernel\_require.rb:55:in `require'

from ../redis-3.0.0/src/redis-trib.rb:25:in `<main>'

redis-trib 位于Redis源码的 src 文件夹中，它是一个Ruby程序，这个程序通过向实例发送特殊命令来完成创建集群，从上图报错发现，操作系统没有安装[gem-redis]，所以需要执行以下命令完成安装：

gem install redis

安装信息：

Fetching: redis-3.2.1.gem (100%)

Successfully installed redis-3.2.1

Parsing documentation for redis-3.2.1

Installing ri documentation for redis-3.2.1

1 gem installed

之后重新执行以下命令，便可依提示完成集群建设：

./redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005

>>> Creating cluster

Connecting to node 127.0.0.1:7000: OK

Connecting to node 127.0.0.1:7001: OK

Connecting to node 127.0.0.1:7002: OK

Connecting to node 127.0.0.1:7003: OK

Connecting to node 127.0.0.1:7004: OK

Connecting to node 127.0.0.1:7005: OK

>>> Performing hash slots allocation on 6 nodes...

Using 3 masters:

127.0.0.1:7000

127.0.0.1:7001

127.0.0.1:7002

Adding replica 127.0.0.1:7003 to 127.0.0.1:7000

Adding replica 127.0.0.1:7004 to 127.0.0.1:7001

Adding replica 127.0.0.1:7005 to 127.0.0.1:7002

M: 409f3c36392405dab1a98863aca38b1a70a8d9b4 127.0.0.1:7000

slots:0-5460 (5461 slots) master

M: 0de19d6c1054b57faad32c08c64e00455c26afde 127.0.0.1:7001

slots:5461-10922 (5462 slots) master

M: 4b3a15c90158205b386707049a8806bce00be24c 127.0.0.1:7002

slots:10923-16383 (5461 slots) master

S: e96aec079249c8414bb13e0bba5481382170b80c 127.0.0.1:7003

replicates 409f3c36392405dab1a98863aca38b1a70a8d9b4

S: 8079cfa03127743d095b3dfdbd87c417633a7d56 127.0.0.1:7004

replicates 0de19d6c1054b57faad32c08c64e00455c26afde

S: 36fd24dc8fd879678ad171ed90d3269db6bd1c75 127.0.0.1:7005

replicates 4b3a15c90158205b386707049a8806bce00be24c

Can I set the above configuration? (type 'yes' to accept): **yes**

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join...

>>> Performing Cluster Check (using node 127.0.0.1:7000)

M: 409f3c36392405dab1a98863aca38b1a70a8d9b4 127.0.0.1:7000

slots:0-5460 (5461 slots) master

M: 0de19d6c1054b57faad32c08c64e00455c26afde 127.0.0.1:7001

slots:5461-10922 (5462 slots) master

M: 4b3a15c90158205b386707049a8806bce00be24c 127.0.0.1:7002

slots:10923-16383 (5461 slots) master

M: e96aec079249c8414bb13e0bba5481382170b80c 127.0.0.1:7003

slots: (0 slots) master

replicates 409f3c36392405dab1a98863aca38b1a70a8d9b4

M: 8079cfa03127743d095b3dfdbd87c417633a7d56 127.0.0.1:7004

slots: (0 slots) master

replicates 0de19d6c1054b57faad32c08c64e00455c26afde

M: 36fd24dc8fd879678ad171ed90d3269db6bd1c75 127.0.0.1:7005

slots: (0 slots) master

replicates 4b3a15c90158205b386707049a8806bce00be24c

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.（这表示集群中的 16384 个槽都有至少一个主节点在处理， 集群运作正常）

简单来说，以上命令的意思就是让redis-trib程序创建一个包含三个主节点和三个从节点的集群。接着redis-trib会打印出一份预想中的配置给你看，如果你觉得没问题的话，就可以输入yes，redis-trib 就会将这份配置应用到集群当中，命令中的选项--replicas 1 表示为集群中的每个主节点创建一个从节点；下面再来验证下集群配置情况：

./redis-cli -p 7000 cluster info

cluster\_state:ok

cluster\_slots\_assigned:16384

cluster\_slots\_ok:16384

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:6

cluster\_size:3

cluster\_current\_epoch:6

cluster\_my\_epoch:1

cluster\_stats\_messages\_sent:142

cluster\_stats\_messages\_received:142

## 集群测试

$ redis-cli -c -p 7000

127.0.0.1:7000> set foo bar

-> Redirected to slot [12182] located at 127.0.0.1:7002

OK

127.0.0.1:7002> set hello world

-> Redirected to slot [866] located at 127.0.0.1:7000

OK

127.0.0.1:7000> get foo

-> Redirected to slot [12182] located at 127.0.0.1:7002

"bar"

127.0.0.1:7000> get hello

-> Redirected to slot [866] located at 127.0.0.1:7000

"world"

# 参考

Redis 指令在线模拟练习：

<http://try.redis-db.com/>

Redis 指令大全：

<http://redisdoc.com/>

集群教程：

<http://redisdoc.com/topic/cluster-tutorial.html>