

第一章 Redis初识

2018年6月23日 10:02

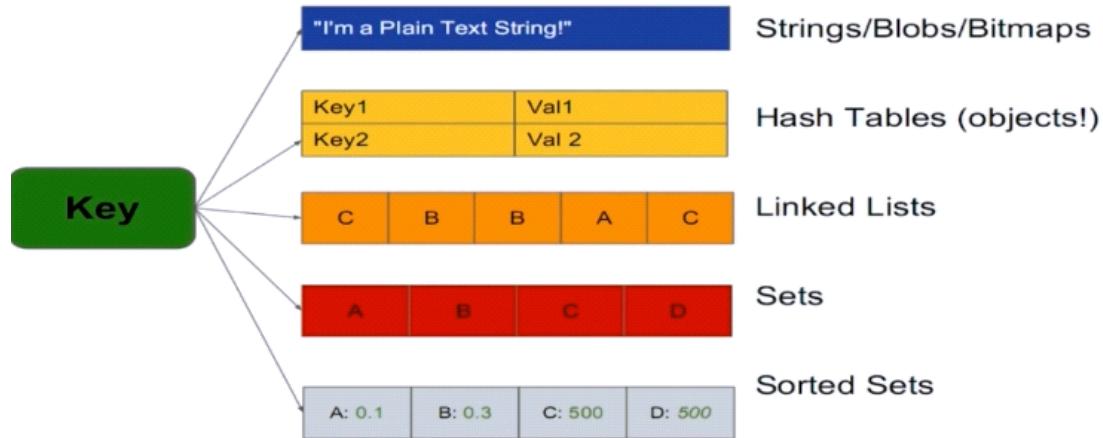
1.Redis是什么？

- (1) 开源
- (2) 是基于键值对 (key-value) 的存储服务系统 (数据库)



- (3) 支持多种数据结构

数据结构



- (4) 性能非常高，功能非常丰富

2.Redis的特性

- (1) 速度快 (真正原因就是内存)
 - ①Redis的所有数据都是存放在内存中的
 - ②Redis使用C语言实现的
 - ③Redis使用单线程架构

速度快-内存

类型	每秒读写次数	随机读写延迟	访问带宽
内存	千万级	80ns	5GB
SSD盘	35000	0.1-0.2ms	100~300MB
机械盘	100左右	10ms	100MB左右

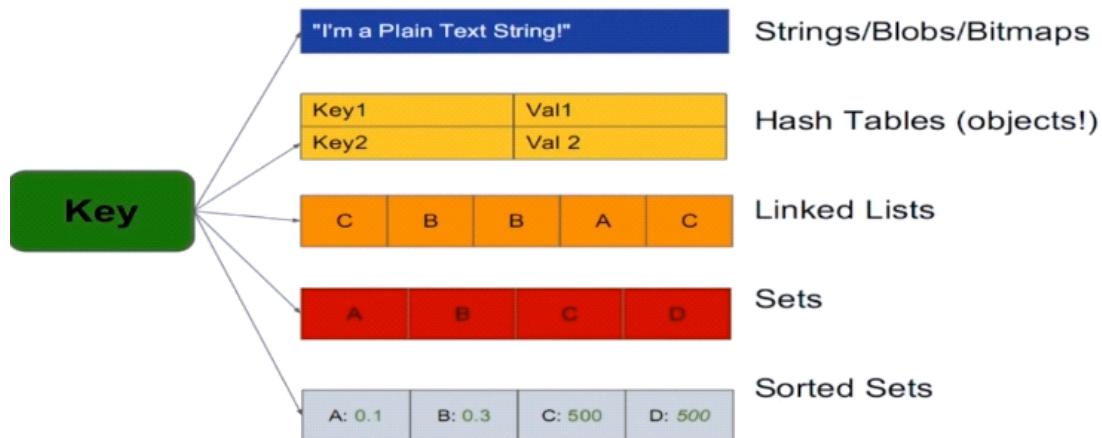
(2) 持久化

通常数据放在内存中是不安全的，一旦发生断电或机械故障，重要的数据就有可能会丢失，因此Redis提供了两种持久化的方式：RDB和AOF，既可以通过两种策略将内存中的数据保存在硬盘中，这样就保证了数据的可持久性

(3) 数据结构 (主要提供5种数据结构)

更多... [了解更多](#)

数据结构



另外还有在字符串的基础上演变的位图 (Bitmaps) 和HyperLogLog两种数据结构，另外还有GEO (地理位置信息)

(4) 支持多种客户端语言

(5) 功能丰富

(6) 简单稳定

①不依赖于外部的库②服务模型是单线程的

(7) 主从复制 (是高可用和分布式一个重要的知识点)

(8) 高可用和分布式

Redis特性(8)-高可用、分布式

高可用



Redis-Sentinel(v2.8)支持高可用

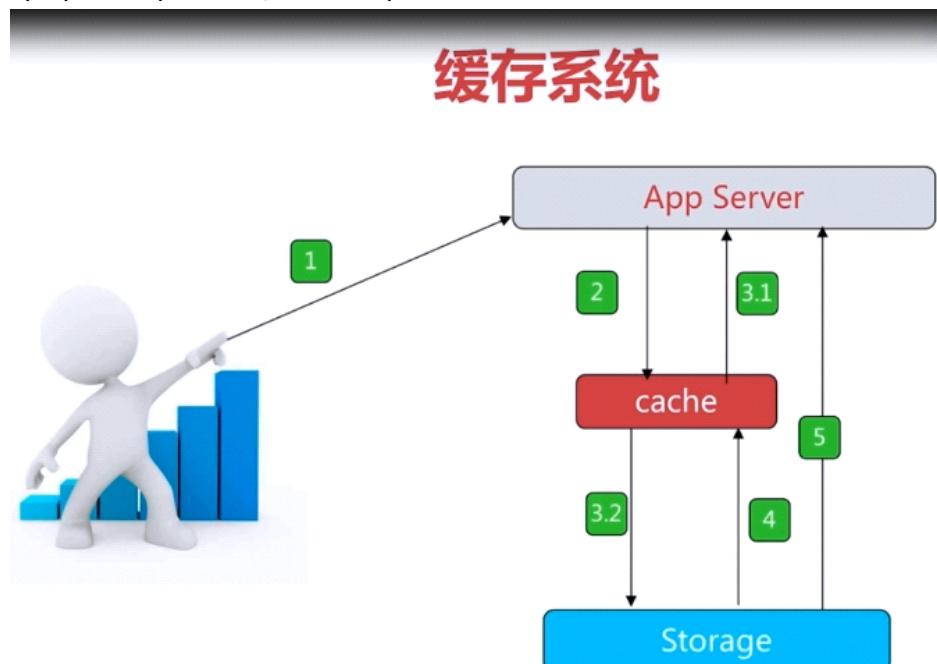
分布式



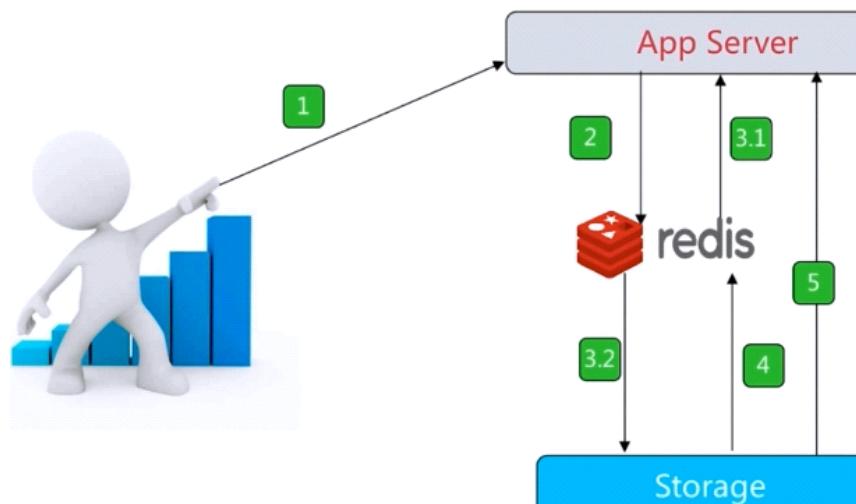
Redis-Cluster(v3.0)支持分布式

2.Redis典型使用场景

(1) 缓存 (典型的使用场景)



缓存系统



(2) 计数器

(3) 消息队列 (发布订阅)

(4) 排行榜

(5) 社交网络

3.Redis的安装

(1) 三种启动方式

验证

```
ps -ef | grep redis
```

```
netstat -antpl | grep redis
```

```
redis-cli -h ip -p port ping
```

动态参数启动Redis

◆ redis-server --port 6380

配置启动Redis

◆ redis-server configPath

三种启动方式比较

- ◆ 生成环境选择配置启动
- ◆ 单机多实例配置文件可以用端口区分开

Redis客户端连接

```
[@zw-34-55 ~]# redis-cli -h 10.10.79.150 -p 6384
10.10.79.150:6384> ping
PONG
10.10.79.150:6384> set hello world
OK
10.10.79.150:6384> get hello
"world"
```

了解客户端的返回值对于后序的开发以及日常的运维是非常有帮助的

Redis客户端返回值

状态回复



```
10.10.79.150:6384> ping
PONG
```

错误回复



```
10.10.79.150:6384> hget hello field
(error) WRONGTYPE Operation against
```

整数回复



```
10.10.79.150:6384> incr hello
(integer) 1
```

Redis客户端返回值(续)

字符串回复



10.10.53.159:6379> get hello
"world"

多行字符串回复



10.10.53.159:6379> mget hello foo
1) "world"
2) "bar"

(2) 常用设置 (Redis的默认端口是6379)

更多一手教程加

Redis常用配置

daemonize



是否是守护进程(no|yes)

port



Redis对外端口号

logfile



Redis系统日志

dir



Redis工作目录



第二章

2018年6月23日 16:31

Redis API的使用和理解

通用命令

◆ 通用命令

◆ 单线程架构

◆ 数据结构和内部编码

1.通用命令 (6种)

(1) 查看所有的键

```
[imooc@localhost src]$ ./redis-cli
127.0.0.1:6379> set hello world
OK
127.0.0.1:6379> set php good
OK
127.0.0.1:6379> set java beat
OK
127.0.0.1:6379> keys *
1) "java"
2) "php"
3) "hello"

127.0.0.1:6379> mset hello world hehe haha php good phe his
OK
127.0.0.1:6379> keys he*
1) "hello"
2) "hehe"
127.0.0.1:6379> keys he[h-l]*
1) "hello"
2) "hehe"
127.0.0.1:6379> keys ph?
1) "php"
2) "phe"

(2) 键的总数
127.0.0.1:6379> dbsize
(integer) 3

(3) 键是否存在
127.0.0.1:6379> set a b
OK
127.0.0.1:6379> exit a
[imooc@localhost src]$ ./redis-cli
127.0.0.1:6379> exists a
(integer) 1
127.0.0.1:6379> del a b
(integer) 1
127.0.0.1:6379> exists a b
(error) ERR unknown command 'exists'
127.0.0.1:6379> exist a b
(error) ERR unknown command 'exist'
127.0.0.1:6379> exists a b
(integer) 0

(4) 删除键
Del

(5) 键过期
```

expire、ttl、persist

API expire key seconds
#key在seconds秒后过期

API ttl key
#查看key剩余的过期时间

API persist key
#去掉key的过期时间

```
127.0.0.1:6379> set hello world
OK
127.0.0.1:6379> expire hello 20
(integer) 1
127.0.0.1:6379> persist hello
(integer) 1
127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379> get hello
"world"

ttl有三种返回值：(1) 大于或等于0的整数：表示键剩余的时间
(2) -1：没设置过期时间
(3) -2：键不存在
(6) type：返回key的类型
```

type

API type key #返回key的类型

演示

```
127.0.0.1:6379> set a b
OK
127.0.0.1:6379> type a
string
127.0.0.1:6379> sadd myset 1 2 3
(integer) 3
127.0.0.1:6379> type myset
set
```

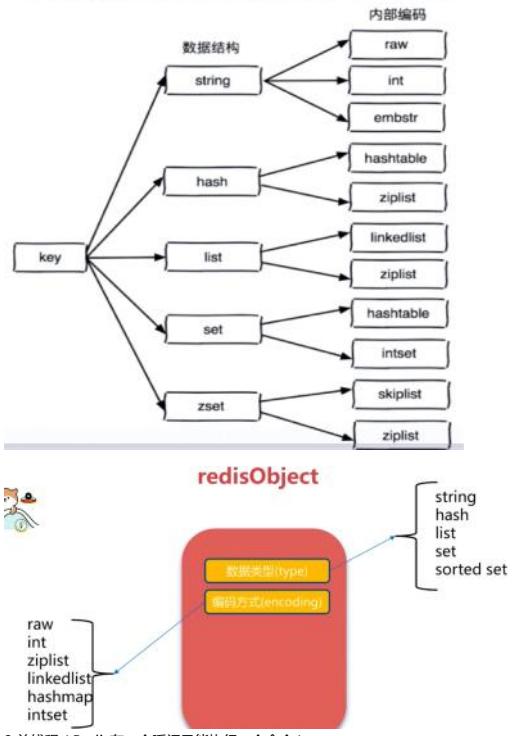
string
hash
list
set
zset
none

时间复杂度

命令	时间复杂度
keys	O(n)
dbsize	O(1)
del	O(1)
exists	O(1)
expire	O(1)
type	O(1)

2. 数据结构和内部编码

数据结构和内部编码



3. 单线程 (Redis在一个瞬间只能执行一个命令)



Redis

(1) 单线程为什么会这么快

① 纯内存访问 (主要原因) :

Redis所有的命令都是存放在内存中，内存响应时间大约是100纳秒，这是Redis达到每秒百万级的重要基础

② 非阻塞I/O

③ 单线程避免了线程切换和竞争产生的消耗

注意：

单线程会有一个问题，对于每个命令的执行时间是有要求的。如果某个命令执行过长，会造成其他命令阻塞，对于Redis这种高性能的服务是致命的，所以Redis是面向快速执行场景的数据库。

单线程

1. 一次只运行一条命令
2. 拒绝长(慢)命令
keys, flushall, flushdb, slow lua script, mult/exec, operate big value(collection)
3. 其实不是单线程
fsync file descriptor
close file descriptor

4. 字符串

①字符串键值结构

字符串键值结构

key	value	
hello	world	{ "product": { "id": "2951", "name": "testing 01", "quantity": 4 } }
counter	1)
bits	1 0 1 1 1 1 0 1	
		※ Up to 512MB

②使用场景

缓存 (大部分使用Redis的场景)、计数器、分布式锁.....

③命令

get, set, del

API	get key #获取key对应的value	o(1)
API	set key value #设置key-value	o(1)
API	del key #删除key-value	o(1)

incr, decr, incrby, decrby

API	incr key #key自增1,如果key不存在,自增后get(key)=1	o(1)
API	decr key #key自减1,如果key不存在,自减后get(key)=-1	o(1)
API	incrby key k #key自增k,如果key不存在,自增后get(key)=k	o(1)
API	decrby key k #key自减k,如果key不存在,自减后get(key)=-k	o(1)

```
[imooc@localhost src]$ ./redis-cli
127.0.0.1:6379> set hello "world"
OK
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> del hello
(integer) 1
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> get counter
(nil)
127.0.0.1:6379> incr counter
(integer) 1
127.0.0.1:6379> incrby counter 99
(integer) 100
127.0.0.1:6379> decr counter
(integer) 99
127.0.0.1:6379> get counter
"99"
```

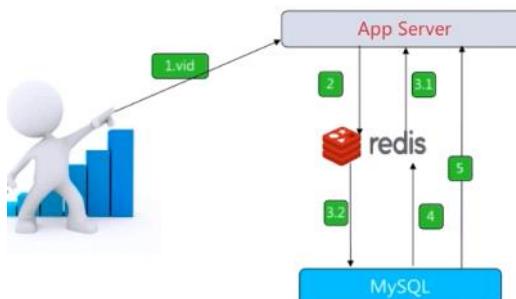
实战1：记录网站每个用户的个人主页的访问量？

incr userid:pageview (单线程 : 无竞争)

实战2：

实现如下功能：

缓存视频的基本信息（数据源在MySQL中）伪代码

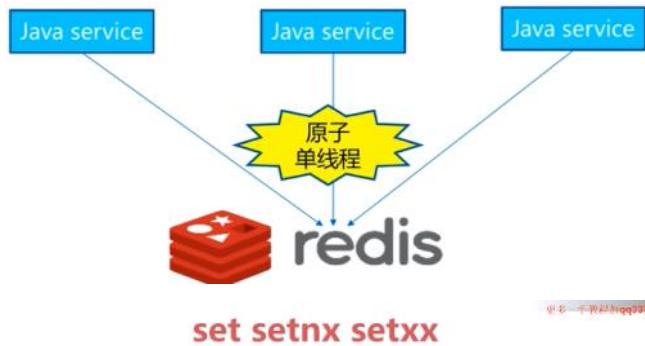


```
public VideoInfo get(long id) {  
    String redisKey = redisPrefix + id;  
    VideoInfo videoInfo = redis.get(redisKey);  
    if (videoInfo == null) {  
        videoInfo = mysql.get(id);  
        if (videoInfo != null) {  
            //序列化  
            redis.set(redisKey, serialize(videoInfo));  
        }  
    }  
    return videoInfo;  
}
```

实战3：

实现如下功能：

分布式id生成器



set setnx setxx

API set key value
#不管key是否存在，都设置 o(1)

API setnx key value
#key不存在，才设置 o(1)

API set key value xx
#key存在，才设置 o(1)

其中setnx操作可以看成是一个add操作，set xx可以看做一个更新操作

set setnx set xx

```
127.0.0.1:6379> exists php  
(integer) 0  
127.0.0.1:6379> set php good  
OK  
127.0.0.1:6379> setnx php bad  
(integer) 0  
127.0.0.1:6379> set php best xx  
OK  
127.0.0.1:6379> get php  
"best"  
127.0.0.1:6379> exists java  
(integer) 0  
127.0.0.1:6379> setnx java best  
(integer) 1  
127.0.0.1:6379> set java easy xx  
OK  
127.0.0.1:6379> get java  
"easy"  
127.0.0.1:6379> exists lua  
(integer) 0  
127.0.0.1:6379> set lua hehe xx  
OK
```

mget mset

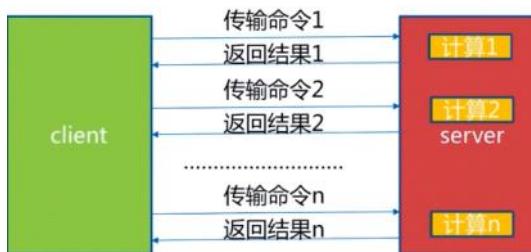
API mget key1 key2 key3...
#批量获取key,原子操作 o(n)

API mset key1 value1 key2 value2 key3 value3
#批量设置key-value o(n)

mget mset

```
127.0.0.1:6379> mset hello world java best php good
OK
127.0.0.1:6379> mget hello java php
1) "world"
2) "best"
3) "good"
```

n次get



$$\text{n次get} = \text{n次网络时间} + \text{n次命令时间}$$

1次mget



$$\text{1次mget} = \text{1次网络时间} + \text{n次命令时间}$$

getset、append、strlen

api getset key newvalue
#set key newvalue并返回旧的value o(1)

api append key value
#将value追加到旧的value o(1)

api strlen key
#返回字符串的长度(注意中文) o(1)

```
127.0.0.1:6379> set hello world
OK
127.0.0.1:6379> getset hello php
"world"
127.0.0.1:6379> append hello ",java"
(integer) 8
127.0.0.1:6379> get hello
"php,java"
127.0.0.1:6379> strlen hello
(integer) 8
127.0.0.1:6379> set hello "足球"
OK
127.0.0.1:6379> strlen hello
(integer) 6
```

incrbyfloat getrange setrange

api incrbyfloat key 3.5
#增加key对应的值3.5 o(1)

api getrange key start end
#获取字符串指定下标所有的值 o(1)

api setrange key index value
#设置指定下标所有对应的值 o(1)

incrbyfloat getrange setrange

演示

```
127.0.0.1:6379> incr counter
(integer) 1
127.0.0.1:6379> incrbyfloat counter 1.1
"2.1"
127.0.0.1:6379> get counter
"2.1"
127.0.0.1:6379> set hello javapest
OK
127.0.0.1:6379> getrange hello 0 2
"jav"
127.0.0.1:6379> setrange hello 4 p
(integer) 8
127.0.0.1:6379> get hello
"javapest"
```

3.哈希

key field value

user:1:info	name	Ronaldo
	age	40
	Date	201

哈希键值结构

key field value

user:1:info	name	Ronaldo
	age	40
	Date	201
	viewCounter	50

add a new value

(2) 特点

特点

Mapmap?

Small redis

field不能相同,value可以相同

(3) 重要的API，所有哈希的命令都是以H开头的

①

hget、hset、hdel

API hget key field
#获取hash key对应的field的value

o(1)

API hset key field value
#设置hash key对应field的value

o(1)

API hdel key field
#删除hash key对应field的value

o(1)

hget、hset、hdel

```
127.0.0.1:6379> hset user:1:info age 23
(integer) 1
127.0.0.1:6379> hget user:1:info age
"23"
127.0.0.1:6379> hset user:1:info name ronaldo
(integer) 1
127.0.0.1:6379> hgetall user:1:info
1) "age"
2) "23"
3) "name"
4) "ronaldo"
127.0.0.1:6379> hdel user:1:info age
(integer) 1
127.0.0.1:6379> hgetall user:1:info
1) "name"
2) "ronaldo"
```

②

hexists、hlen

API hexists key field
#判断hash key是否有field

o(1)

API hlen key
#获取hash key field的数量

o(1)

hexists、hlen

演示

```
127.0.0.1:6379> hgetall user:1:info
1) "name"
2) "ronaldo"
3) "age"
4) "23"
127.0.0.1:6379> hexists user:1:info name
(integer) 1
127.0.0.1:6379> hlen user:1:info
(integer) 2
```

③

hmget、hmset

API hmget key field1 field2... fieldN
#批量获取hash key的一批field对应的值

o(n)

API hmset key field1 value1 field2 value2... fieldN valueN
#批量设置hash key的一批field value

o(n)

```
127.0.0.1:6379> hmset user:2:info age 1 name haha page 50
OK
127.0.0.1:6379> hmget user:2:info age name
1) "1"
2) "haha"
127.0.0.1:6379>
```

hgetall、hvals、hkeys

API hgetall key
#返回hash key对应所有的field和value

O(n)

API hvals key
#返回hash key对应所有field的value

O(n)

API hkeys key
#返回hash key对应所有field

O(n)

hgetall、hvals、hkeys

示例
127.0.0.1:6379> hgetall user:2:info
1) "age"
2) "30"
3) "name"
4) "kaka"
5) "page"
6) "50"
127.0.0.1:6379> hvals user:2:info
1) "30"
2) "kaka"
3) "50"
127.0.0.1:6379> hkeys user:2:info
1) "age"
2) "name"
3) "page"

当哈希里面存在了很多属性的时候要小心hgetall

(4) 实战

①

实现如下功能：

记录网站每个用户个人主页的访问量？

hincrby user:1:info pageview count

②

```
public VideoInfo get(long id) {  
    String redisKey = redisPrefix + id;  
    Map<String, String> hashMap = redis.hgetAll(redisKey);  
    VideoInfo videoInfo = transferMapToVideo(hashMap);  
    if (videoInfo == null) {  
        videoInfo = mysql.get(id);  
        if (videoInfo != null) {  
            redis.hmset(redisKey, transferVideoToMap(videoInfo));  
        }  
    }  
    return videoInfo;  
}
```

(5) 对比

相似的API

get
set setnx
del
incr incrby decr decrby

hget
hset hsetnx
hdel
hincrby

(6) 用户信息缓存的方法

①

用户信息(string实现)

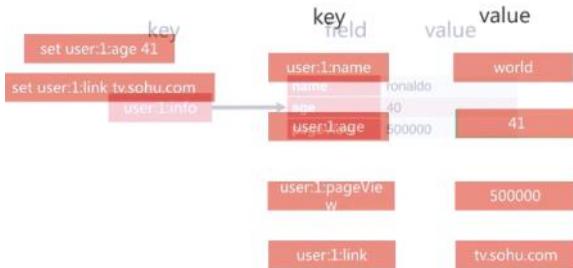
key value(serializable,json,xml,protobuf)

user:1

```
{  
    "id": 1,  
    "name": "ronaldo",  
    "age": 40,  
    "pageView": 500000  
}
```

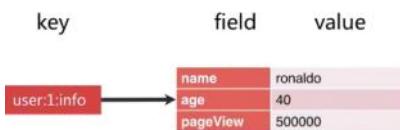
②

用户信息(string实现-v2)



②

用户信息(hash)



41

hset user:1:info age 41

3种方案比较

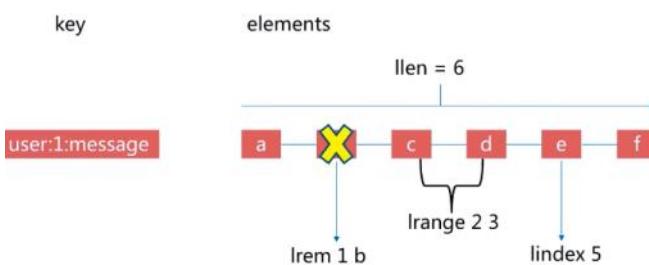
命令	优点	缺点
string v1	编程简单 可能节约内存	1. 序列化开销 2. 设置属性要操作整个数据。
string v2	直观 可以部分更新	1. 内存占用较大 2. key较为分散
hash	直观 节省空间 可以部分更新	1. 编程稍微复杂 2. ttl不好控制

ttl是过期时间

4.列表 (lists)

(1) 数据结构

列表结构



(2) 特点

- ①列表中的元素是有序的，这就意味着可以通过索引下标获取某个元素或者某个范围的元素列表
- ②列表中的元素是可以重复的

③列表中的元素是可以从左右两边进行插入删除的

(3) 重要的 "API"

①增

rpush

API rpush key value1 value2 ...valueN
#从列表右端插入值(1-N个)

O(1~n)



rpush listkey c b a

lpush

API lpush key value1 value2 ...valueN
#从列表左端插入值(1-N个)

O(1~n)



lpush listkey c b a

②插

linsert

API linsert key before|after value newValue
#在list指定的值前|后插入newValue

O(n)



linsert listkey before b java

③删

linsert listkey after b php

lpop

API lpop key
#从列表左侧弹出一个item

O(1)



lpop listkey

rpop

API rpop key
#从列表右侧弹出一个item

O(1)



rpop listkey

lrem

API lrem key count value
#根据count值，从列表中删除所有value相等的项
(1) count>0，从左到右，删除最多count个value相等的项
(2) count<0，从右到左，删除最多Math.abs(count)个value相等的项
(3) count=0，删除所有value相等的项

O(n)



lrem listkey 0 a

lrem

API lrem key count value
#根据count值，从列表中删除所有value相等的项
(1) count>0，从左到右，删除最多count个value相等的项
(2) count<0，从右到左，删除最多Math.abs(count)个value相等的项
(3) count=0，删除所有value相等的项

O(n)



lrem listkey 0 a

lrem listkey -1 c

Itrim

API Itrim key start end
#按照索引范围修剪列表

O(n)



Itrim listkey 1 4

Itrim

API Itrim key start end
#按照索引范围修剪列表

O(n)



Itrim listkey 1 4

Itrim listkey 0 2

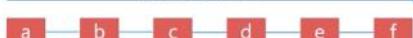
④查

Irange

API Irange key start end (包含end)
#获取列表指定索引范围所有item

O(n)

索引从左 : 0~5

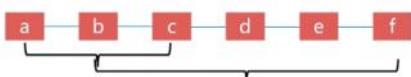


索引从右 : -1~-6

Irange

API Irange key start end (包含end)
#获取列表指定索引范围所有item

O(n)



Irange listkey 0 2

Irange listkey 1 -1

lindex

API lindex key index
#获取列表指定索引的item

O(n)



lindex listkey 0

lindex listkey -1

llen

API llen key
#获取列表长度

O(1)



当前的长度是6

⑤改

Iset

API lset key index newValue
#设置列表指定索引值为newValue

$O(n)$



lset listkey 2 java

就是把c换成了java

blpop brpop

api blpop key timeout
#lpop阻塞版本, timeout是阻塞超时时间, timeout=0为永远不阻塞

$O(1)$

api brpop key timeout
#rpop阻塞版本, timeout是阻塞超时时间, timeout=0为永远不阻塞

$O(1)$

TIPS

1. LRUSH + LPOP = Stack
2. LPUSH + RPOP = Queue
3. LPUSH + LTRIM = Capped Collection
4. LPUSH + BRPOP = Message Queue

```
127.0.0.1:6379> rpush mylist a b c
(integer) 3
127.0.0.1:6379> lrange listkey 0 -1
(empty list or set)
127.0.0.1:6379> lrange mylist 0 -1
1) "a"
2) "b"
3) "c"
127.0.0.1:6379> lpush mylist o
(integer) 4
127.0.0.1:6379> lrange mylist 0 -1
1) "o"
2) "a"
3) "b"
4) "c"
127.0.0.1:6379> rpop mylist
"c"
127.0.0.1:6379> lrange mylist 0 -1
1) "o"
2) "a"
3) "b"
```

(4) 实战 (微博)





5. 集合 (Set)

集合结构

key values



Redis除了支持集合内部的增删改之外，同时还支持集合去交集

(2) 特点

特点

无序

无重复

集合间操作

(3) 集合内的API，都是以s开头

sadd srem

API sadd key element
#向集合key添加element(如果element已经存在，添加失败)

o(1)

API srem key element
#将集合key中的element移除掉

scard sismember srandmember smembers

user:1:follow



```
scard user:1:follow = 4 #计算集合大小
sismember user:1:follow it = 1(存在) #判断it是否在集合中
srandmember user:1:follow count= his #从集合中随机挑count个元素
spop user:1:follow = sports #从集合中随机弹出一个元素
smembers user:1:follow = music his sports it #获取集合所有元素
```

smembers

无序

小心使用

srandmember 和 spop

spop从集合弹出

srandmember不会破坏集合

spop是从集合中随机弹出元素，而且每次只能弹出一个，执行完spop命令后元素会从集合中删除；srandmember是从返回指定个数的元素，弹出的元素不会从集合中删除
实战：抽奖系统

实战-抽奖系统

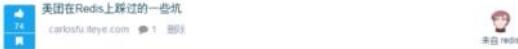


can be spop

实战-Like、赞、踩

推荐两篇redis的文章：docker快速构建redis cluster的环境：[\[转\]](#) [阅读链接](#)；redis hash 和set元素为空时，会被自动删除，使用时请注意导致的属性被清理，这个估计发生概率很高的。如果ttl设置不合理，会导致类似‘内存泄露’的事情发生。[\[转\]](#) [阅读链接](#)。
2015-12-6 11:15 来自 网易 www.163.com

收藏 18 转发 18 评论 4 踩 5

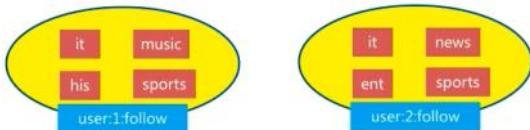


实战-标签(tag)



集合间的API

sdiff sinter sunion



sdiff user:1:follow user:2:follow = music his #差集
sinter user:1:follow user:2:follow = it sports #交集
sunion user:1:follow user:2:follow = it music his sports news ent #并集
sdiff|sinter|sunion + store destkey .. #将差集、交集、并集结果保存在destkey中
集合间的简单实战

共同关注？



使用集合类型的应用场景主要分为以下几种：(1) 标签 (2) 生成随机数，比如抽奖 (3) 社交需求

TIPS

SADD = Tagging

SPOP/SRANDOMMEMBER = Random item

SADD + SINTER = Social Graph

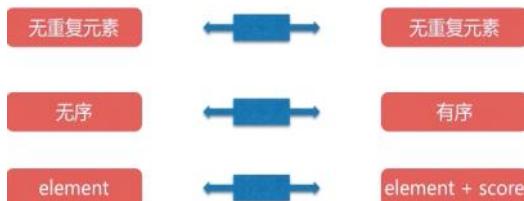
6.有序集合（是一个集合，但是集合中每一个元素是有序的）

①有序集合的结构

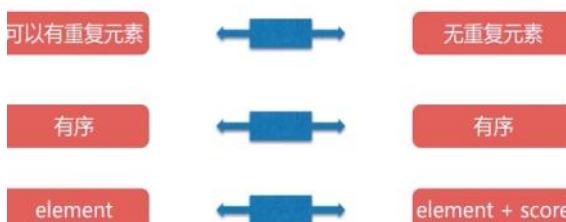
有序集合结构

key	score	value
user:ranking	1	kris
	91	mike
	200	frank
	220	chris
	250	martin
	251	tom

集合 Vs 有序集合



列表 Vs 有序集合



(2) 重要的API，有序集合都是以z开头的命令

zadd

API zadd key score element(可以是多对)
#添加score和element
 $O(\log N)$

有序集合里面元素是不能重复的，但是score是可以重复的

user:1:ranking	
	1 kris
	91 mike
	200 frank
	220 chris

zadd user:1:ranking 225 tom

zrem

[API] zrem key element(可以是多个)	#删除元素
	o(1)
user:1:ranking	
	1 kris
	91 mike
	200 frank
	220 chris
	225 tom

zrem user:1:ranking 225 tom

zscore

[API] zscore key element	#返回元素的分数
	o(1)
user:1:ranking	
	1 kris
	91 mike
	200 frank
	220 chris
	225 tom

zscore user:1:ranking tom

zincrby

[API] zincrby key increScore element	#增加或减少元素的分数
	o(1)
user:1:ranking	
	1 kris
	91 mike
	200 frank
	220 chris
	225 tom

zincrby user:1:ranking 9 mike

zcard

[API] zcard key	#返回元素的总个数
	o(1)
user:1:ranking	
	1 kris
	91 mike
	200 frank
	220 chris
	225 tom

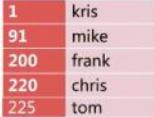
zcard user:1:ranking

zrange

[API] zrange key start end [WITHSCORES]	#返回指定索引范围内的升序元素[分值]
	o(log(n)+m)
user:1:ranking	
	1 kris
	91 mike
	200 frank
	220 chris
	225 tom

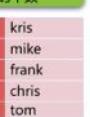
zrange user:1:ranking 1 3 withscores

zrangebyscore

API	<code>zrangebyscore key minScore maxScore [WITHSCORES]</code>	#返回指定分数范围内的升序元素[分值]	<code>O(log(n)+m)</code>
			

```
zrangebyscore user:1:ranking 90 210 withscores
```

zcount

API	<code>zcount key minScore maxScore</code>	#返回有序集合内在指定分数范围内的个数	<code>O(log(n)+m)</code>
			

```
zcount user:1:ranking 200 221
```

zremrangebyrank

API	<code>zremrangebyrank key start end</code>	#删除指定排名内的升序元素	<code>O(log(n)+m)</code>
			

```
zremrangebyrank user:1:ranking 1 2
```

zremrangebyscore

API	<code>zremrangebyscore key minScore maxScore</code>	#删除指定分数内的升序元素	<code>O(log(n)+m)</code>
			

```
zremrangebyscore user:1:ranking 90 210
```

前面的这些排序都是从低到高，如果要实现从高到低则可以在前面加上rev即可

查缺补漏

- ◆ zrevrank
- ◆ zrevrange
- ◆ zrevrangebyscore
- ◆ zinterstore
- ◆ zunionstore

有序集合总结

操作类型	命令
基本操作	<code>zadd zrem zcard</code> <code>zincrby zscore</code>
范围操作	<code>zrange</code> <code>zrangebyscore</code> <code>zcount</code> <code>zremrangebyrank</code>
集合操作	<code>zunionstore</code> <code>zinterstore</code>

```
127.0.0.1:6379> zadd player:rank 1000 ronaldo 900 messi 800 c-ronaldo 600 kaka
(integer) 4
127.0.0.1:6379> zscore player:ranking kaka
(nil)
127.0.0.1:6379> zscore player:rank kaka
"600"
127.0.0.1:6379> zrank player:rank ronaldo
(integer) 3
127.0.0.1:6379> zrem player:rank messi
(integer) 1
127.0.0.1:6379> zrange player:rank 0 -1 withscores
(error) ERR value is not an integer or out of range
127.0.0.1:6379> zrange player:rank 0 -1 withscores
1) "kaka"
2) "600"
3) "c-ronaldo"
4) "800"
5) "ronaldo"
6) "1000"
```

```
127.0.0.1:6379> zadd playerrank 1000 ronaldo 900 messi 800 c...
ronaldo 600 kaka
(integer) 4
127.0.0.1:6379> zrange playerrank 0 -1
1) "kaka"
2) "ronaldo"
3) "messi"
4) "ronaldo"
127.0.0.1:6379> zcount playerrank 700 901
(integer) 2
127.0.0.1:6379> zrangebyscore playerrank 700 901
1) <"ronaldo">
2) <"messi">
127.0.0.1:6379> zremrangebyrank playerrank 0 1
(integer) 2
127.0.0.1:6379> zrange playerrank 0 -1 withscores
1) "messi"
2) "ronaldo"
3) "900"
4) "1000"
```

实战1：排行榜

实战-排行榜



score: timeStamp saleCount followCount

第三章 Redis的客户端

2018年6月24日 14:22

Jedis就是一个基于Java语言的Java客户端

课程目录

Redis客户端

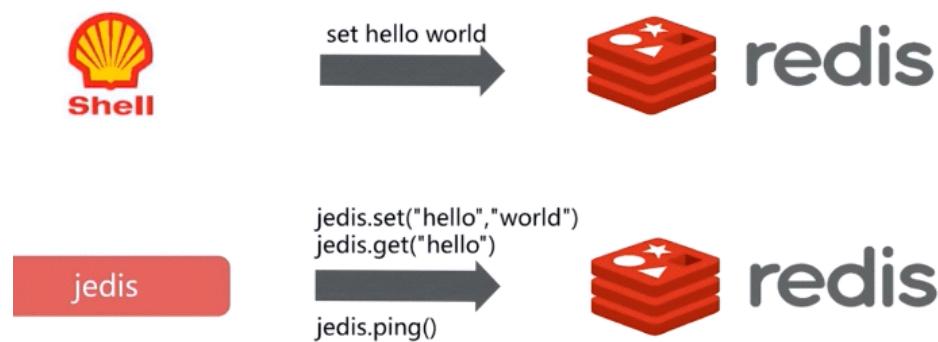
- ◆ Java客户端: Jedis
- ◆ python客户端: redis-py

Redis客户端

Java客户端: Jedis

- ◆ 获取Jedis
- ◆ Jedis连接池使用
- ◆ Jedis基本使用

Jedis是什么



Maven依赖

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.9.0</version>
    <type>jar</type>
    <scope>compile</scope>
</dependency>
```

更多一手教程加qq33799771

Jedis直连

```
# 1. 生成一个Jedis对象，这个对象负责和指定Redis节点进行通信
Jedis jedis = new Jedis("127.0.0.1", 6379);
# 2. jedis执行set操作
jedis.set("hello", "world");
# 3. jedis执行get操作, value="world"
String value = jedis.get("hello");
```

Jedis(String host, int port, int connectionTimeout, int soTimeout)

- host : Redis节点的所在机器的IP
- port : Redis节点的端口
- connectionTimeout : 客户端连接超时
- soTimeout : 客户端读写超时

本质上建立的一次TCP连接

更多一手教程加qq33799771

简单使用

```
// 1.string
// 输出结果：OK
jedis.set("hello", "world");
// 输出结果：world
jedis.get("hello");
// 输出结果：1
jedis.incr("counter");
```

```
// 2.hash
jedis.hset("myhash", "f1", "v1");
jedis.hset("myhash", "f2", "v2");
// 输出结果：{f1=v1, f2=v2}
jedis.hgetAll("myhash");
```

简单使用

演示

```
// 3.list  
jedis.rpush("mylist", "1");  
jedis.rpush("mylist", "2");  
jedis.rpush("mylist", "3");  
// 输出结果 : [1, 2, 3]  
jedis.lrange("mylist", 0, -1);
```

```
// 4.set  
jedis.sadd("myset", "a");  
jedis.sadd("myset", "b");  
jedis.sadd("myset", "a");  
// 输出结果 : [b, a]  
jedis.smembers("myset");
```

简单使用

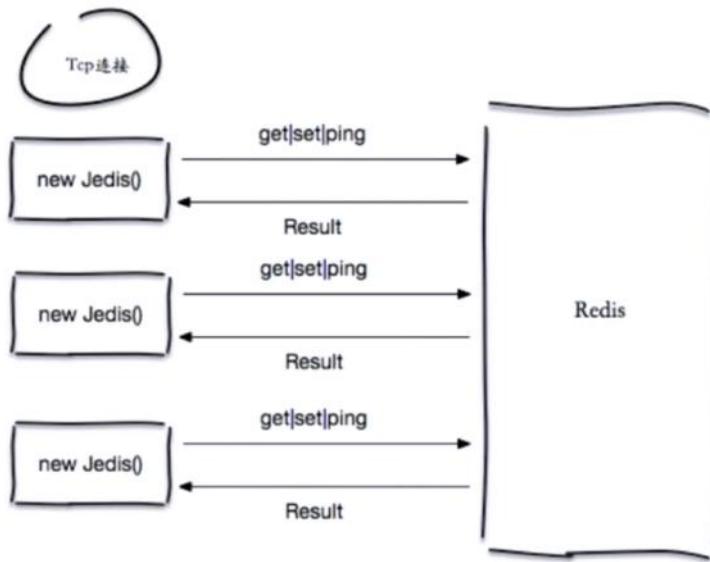
```
// 5.zset  
jedis.zadd("myzset", 99, "tom");  
jedis.zadd("myzset", 66, "peter");  
jedis.zadd("myzset", 33, "james");  
// 输出结果 : [[[{"james"},33.0], [{"peter"},66.0], [{"tom"},99.0]]  
jedis.zrangeWithScores("myzset", 0, -1);
```

Java客户端 : Jedis

Jedis连接池使用

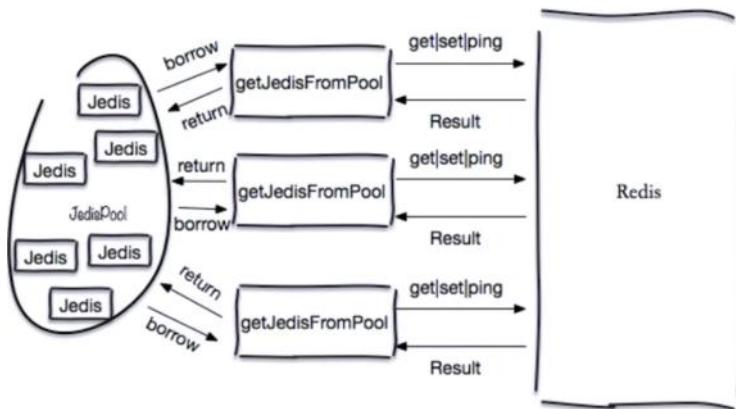
- ◆ Jedis直连
- ◆ Jedis连接池
- ◆ 方案对比
- ◆ JedisPool使用

Jedis直连



- 1.生成Jedis对象
- 2.Jedis执行命令
- 3.返回执行结果
- 4.关闭Jedis连接

Jedis连接池



- 1.从资源池借Jedis对象
- 2.Jedis执行命令
- 3.返回执行结果
- 4.归还Jedis对象给连接池

方案对比

	优点	缺点
直连	<ul style="list-style-type: none">简单方便适用于少量长期连接的场景	<ul style="list-style-type: none">存在每次新建/关闭TCP开销资源无法控制,存在连接泄露的可能Jedis对象线程不安全
连接池	<ul style="list-style-type: none">Jedis预先生成，降低开销使用连接池的形式保护和控制资源的使用	相对于直连，使用相对麻烦，尤其在资源的管理上需要很多参数来保证，一旦规划不合理也会出现问题。

简单使用

演示

```
初始化Jedis连接池，通常来讲JedisPool是单例的。GenericObjectPoolConfig  
poolConfig = new GenericObjectPoolConfig();  
JedisPool jedisPool = new JedisPool(poolConfig, "127.0.0.1", 6379);
```

```
Jedis jedis = null;  
try {  
    // 1. 从连接池获取jedis对象  
    jedis = jedisPool.getResource();  
    // 2. 执行操作  
    jedis.set("hello" , "world" );  
} catch (Exception e) {  
    e.printStackTrace();  
} finally {  
    if (jedis != null)  
        // 如果使用JedisPool.close操作不是关闭连接，代表归还连接池  
        jedis.close();
```

第八章 Redis cluster

2018年6月27日 20:04

一、呼唤集群

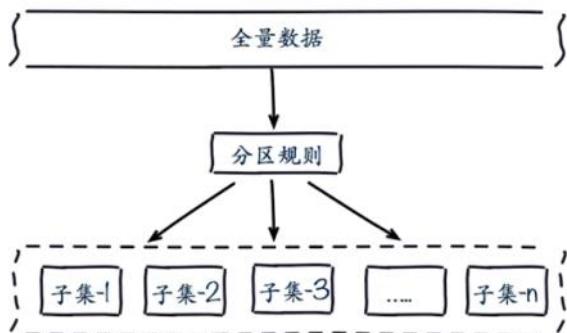
Redis的性能每秒可以实现上万的命令，但是如果有些业务需要百万级别的QPS怎么办（并发量）？

数据量的需求：当业务需要量更大的时候，需要分布式，就是将数据放在不同的地方。如果机器的内存是16~25G，业务需要500G该怎么办？

早期的时候解决的方法是使用更强悍的机器，正确的解决的方法是加机器，就是实现分片。总结起来，集群是规模化的需求：（1）并发量（OPS或者QPS）（2）数据量：“大数据”

二、数据分布

分布式数据库-数据分区



常用的两种分区方式：（1）顺序分区（2）哈希分区

数据分布对比

分布方式	特点	典型产品
哈希分布	数据分散度高 键值分布业务无关 无法顺序访问 支持批量操作	一致性哈希 Memcache Redis Cluster 其他缓存产品
顺序分布	数据分散度易倾斜 键值业务相关 可顺序访问 支持批量操作	BigTable HBase

哈希分布的典型产品是缓存的产品，其中Redis cluster是哈希分布的，采用的分区方式是虚拟槽分区

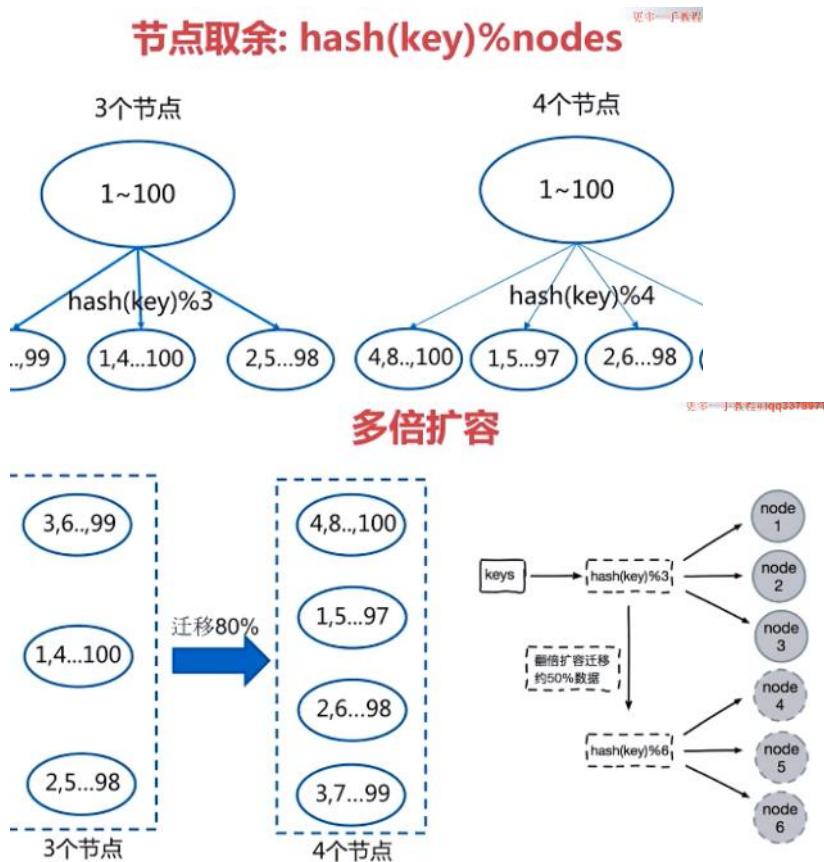
哈希分布

◆ 节点取余分区

哈希分布

- ◆ 节点取余分区
- ◆ 一致性哈希分区
- ◆ 虚拟槽分区

(1) 节点取余



缺点：节点取余是很早的一种使用方式，如果业务对数据库压力不大，不靠缓存支撑的时候可以使用，但是这种方式确实是一种很古老的，并且不建议使用的一种方式

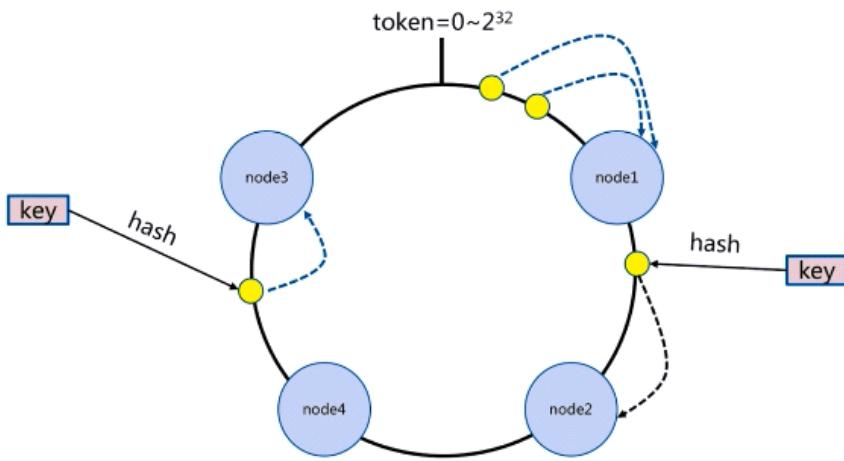
节点取余

- ◆ 客户端分片：哈希 + 取余
- ◆ 节点伸缩：数据节点关系变化，导致数据迁移
- ◆ 迁移数量和添加节点数量有关：建议翻倍扩容

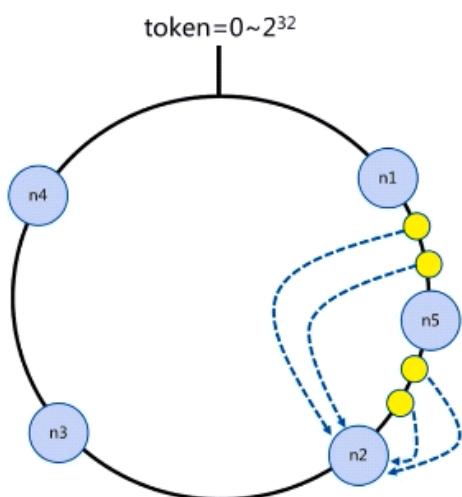
(2) 一致性哈希 (在memcache上使用的是比较广泛的)

顺时针的寻找最近的，一般是在节点比较多的时候采用的缓存方式

一致性哈希



一致性哈希-扩容



客户端分片：在客户端分好哈希值，然后再去取值

一致性哈希

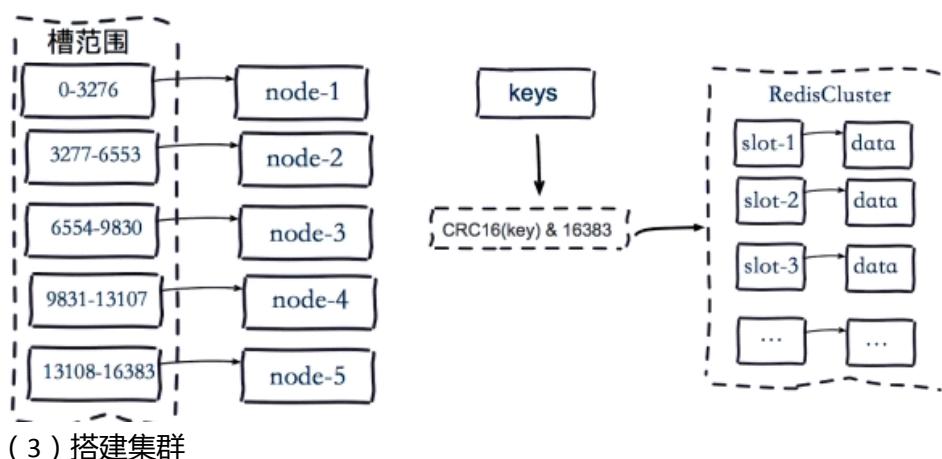
- ◆ 客户端分片：哈希 + 顺时针（优化取余）
- ◆ 节点伸缩：只影响邻近节点，但是还是有数据迁移
- ◆ 翻倍伸缩：保证最小迁移数据和负载均衡
所谓负载均衡就是如果在1和2之间添加，那个3和4相对1和2就会不平衡，所以推荐使用翻倍扩容的方式，比如原来是4个结点，扩容之后变成8个结点。

(3) 虚拟槽分区 (Redis Cluster采用的分区方式)

虚拟槽分区

- ◆ 预设虚拟槽：每个槽映射一个数据子集，一般比节点数大
- ◆ 良好的哈希函数：例如CRC16
- ◆ 服务端管理节点、槽、数据：例如Redis Cluster

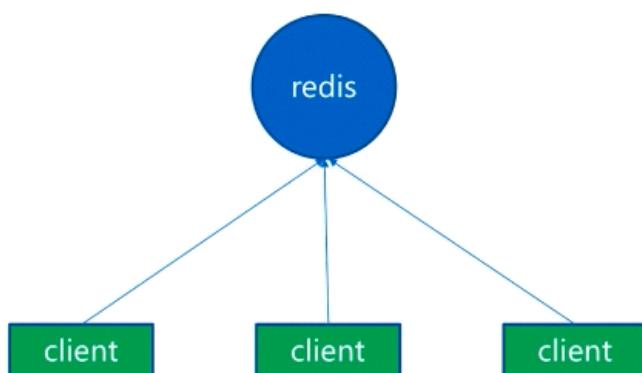
虚拟槽分配



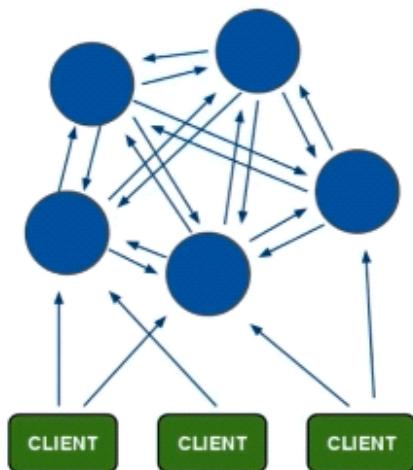
(3) 搭建集群

1. 基本架构

单机架构



分布式架构



(1) 彼此之间是进行通信的 (2) 每个节点都可以进行读写操作，这种情况下是需要智能客户端

重点：Redis cluster的基本架构

Redis Cluster 架构

节点

meet

指派槽

复制

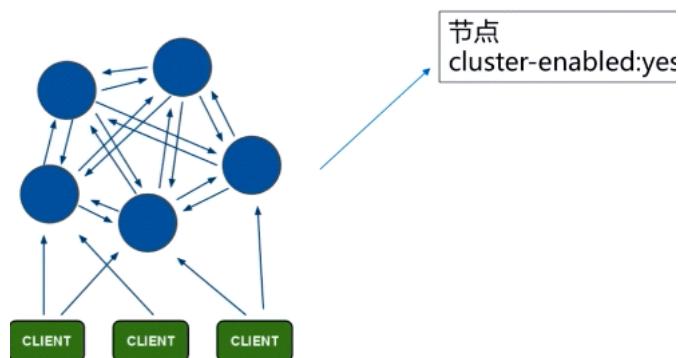
0

复制：主节点挂了，从节点会顶上来

2.Redis Cluster

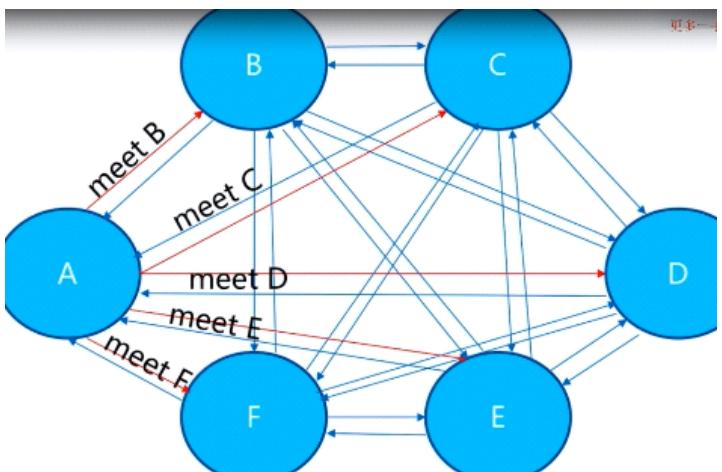
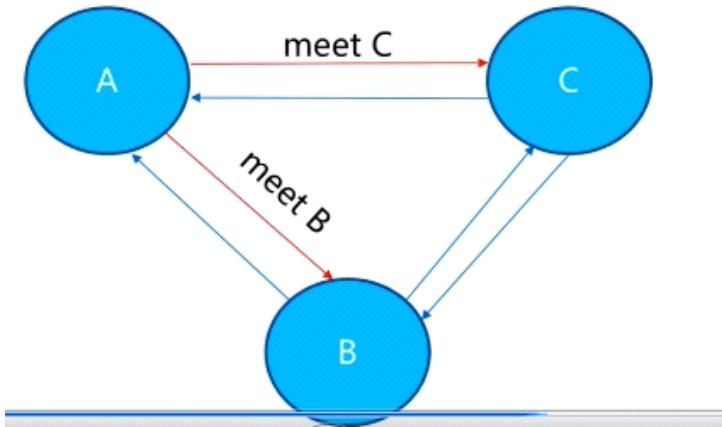
①yes的话就是集群模式来启动

节点

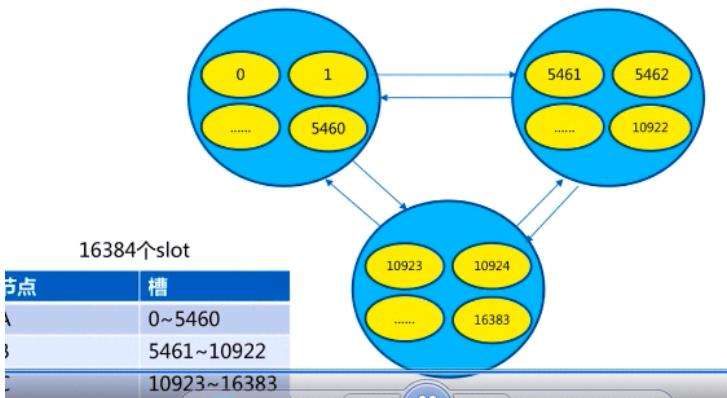


②所有节点共享消息

meet



指派槽



3.Redis Cluster的特性

- (1) 可以实现主从复制
- (2) 高可用，即当主节点挂了的时候从节点可以实现一个晋升，进而实现高可用
- (3) 是分片的

4.两种安装方式

两种安装

原生命令安装

官方工具安装

(1) 原生命令安装

- ①配置开启节点
- ②meet：实现节点之间的相互通信（节点之间的握手）
- ③指派曹
- ④主从

配置开启Redis

```
port ${port}
daemonize yes
dir "/opt/redis/redis/data/"
dbfilename "dump-${port}.rdb"
logfile "${port}.log"
cluster-enabled yes
cluster-config-file nodes-${port}.conf
```

配置开启Redis(续)

redis-server redis-7000.conf

redis-server redis-7001.conf

redis-server redis-7002.conf

redis-server redis-7003.conf

redis-server redis-7004.conf

redis-server redis-7005.conf

接下来是meet操作

cluster meet ip port

```
redis-cli -h 127.0.0.1 -p 7000 cluster meet 127.0.0.1 7001  
redis-cli -h 127.0.0.1 -p 7000 cluster meet 127.0.0.1 7002  
redis-cli -h 127.0.0.1 -p 7000 cluster meet 127.0.0.1 7003  
redis-cli -h 127.0.0.1 -p 7000 cluster meet 127.0.0.1 7004  
redis-cli -h 127.0.0.1 -p 7000 cluster meet 127.0.0.1 7005
```

Cluster节点主要配置

```
cluster-enabled yes  
cluster-node-timeout 15000  
cluster-config-file "nodes.conf"  
cluster-require-full-coverage yes
```

```
port 7000  
daemonize yes  
dir "/opt/soft/redis/data"  
logfile "7000.log"  
dbfilename "dump-7000.rdb"  
cluster-enabled yes  
cluster-config-file nodes-7000.conf  
cluster-require-full-coverage no
```

```
→ config sed 's/7000/7001/g' redis-7000.conf > redis-7001.conf  
→ config sed 's/7000/7002/g' redis-7000.conf > redis-7002.conf  
→ config sed 's/7000/7003/g' redis-7000.conf > redis-7003.conf  
→ config sed 's/7000/7004/g' redis-7000.conf > redis-7004.conf  
→ config sed 's/7000/7005/g' redis-7000.conf > redis-7005.conf  
→ config redis-server redis-7000.conf  
→ config ps -ef | grep redis  
 501 23831      1  0  9:00下午 ??          0:00.01 redis-server *:7000 [clu  
ster]  
 501 23837 23495  0  9:00下午 ttys000    0:00.00 grep --color=auto --excl  
ude-dir=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --excl  
ude-dir=.svn redis
```

```
→ config redis-server redis-7003.conf  
→ config redis-server redis-7004.conf  
→ config redis-server redis-7005.conf  
→ config ps -ef | grep redis-server  
 501 23831      1  0  9:00下午 ??          0:00.03 redis-server *:7000 [clu  
ster]  
 501 23850      1  0  9:01下午 ??          0:00.02 redis-server *:7001 [clu  
ster]  
 501 23856      1  0  9:01下午 ??          0:00.02 redis-server *:7002 [clu  
ster]  
 501 23863      1  0  9:01下午 ??          0:00.02 redis-server *:7003 [clu  
ster]  
 501 23869      1  0  9:01下午 ??          0:00.01 redis-server *:7004 [clu  
ster]  
 501 23875      1  0  9:01下午 ??          0:00.01 redis-server *:7005 [clu  
ster]  
 501 23882 23495  0  9:01下午 ttys000    0:00.00 grep --color=auto --excl  
ude-dir=.bzr --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclu  
de-dir=.svn redis-server
```

```
→ data redis-cli -p 7000 cluster nodes  
560598cc5f6b13663ee7aaa9ff403e799e7d4918 :7000 myself,master - 0 0 0 connec  
ted  
→ data 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
```

最后意向一般设置成no

接着是分配槽

更多一手教程加qq3379977696

分配槽

cluster addslots slot [slot ...]

redis-cli -h 127.0.0.1 -p 7000 cluster addslots {0...5461}

redis-cli -h 127.0.0.1 -p 7001 cluster addslots {5462...10922}

redis-cli -h 127.0.0.1 -p 7002 cluster addslots {10923...16383}

最后设置主从关系

设置主从

cluster replicate node-id

```
redis-cli -h 127.0.0.1 -p 7003 cluster replicate ${node-id-7000}
```

```
redis-cli -h 127.0.0.1 -p 7004 cluster replicate ${node-id-7001}
```

```
redis-cli -h 127.0.0.1 -p 7005 cluster replicate ${node-id-7002}
```

这样一个集群就建立好了，是一种高可用的集群，3主3从

5.Ruby环境准备

Ruby环境准备

下载、编译、安装Ruby

安装rubygem redis

安装redis-trib.rb

下载、编译、安装Ruby

(1) 下载ruby

```
wget https://cache.ruby-lang.org/pub/ruby/2.3/ruby-2.3.1.tar.gz
```

(2) 安装ruby

```
tar -xvf ruby-2.3.1.tar.gz  
./configure --prefix=/usr/local/ruby  
make  
make install  
cd /usr/local/ruby  
cp bin/ruby /usr/local/bin  
cp bin/gem /usr/local/bin
```

安装rubygem redis

```
wget http://rubygems.org/downloads/redis-3.3.0.gem
```

```
gem install -l redis-3.3.0.gem
```

```
gem list --check redis gem
```

安装redis-trib.rb

```
cp ${REDIS_HOME}/src/redis-trib.rb /usr/local/bin
```

第四章 Redis瑞士军刀

2018年6月27日 21:58

课程目录

瑞士军刀Redis

- ◆ 慢查询
- ◆ Bitmap
- ◆ pipeline
- ◆ HyperLogLog
- ◆ 发布订阅

一、慢查询

瑞士军刀Redis

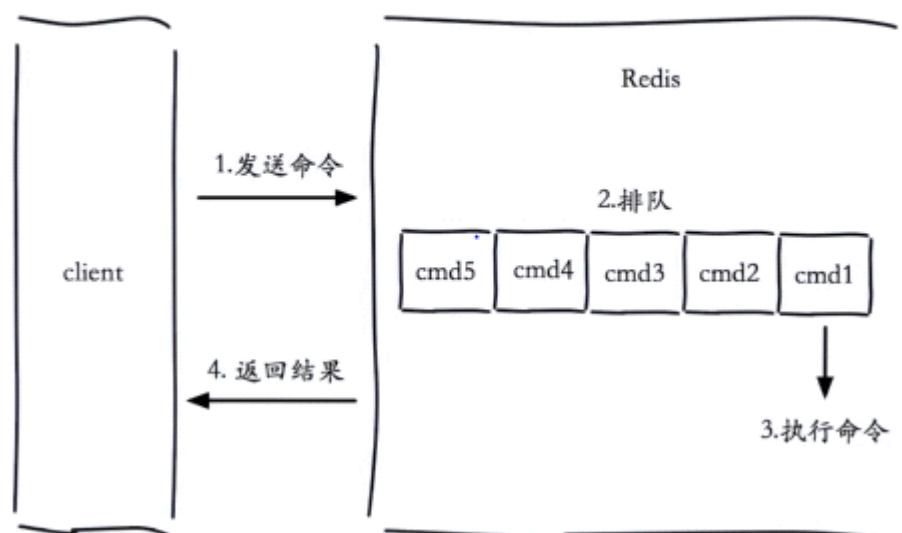
慢查询

- ◆ 生命周期
- ◆ 两个配置
- ◆ 三个命令
- ◆ 运维经验

Redis是一个单线程的

(1) 首先慢查询是发生在第三个阶段 (2) 客户端超时不一定能引起慢查询；但是慢查询是客户端超时的一个因素

生命周期

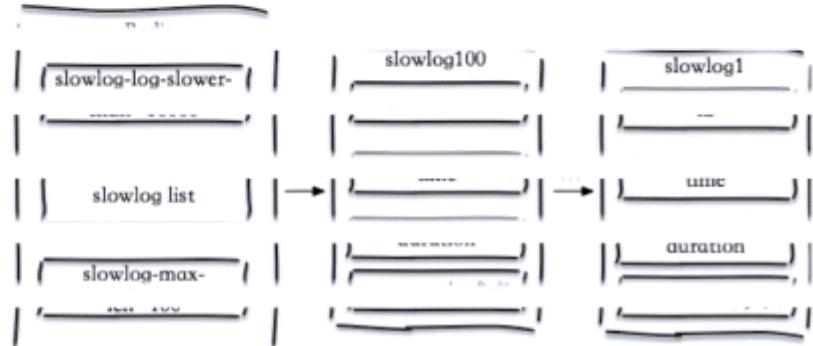


两个配置- slowlog-max-len

1. 先进先出队列

2. 固定长度

3. 保存在内存内



两个配置- slowlog-log-slower-than

1. 慢查询阈值(单位：微秒)

2. slowlog-log-slower-than=0 , 记录所有命令

3. slowlog-log-slower-than<0 , 不记录任何命令。

配置方法

1. 默认值

- config get slowlog-max-len = 128
- config get slowlog-log-slower-than = 10000

2. 修改配置文件重启

3. 动态配置

- config set slowlog-max-len 1000

运维经验

1. slowlog-max-len不要设置过大，默认10ms，通常设置1ms
2. slowlog-log-slower-than不要设置过小，通常设置1000左右。
3. 理解命令生命周期。
4. 定期持久化慢查询。

数据倾斜

2018年7月8日 20:47

节点和槽分配不均

- ◆ redis-trib.rb info ip:port查看节点、槽、键值分布
- ◆ redis-trib.rb rebalance ip:port进行均衡(谨慎使用)

原因主要有四个

不同槽对应键值数量差异较大

- ◆ CRC16正常情况下比较均匀。
- ◆ 可能存在hash_tag
- ◆ cluster countkeysinslot {slot}获取槽对应键值个数

包含bigkey

- ◆ bigkey：例如大字符串、几百万的元素的hash、set等
- ◆ 从节点：redis-cli --bigkeys
- ◆ 优化：优化数据结构。

内存相关配置不一致

- ◆ hash-max-ziplist-value、set-max-intset-entries等
- ◆ 优化：定期“检查”配置一致性

请求倾斜

2018年7月8日 20:55

请求倾斜

- ◆ 热点key：重要的key或者bigkey
- ◆ 优化：
 - 避免bigkey
 - 热键不要用hash_tag
 - 当一致性不高时，可以用本地缓存 + MQ

集群读写分离

2018年7月8日 20:57

集群读写分离

◆ 只读连接：集群模式的从节点不接受任何读写请求。

- 重定向到负责槽的主节点
- readonly命令可以读：连接级别命令

◆ 读写分离：更加复杂

- 同样的问题：复制延迟、读取过期数据、从节点故障
- 修改客户端：cluster slaves {nodeId}

```
[root@localhost config]# redis-cli -c -p 7000
127.0.0.1:7000> set hello world
OK
127.0.0.1:7000> exit
[root@localhost config]# redis-cli -c -p 7003
127.0.0.1:7003> get hello
-> Redirected to slot [866] located at 127.0.0.1:7000
"world"
127.0.0.1:7000> get hello
"world"
127.0.0.1:7000>
[root@localhost config]# redis-cli -c -p 7003
127.0.0.1:7003> readonly
OK
127.0.0.1:7003> get hello
"world"
127.0.0.1:7003> ■
```

读写分离在集群模式下不建议使用，使用读写分离可能会要求重新下载一个客户端

数据迁移

2018年7月8日 21:03

离线/在线迁移

◆ 官方迁移工具 : redis-trib.rb import

- 只能从单机迁移到集群
- 不支持在线迁移 : source需要停写
- 不支持断点续传
- 单线程迁移 : 影响速度

◆ 在线迁移 :

- 唯品会redis-migrate-tool
- 豌豆荚 : redis-port

集群和单机

2018年7月8日 21:09

更多一手教程加qq337097

集群限制

- ◆ key批量操作支持有限：例如mget、mset必须在一个slot
- ◆ Key事务和Lua支持有限：操作的key必须在一个节点
- ◆ key是数据分区的最小粒度：不支持bigkey分区
- ◆ 不支持多个数据库：集群模式下只有一个db 0
- ◆ 复制只支持一层：不支持树形复制结构



1. Redis Cluster：满足容量和性能的扩展性，很多业务“不需要”。

- 大多数时客户端性能会“降低”。
- 命令无法跨节点使用：mget、keys、scan、flush、sinter等。
- Lua和事务无法跨节点使用。
- 客户端维护更复杂：SDK和应用本身消耗(例如更多的连接池)。

2. 很多场景Redis Sentinel已经足够好。

集群总结

2018年7月8日 21:12

集群总结

更多一手教程加qq337997761

- ◆ Redis cluster数据分区规则采用虚拟槽方式(16384个槽)，每个节点负责一部分槽和相关数据，实现数据和请求的负载均衡。
- ◆ 搭建集群划分四个步骤：准备节点、节点握手、分配槽、复制。
redis-trib.rb工具用于快速搭建集群。
- ◆ 集群伸缩通过在节点之间移动槽和相关数据实现。
 - 扩容时根据槽迁移计划把槽从源节点迁移到新节点。
 - 收缩时如果下线的节点有负责的槽需要迁移到其它节点，再通过cluster forget命令让集群内所有节点忘记被下线节点。

集群总结

更多一手教程加qq337997761

- ◆ 使用smart客户端操作集群达到通信效率最大化，客户端内部负责计算维护键->槽->节点的映射，用于快速定位到目标节点。
- ◆ 集群自动故障转移过程分为故障发现和节点恢复。节点下线分为主观下线和客观下线，当超过半数主节点认为故障节点为主观下线时标记它为客观下线状态。从节点负责对客观下线的主节点触发故障恢复流程，保证集群的可用性。
- ◆ 开发运维常见问题包括：超大规模集群带宽消耗，pub/sub广播问题，集群倾斜问题，单机和集群对比等