# Redis

## Redis的简介

Redis是一个Key-Value的储存系统。其作用有数据的缓存，数据库，还有消息中间件.和Mamcache类似，不过Redis支持的Value类型更多，有string(字符串,list(链表,set(集合,hash哈希,zset(有序集合.Redis支持每秒10万数据的访问

## Redis的持久化策略

Redis数据时储存在内存中的，服务器宕机或者断电理应会导致缓存清空，但是实际却没有，因为Redis会定期根据配置文件的配置持久换数据，恢复数据也是根据配置文件中指定的持久化文件恢复的

### 2.1dump.rdb模式

特点：

* 是redis默认的配置的模式
* 因为是在周期内恢复数据的，可能会导致数据的丢失(特别重要的数据不能使用此模式
* 效率高
* 采用内存快照的方式进行持久化的，所以文件大小可控
* 是线程同步的，所以会阻塞

### 2.2RDB模式下的配置

1. 内存是1K=1000byte，而1KB是等于1024Byte,详情如下

# 1k => 1000 bytes

# 1kb => 1024 bytes

# 1m => 1000000 bytes

# 1mb => 1024\*1024 bytes

# 1g => 1000000000 bytes

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

1. 持久化命令

在客户端运行save命令，会马上执行一次持久化操作，该操作是Redis的默认操作，所以是线程同步的，会造成阻塞

在客户端执行bgsave命令，会在后台执行持久化操作，但不一定会马上执行，就跟Java中的GC一样，该操作时线程异步的，所以不会阻塞

1. RDB持久化策略

save 900 1 在900秒内执行一次set操作时持久化一次.

save 300 10 在300秒内执行10此set操作.则持久化一次.

save 60 10000 在60秒内,执行10000此持久化操作.则set一次.

Save 1 1 在1秒内执行一次set操作时，则持久化一次，但是会降低效率

之后在启动服务器时加载配置文件。

Redis-server redis.conf

这种通过服务器配置文件触发RDB的方式，与bgsave命令类似，达到触发条件时，会forks一个子进程进行数据同步，不过最好不要通过这方式来触发RDB持久化，因为设置触发的时间太短，则容易频繁写入rdb文件，影响服务器性能，时间设置太长则会造成数据丢失。

#### rdb文件

前面介绍了三种让服务器生成rdb文件的方式，无论是由主进程生成还是子进程来生成，其过程如下：

1. 生成临时rdb文件，并写入数据
2. 完成数据写入，用临时文件代替正式rdb文件
3. 删除原来的rdb文件

RDB默认生成的文件名为dump.rdb，当然，我可以通过配置文件进行更加详细配置，比如在单机下启动多个redis服务器进程时，可以通过端口号配置不同的rdb名称，如下所示：

#是否压缩rdbwe文件

Rdbcompression yes

#rdb文件的名称

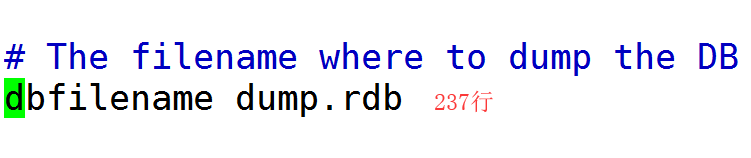
Dbfilename redis-6379.rdb

#rdb文件保存的目录

Dir ~/redis/

1. RDB持久化操作的的文件名称配置

Dbfilename dump.rdb



#### RDB的几个优点

1. 与AOF方式相比，通过rdb文件恢复数据比较快。
2. Rdb文件非常紧凑，适合于数据备份
3. 通过RDB进行数据备份，由于使用子进程生成、所以对Redis服务器性能影响较小

#### RDB的几个缺点

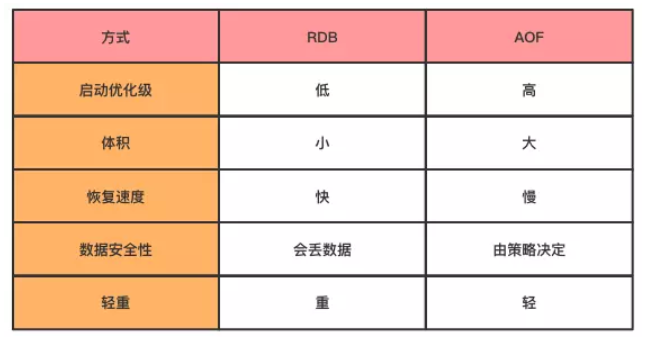
1. 如果服务器宕机的话，采用RDB的方式会造成某个时段内数据的丢失，比如我们设置10分钟同步一次或5分钟达到1000次写入就同步一次，那么如果还没达到触发条件服务器就死机了，那么这个时间段的数据会丢失。
2. 使用save命令会造成服务器阻塞，直接数据同步完成才能接收后续请求
3. 使用bgsave命令在forks子进程时，如果数据量太大，forks的过程也会发生阻塞，另外forks子进程会消耗内存

## 2.3 appendOnly.aof模式

该模式能保证数据的完整性，可实现实时数据的持久化，但是效率低于RDB模式

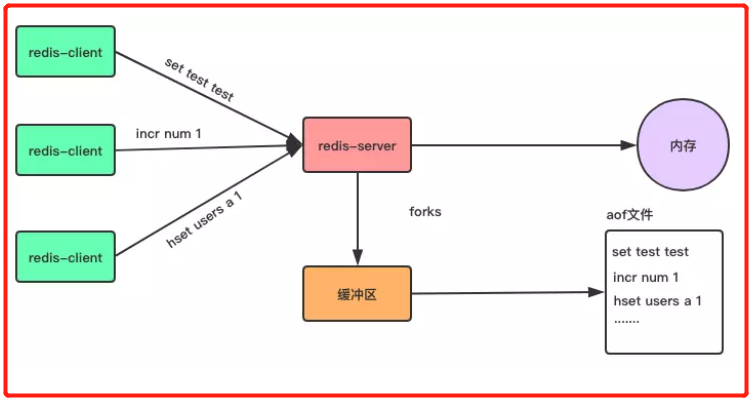
特点：

1. 如果要开启改模式要修改配置文件
2. 可实现数据实时的持久化
3. 效率低于RDB模式
4. 因为该模式是记录用户的所有操作，所以持久化的文件会越来越大，文件越大数据恢复的效率会越低
5. 该操作的异步的，不会导致阻塞

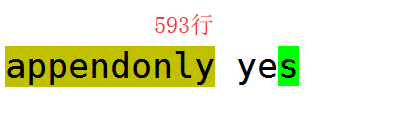


### 2.3.1如果开启Aof模式呢

与RDB存储某个时刻的快照不同，AOF持久化方式会记录客户端对服务器的每一次写操作命令，并将这些写操作以Redis协议追加保存到后缀为aof文件末尾，在Redis服务器重启时，会加载并运行aof文件的命令，以达到恢复数据的目的。

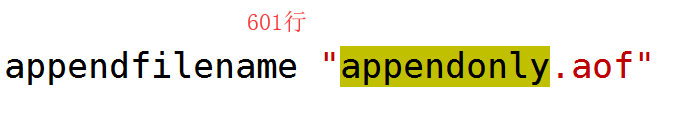


1. 在配置文件中设置 appendOnly yes 则开启了AOF模式



1. 如果修改appendonly.aof的文件名称

Appendfilename “appendonly.aof” 这里修改



1. Aof的持久化策略：如下

#### 三种写入策略

# appendfsync always 执行一次set操作,持久化一次.

appendfsync everysec 每秒持久化一次.

# appendfsync no 不开启持久化

有了持久化策略，当然也会有内存的策略，不然可能会导致内存的益处

## 3.Redis的内存策略

#### Redis的内存回收主要分为过期删除策略和内存淘汰策略两种

### 过期删除策略

当Redis没有适当的内存策略时，就可能到导致内存的益处，所以Redis必须使用内存优化算法,LRU算法

### 3.1LRU算法

内存管理的一种页面置换算法，对于在内存中但又不使用的内存块称为LRU,操作系统会根据哪些数据属于LRU,然后将其移除腾出新的空间来储存新的数据

### 3.2Redis内存大小的设定

在537行设定数据.1kb 1mb 1gb

Maxmemory <bytes> 默认

### 3.3Redis的内存策略

1. volatile-lru 设定了超时时间的数据，按照LRU算法
2. allkeys-lru 设定了全部key值，按照LRU算法
3. volatile-random 设定了超时时间随机
4. allkeys-random 所有键随机
5. valatitle-ttl 根据ttl存活时间 删除数据
6. noeviction 用不删除

## 关于缓存使用的问题

### 缓存穿透

特点:用户频繁查询一个不存在的数据.redis本身不会缓存该数据.数据库中没有该数据.

如果是恶意用户用多线程同时查询

### 缓存击穿

特点：数据库中有这个数据，但是Redis中没有缓存到，并发执行了时，可能会导致数据库宕机

### 缓存雪崩

特点：数据库中有数据，但是redis中的缓存数据命中率低，导致大量的请求直接访问数据库

## 5.Redis的分片技术

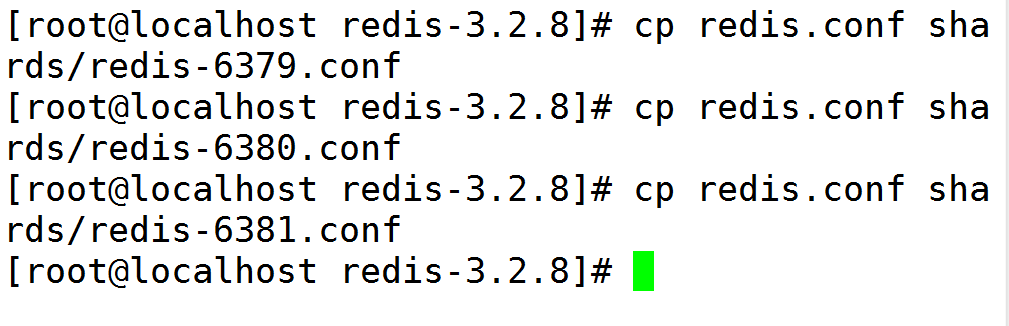
为什么要引入分片技术，因为单台的Redis不能分配过大的内存空间，内存空间过大就会导致寻址过慢，降低效率

定义：多台Redis为用户提供一个超大的内存空间，并且每个Redis中只保存一部分的数据

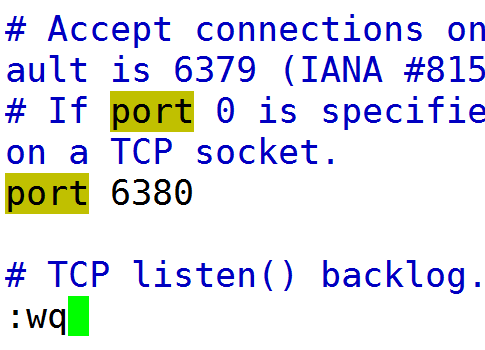
### 5.1如何实现Redis的分片

1. 首先复制Redis.conf的文件，一份配置文件代表一台Redis ，配置文件不

能同名



1. 改Port端口



1. 启动Redis：redis-server Redis.conf（Redis相应的配置文件）

[root@localhost shards]# redis-server redis-6379.conf

[root@localhost shards]# redis-server redis-6380.conf

[root@localhost shards]# redis-server redis-6381.conf

1. 然后用命令 : ps -ef|grep redis 查看进程是否已启动

### 5.2Spring 整合Redis的分片

* 采用ShardedJedis 这个类进行构造，其构造函数中的参数可传入单个或一个Set集合，因为Set集合可以去重

### 5.3Redis分片的缺点

* Redis本身没有实现高可用
* 如果有一台Redis宕机了就会影响整个分片系统的运行
* 一旦运行了就不能关机
* 可能会造成数据的丢失

### 5.4Redis分片的特性

1. 哈希一致性，同一个IP每次哈希得到的结构都是一样的
2. 均衡性，如果有三个Redis，由于哈希得到的结构不一定是平均分配的，可能导致内存均衡不一，这时候Redis就会创建有虚拟节点，将数据转储
3. 单调性，Redis分片搭建后，节点只能多，不能少，否则会系统崩溃
4. 负载/离散性：在分布式系统中用户使用的内存时，不能获得全部的内存空间，导致不同的Key可能会存储到同一个地址，相同的key保存到了不同的位置时称之为**分散性**

## Redis的主从搭建（实现Redis高可用的前提条件）

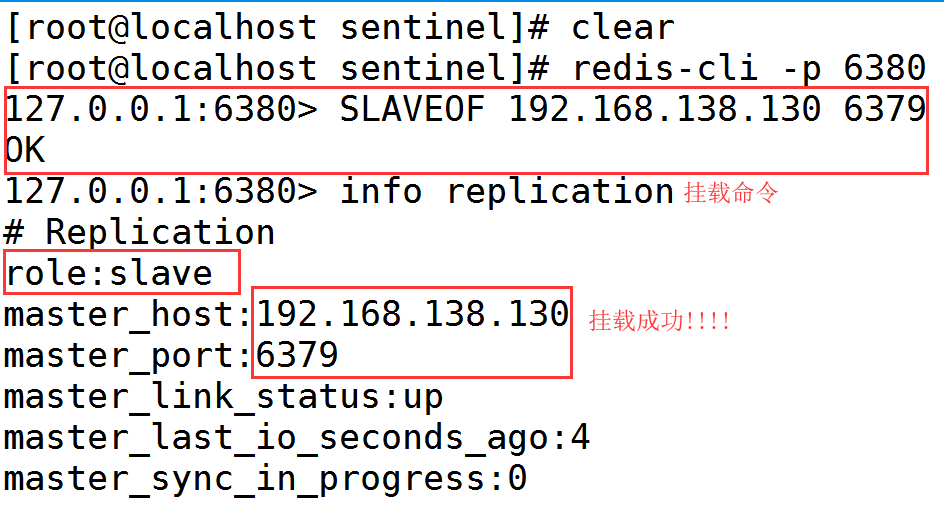
* 首先要复制配置文件Redis.conf
* 改端口
* 启动服务

[root@localhost sentinel]# redis-server redis-6379.conf

[root@localhost sentinel]# redis-server redis-6380.conf

[root@localhost sentinel]# redis-server redis-6381.conf

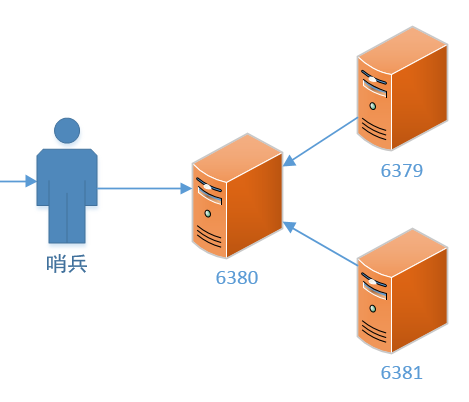
* 启动Redis 进入后台 redis-cli -p 端口号
* 执行挂载命令 slaveof 主机的IP 主机的端口



* 查看是否挂载成功 info replication
* 完成上述操作后，实现了主从挂载，但是从的只能读，不能写

## 6哨兵sentinel

为什么要引入哨兵，因为Redis主机可能会挂掉，然后要有人在从机中选一台出来当主机，继续实现主从挂载，当原主机修好了，就会成为从机



* 哨兵可以监听只主机的生命状态(心跳检测，利用PING-PONG实现
* 如果发起了三次PING 主机都没有响应PONG,则认为主机宕机了
* 并且修改其他机器的配置文件，实现主从挂载，切换主机操作

## 7.五个数据结构

### 7.1 String

Redis的字符串是动态字符串，是可以修改的字符串，内部结构上类似于JAVA的ArrayList，采用预分配冗余空间来减少内存的频繁分配，内部为当前字符串实际分配的空间capacity一般要高于实际字符串长度len。当字符串长度小于1M时，扩容都是加倍现有的空间，如果超过1M，扩容时一次只会多扩1M的空间。需要注意的是字符串最大长度为512M。

**初始化字符串** 需要提供「变量名称」和「变量的内容」

> set ireader beijing.zhangyue.keji.gufen.youxian.gongsi

OK

**获取字符串的内容** 提供「变量名称」

> get ireader

"beijing.zhangyue.keji.gufen.youxian.gongsi"

**获取字符串的长度** 提供「变量名称」

> strlen ireader

(integer) 42

**获取子串** 提供「变量名称」以及开始和结束位置[start, end]

Getrange ireader 28 34

"youxian"

**覆盖子串** 提供「变量名称」以及开始位置和目标子串

Setrange ireader 28 wooxian

get ireader

"beijing.zhangyue.keji.gufen.wooxian.gongsi"

**追加子串**

**Append ireader .hao**

(integer) 46 # 返回长度

> get ireader

"beijing.zhangyue.keji.gufen.wooxian.gongsi.hao"

遗憾的是字符串没有提供字串插入方法和子串删除方法。

**计数器** 如果字符串的内容是一个整数，那么还可以将字符串当成计数器来使用。

Set ireader 40

OK

> get ireader

"42"

> incrby ireader 100

(integer) 142

> get ireader

"142"

> decrby ireader 100

(integer) 42

> incr ireader # 等价于incrby ireader 1

(integer) 43

> decr ireader # 等价于decrby ireader 1

(integer) 42

计数器是有范围的，它不能超过Long.Max，不能低于Long.MIN，字符串是由多个字节组成，每个字节又是由 8 个 bit 组成，如此便可以将一个字符串看成很多 bit 的组合，这便是 bitmap「位图」数据结构，位图的具体使用会放到后面的章节来讲。

**过期和删除** 字符串可以使用del指令进行主动删除，可以使用expire指令设置过期时间，到点会自动删除，这属于被动删除。可以使用ttl指令获取字符串的寿命。

> expire ireader 60

(integer) 1 # 1表示设置成功，0表示变量ireader不存在

> ttl ireader

(integer) 50 # 还有50秒的寿命，返回-2表示变量不存在，-1表示没有设置过期时间

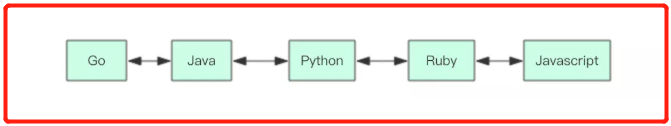
> del ireader

(integer) 1 # 删除成功返回1

> get ireader

(nil) # 变量ireader没有了

### 7.2 list



Redis将列表数据结构命名为list而不是array，是因为列表的存储结构用的是链表而不是数组，而且链表还是双向链表。因为它是链表，所以随机定位性能较弱，首尾插入删除性能较优。如果list的列表长度很长，使用时我们一定要关注链表相关操作的时间复杂度。

**负下标** 链表元素的位置使用自然数0,1,2,....n-1表示，还可以使用负数-1,-2,...-n来表示，-1表示「倒数第一」，-2表示「倒数第二」，那么-n就表示第一个元素，对应的下标为0。

**队列／堆栈** 链表可以从表头或者表尾追加和移除元素，结合使用rpush/rpop/lpush/lpop四条指令,可以将链表最为队列或者栈使用，向左向右进行都可以

# 右进左出

> rpush ireader go

(integer) 1

> rpush ireader java python

(integer) 3

> lpop ireader

"go"

> lpop ireader

"java"

> lpop ireader

"python"

# 左进右出

> lpush ireader go java python

(integer) 3

> rpop ireader

"go"

在日常应用中，列表常用来作为异步队列来使用。

**长度** 使用llen指令获取链表长度

> rpush ireader go java python

(integer) 3

> llen ireader

(integer) 3

**随机读** 可以使用lindex指令访问指定位置的元素，使用lrange指令来获取链表子元素列表，提供start和end下标参数，下标从0开始

rpush ireader go java python

(integer) 3

> lindex ireader 1

"java"

> lrange ireader 0 2

1) "go"

2) "java"

3) "python"

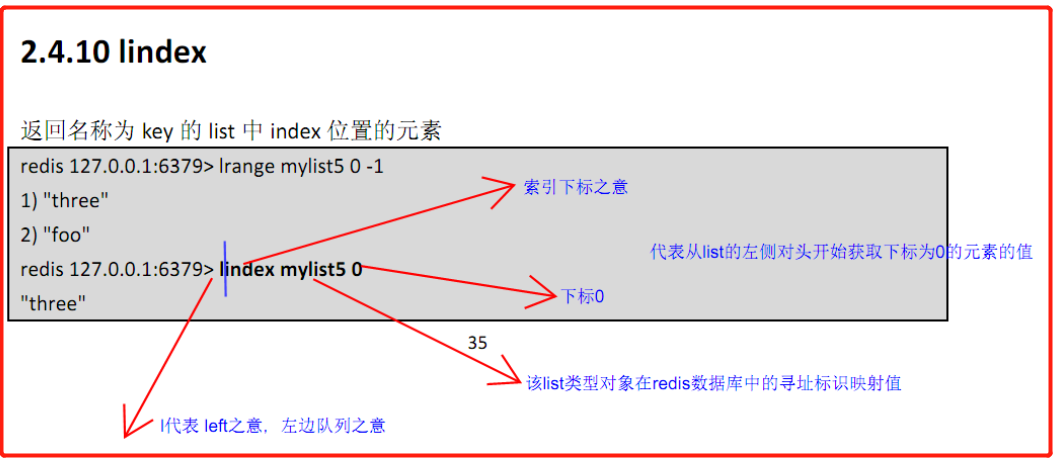
> lrange ireader 0 -1 # -1表示倒数第一

1) "go"

2) "java"

3) "python"

使用lrange获取全部元素时，需要提供end\_index，如果没有负下标，就需要首先通过llen指令获取长度，才可以得出end\_index的值，有了负下标，使用-1代替end\_index就可以达到相同的效果。



**修改元素** 使用lset指令在指定位置修改元素。

> rpush ireader go java python

(integer) 3

> lset ireader 1 javascript

OK

Lrange ireader 0 -1

1) "go"

2) "javascript"

3) "python"

**插入元素** 使用linsert指令在列表的中间位置插入元素，有经验的程序员都知道在插入元素时，我们经常搞不清楚是在指定位置的前面插入还是后面插入，所以antirez在linsert指令里增加了方向参数before/after来显示指示前置和后置插入。不过让人意想不到的是linsert指令并不是通过指定位置来插入，而是通过指定具体的值。这是因为在分布式环境下，列表的元素总是频繁变动的，意味着上一时刻计算的元素下标在下一时刻可能就不是你所期望的下标了。

> rpush ireader go java python

(integer) 3

Linsert ireader before java ruby

(integer) 4

> lrange ireader 0 -1

1) "go"

2) "ruby"

3) "java"

4) "python"

**删除元素** 列表的删除操作也不是通过指定下标来确定元素的，你需要指定删除的最大个数以及元素的值

> rpush ireader go java python

(integer) 3

> lrem ireader 1 java

(integer) 1

> lrange ireader 0 -1

1) "go"

2) "python"

**定长列表** 在实际应用场景中，我们有时候会遇到「定长列表」的需求。比如要以走马灯的形式实时显示中奖用户名列表，因为中奖用户实在太多，能显示的数量一般不超过100条，那么这里就会使用到定长列表。维持定长列表的指令是ltrim, 需要提供两个参数start和end，表示需要保留列表的下标范围，范围之外的所有元素都将被移除。

> rpush ireader go java python javascript ruby erlang rust cpp

(integer) 8

Ltrim ireader -3 -1

OK

> lrange ireader 0 -1

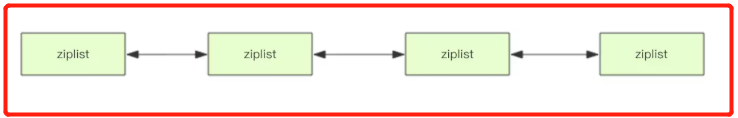
1) "erlang"

2) "rust"

3) "cpp"

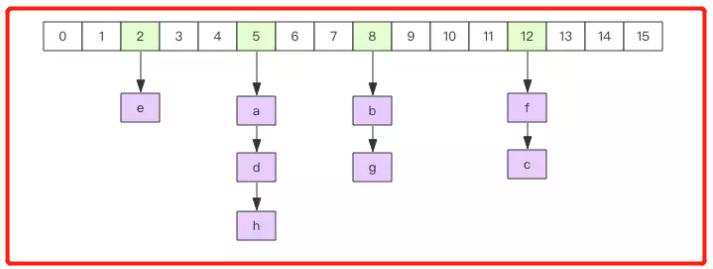
如果指定参数的end对应的真实下标小于start,是效果相当于del指令，因为这样的参数表示需要需要保留列表元素的下标范围为空。

**快速列表**



如果再深入一点，你会发现Redis底层存储的还不是一个简单的linkedlist，而是称之为快速链表quicklist的一个结构。首先在列表元素较少的情况下会使用一块连续的存储存储，这个结构是ziplist，也就是压缩列表。它将所有元素紧挨着一起存储，分配的是一块连续的内存。当数据量比较多的时候才会改成quicklist。因为普通的链表需要的附加指针，所以占用空间较大。比如这个列表里存的只是int类型的数据，结构上还需要两个额外的指针prev和next。所以Redis将链表和ziplist结合起来组合成quicklist。也就是将多个ziplist使用双向指针串起来使用。这样既满足了快速的插入删除性能，又不会出现太大的空间冗余。

### 7.3 hash



**增加元素** 可以使用hset一次增加一个键值对,也可以对hmset一次增加多个键值对

> hset ireader go fast

(integer) 1

> hmset ireader java fast python slow

OK

**获取元素** 可以通过hget定位具体key对应的value，可以通过hmget获取多个key对应的value，可以使用hgetall获取所有键值对的集合，也可以用hkeys和 hvals获得所有key列表和value列表

hmset ireader go fast java fast python slow

OK

> hget ireader go

"fast"

> hmget ireader go python

1) "fast"

2) "slow"

> hgetall ireader

1) "go"

2) "fast"

3) "java"

4) "fast"

5) "python"

6) "slow"

>hkeys ireader

1) "go"

2) "java"

3) "python"

> hvals ireader

1) "fast"

2) "fast"

3) "slow"

**删除元素** 可以使用hdel删除指定key，hdel支持同时删除多个key

> hmset ireader go fast java fast python slow

OK

>hdel ireader go

(integer) 1

>hmdel ireader java python

(integer) 2

**判断元素是否存在**通常我们使用hget获得key对应的value是否为空就直到对应的元素是否存在了，不过如果value的字符串长度特别大，通过这种方式来判断元素是否存在就略显浪费，可以使用hexists指令

> hmset ireader go fast java fast python slow

OK

>exists ireader go

(integer) 1

**计数器** hash结构还可以当成计数器来使用，对于 内部的每一个Key都可以作为独立的计数器。如果value不是整数，调用hincrby 指令会出错

> hincrby ireader go 1

(integer) 1

>hincrby ireader java 2

(integer) 2

>hincrby ireader python 4

(integer) 4

>hgetall ireader

1) "go"

2) "1"

3) "python"

4) "4"

5) "java"

6) "4"

> hset ireader rust good

(integer) 1

> hincrby ireader rust 1

(error) ERR hash value is not an integer

**扩容** 当hash内部的元素比较拥挤时(hash碰撞比较频繁)，就需要进行扩容。扩容需要申请新的两倍大小的数组，然后将所有的键值对重新分配到新的数组下标对应的链表中。如果hash结构很大，比如有上百万个键值对，那么一次完整rehash的过程就会耗时很长，这对于单线程的redis来说有点压力山大。所以Redis采用了渐进式rehash的方案。它会同时保留两个新旧hash结构，在后续的定时任务及hash结构的读写指令中将旧结构的数据逐渐迁移到新的结构中，这样就可以避免因扩容导致的线程卡顿现象

**缩容** Redis的hash结构不但有扩容还有缩容，从这一点出发，它要比Java的HashMap要厉害一些，Java的HashMap只有扩容。缩容的原理和扩容是一致的，只不过新的数组大小要比旧数组小一倍。

### 7.4 set

Java程序员都知道HashSet的内部实现使用的是HashMap，只不过所有的value都指向同一个对象。Redis的set结构也是一样，它的内部也使用hash结构，所有的value都指向同一个内部值。

**增加元素** 可以一次增加多个元素sadd

> sadd ireader go java python

(integer) 3

**读取元素** 使用smembers列出所有元素，使用scard获取元素集合长度，使用srandmember随机获取count个元素，如果不提供count，默认为1

>sadd ireader go java python

(integer) 3

> smembers ireader

1) "java"

2) "python"

3) "go"

> scard ireader

(integer) 3

> srandmember ireader

"java"

**删除元素** 使用srem 删除一个到多个元素，使用spop删除随机一个元素

> sadd ireader go java python rust erlang

(integer) 5

>srem ireader go java

(integer) 2

>spop ireader

"erlang"

**判断元素是否存在** 使用sismember指令，只能接收单个元素

> sadd ireader go java python rust erlang

(integer) 5

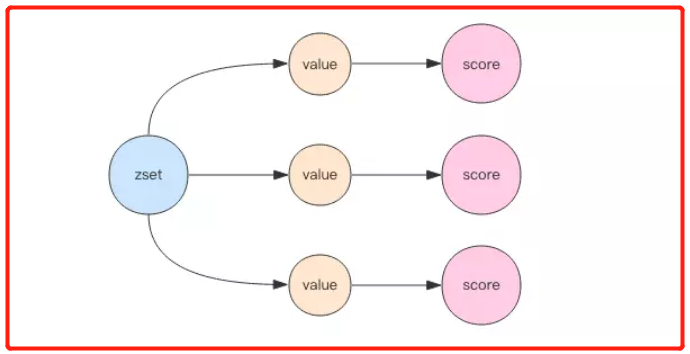
> sismember ireader java

(integer) 1

> sismember ireader

(integer) 0

### 7.5 sortedset(zset)



SortedSet(zset)是Redis提供的一个非常特别的数据结构，一方面它等价于Java的数据结构Map<String,Double>,可以给每一个元素value赋予一个权重score，另一方面又类似于TreeSet,内部的元素会按照权重score进行排序，可以得到每个元素的名次，还可以通过score的范围来获取元素的列表。

zset底层实现使用了两个数据结构，第一个是hash，第二个是跳跃列表，hash的作用就是关联元素value和权重score,保障元素value的唯一性，可以通过value找到对应的score值。跳跃列表的目的在于给元素value排序，根据score的范围获取元素列表。

**增加元素** 通过zadd指令可以增加一到多个value/score对，score放在前面（如果值相同会覆盖吗？）

> zadd ireader 2.0 java

(integer) 1

>zadd ireader 4.0 java 3.0 python

(integer) 2

**长度** 通过指令zcard可以得到zset的元素个数

> zcard ireader

(integer) 3

**删除元素** 通过指令zrem可以删除zset中的元素，可以一次删除多个

> zrem ireader go python

(integer) 2

**计数器** 同hash结构一样，zset也可以作为计数器使用。

> zadd ireader 4.0 python 3.0 go 5.0 java

(integer) 3

>zincrby ireader 1.0 python

"5"

**获取排名和分数** 通过zscore指令获取指定元素的权重，通过zrank指令获取指定元素的正向排名，通过zrevrank指令获取指定元素的反向排名。正向有小到大，负向由大到小

> zscore ireader python

"5"

> zrank ireader go # 分数低的排名考前，rank值小

(integer) 0

> zrank ireader python

(integer) 1

> zrank ireader java

(integer) 2

>zrevrank ireader java

0

**根据score范围获取列表** 通过zrangebyscore指令指定score范围获取对应的元素列表。通过zrevrangebyscore指令获取倒排元素列表。正向是由小到大，负向是由大到小。参数-inf表示负无穷，+inf表示正无穷。

> zrangebyscore ireader 0 5

1) "go"

2) "java"

3) "python"

> zrangebyscore ireader -inf +inf withscores

1) "go"

2) "1"

3) "java"

4) "4"

5) "python"

6) "5"

**根据范围移除元素列表** 可以通过排名范围，也可以通过score范围来一次性移除多个元素

> zremrangebyrank ireader 0 1

(integer) 2

> zadd ireader 4.0 java 1.0 go

(integer) 2

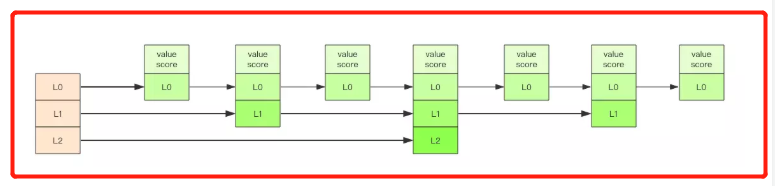
> zremrangebyscore ireader -inf 4

(integer) 2

> zrange ireader 0 -1

1) "python"

**跳跃列表** zset内部的排序功能是通过「跳跃列表」数据结构来实现的，它的结构非常特殊，也比较复杂。这一块的内容深度读者要有心理准备。



## 8.过期时间

### 设置过期时间的命令

EXPIRE <KEY> <TTL> : 将键的生存时间设为 ttl 秒

PEXPIRE <KEY> <TTL> :将键的生存时间设为 ttl 毫秒

EXPIREAT <KEY> <timestamp> :将键的过期时间设为 timestamp 所指定的秒数时间戳

PEXPIREAT <KEY> <timestamp>: 将键的过期时间设为 timestamp 所指定的毫秒数时间戳.

### 取消过期时间的命令

1.重新赋值

127.0.0.1:6379> set message "hello"

OK

127.0.0.1:6379> expire message 60

(integer) 1

127.0.0.1:6379> ttl message

(integer) 54

127.0.0.1:6379> persist message

(integer) 1

127.0.0.1:6379> ttl message

(integer) -1

persist命令就是expire命令的反命令，这个函数在过期字典中查找给定的键,并从过期字典中移除。

比如在数据库当前状态(如上图所示)，当给book这个key移除过期时间：