## 一、键相关

1. **获得符合规则的键名列表**

KEYS pattern

其中，pattern中——?:匹配一个字符；\*：匹配任意个（包括0个）字符；[]：匹配括号间的任一字符，可以使用“-”符号表示一个范围；\x：匹配字符x，用于转义符号，如要匹配“？”就需要\?

1. **判断一个键是否存在**

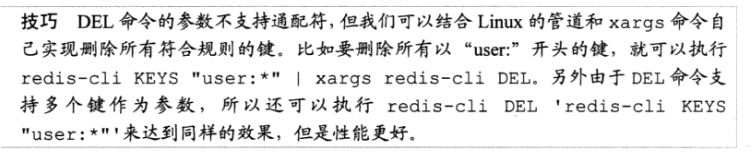
EXISTS key

存在则返回整数类型1，否则返回0

1. **删除键**

DEL key [key …]

可以删除一个或多个键，返回值是删除的键的个数



1. **获得键值的数据类型**

TYPE key

对于键的命名并没有强制的要求，但比较好的实践是用“对象类型：对象id：对象属性”来命名一个键；对于多个单词则推荐使用“.”分割。

## 二、数据类型

### 1、字符串类型

1. **赋值与取值**

SET key value

GET key

1. **递增数字**

INCR key

让当前键值递增，并返回递增后的值。（前提是该键下的值可以转化为整数形式）；当要操作的键不存在是会默认键值为0，所以第一次递增后的结果是1

1. **增加指定的整数**

INCRBY key increment

1. **减少指定的整数**

DECR key

DECRBY key decrement

1. **增加指定浮点数**

INCRBYFLOAT key increment

1. **向尾部追加值**

APPEND key value

返回值是追加后字符串的总长度

1. **获取字符串长度**

STRLEN key

1. **同时获得、设置多个键值**

MGET key [key …]

MSET key value [key value ….]

1. **位操作**

GETBIT key offset

SETBIT key offset value

BITCOUNT key [start] [end]

BITOP operation destkey key [key …]

### 2、散列类型

散列类型适合存储对象：使用对象类别和id构成键名，使用字段表示对象的属性，而字段值则存储属性值。

1. **赋值与取值**

HSET key field value

HGET key field

HMSET key field value [field value …]

HMGET key field [field …]

HGETALL key

HSET命令不区分插入和更新操作，当执行的是插入操作时（即之前字段不存在）HSET命令会返回1，当执行的是更新操作时会返回0.

1. **判断字段是否存在**

HEXISTS key field

如果存在则返回1，否则返回0（如果键不存在也会返回0）

1. **当字段不存在时赋值**

HSETNX key field value

HSETNX命令与HSET命令类似，区别在于如果字段已经存在，HSETNX命令将不执行任何操作。

1. **增加数字**

HINCRBY key field increment

1. **删除字段**

HDEL key field [field …]

1. **只获取字段名或字段值**

HKEYS key

HVALS key

1. **获得字段数量**

HLEN key

### 3、列表类型

列表类型（list）可以存储一个有序的字符串列表，常用的操作是向列表两端添加元素，或者获得列表的某一个片段。

列表类型内部是使用双向链表实现的，所以向列表两端添加元素的时间复杂度为O(1)，获取越接近两端的元素速度就越快。

1. **向列表两端增加元素**

LPUSH key value [value…]

RPUSH key value [value…]

1. **从列表两端弹出元素**

LPOP key

RPOP key

1. **获取列表中元素的个数**

LLEN key

1. **获得列表片段**

LRANGE key start stop

返回索引从start到stop之间的所有元素（包含两端的元素），起始索引为0。支持负索引，表示从右边开始计算序数，如“-1”表示最右边第一个元素，“-2”表示最右边第二个元素

显然，LRANGE numbers 0 -1可以获取列表中的所有元素。

1. **删除列表中指定的值**

LREM key count value

删除列表中前count个值为value的元素，返回值是实际删除的元素个数。

1. **获得、设置指定索引的元素值**

LINDEX key index

LSET key index value

索引从0开始，负数则表示从右边开始计算的索引，最右边元素的索引是-1.

1. **只保留列表指定片段**

LTRIM key start end

删除指定索引范围之外的所有元素，其指定列表范围的方法和LRANGE命令相同。

1. **向列表中插入元素**

LINSERT key BEFORE|AFTER pivot value

首先会在列表中从左到右查找值为pivot的元素，然后根据第二个参数是before还是after来决定将value插入到该元素的前面还是后面。

1. **将元素从一个列表转到另一个列表**

RPOPLPUSH source destination

当source和destination相同时，RPOPLPUSH命令会不断地将队尾的元素移到队首。

### 4、集合类型

1. **增加、删除元素**

SADD key member [member…]

SREM key member [member…]

返回值是成功加入的元素数量（忽略的元素不计算在内）；删除一个或多个元素是，返回删除成功的个数

1. **获得集合中的所有元素**

SMEMBERS key

1. **判断元素是否在集合中**

SISMEMBER key member

当存在时返回1，当值不存在或键不存在时返回0

1. **集合间运算**

SDIFF key [key….]

用来对多个集合执行差集运算，集合a与集合b的差集表示为a-b，代表所有属于a且不属于b的元素构成的集合。

SINTER key [key…]

用来对多个集合执行交集运算。

SUNION key [key…]

用来对多个集合执行并集运算。

1. **进行集合运算并将结果存储**

SDIFFSTORE destination key [key…]

SINTERSTORE destination key [key…]

SUNIONSTORE destination key [key…]

1. **获得集合中元素个数**

SCARD key

1. **随机获得集合中的元素**

SRANDMEMBER key [count]

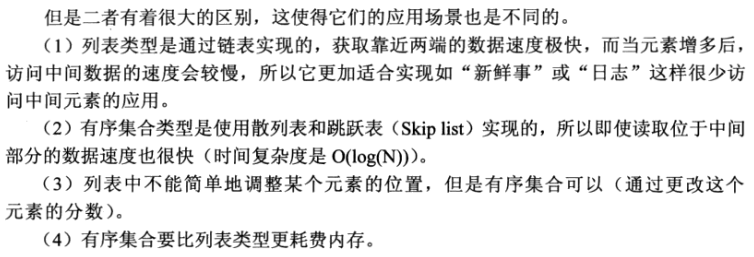
当count为正数时，会随机从集合里获得count个不重复的元素；当count为负数时，会随机从集合里获得|count|个的元素，这些元素有可能相同。

1. **从集合中弹出一个元素（随机）**

SPOP key

### 5、有序集合类型

有序集合类型在某些方面和列表类型有些相似：二者都是有序的，都可以获得某一范围的元素。



1. **增加元素**

ZADD key score member [score member…]

用来向有序集合中加入一个元素和该元素的分数，如果该元素已经存在则会用新的分数替换原有的分数。返回值是新加入到集合中的元素个数（不包含之前已经存在的元素）

1. **获得元素的分数**

ZSCORE key member

1. **获得排名在某个范围的元素列表**

ZRANGE key start stop [withscores]

ZREVRANGE key start stop [withscores]

Zrange命令会按照元素分数从小到大的顺序返回索引从start到stop之间的所有元素（包含两端的元素），与lrange命令十分相似，索引都是从0开始的，负数代表从后向前查找（-1代表最后一个元素）

1. **获得指定分数范围的元素**

ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]

按照元素分数从小到大的顺序返回分数在min和max之间（包含min和max）的元素。

如果希望分数范围不包含端点值，可以在分数前加上“（”符号

参数min和max还支持无穷大，同zadd命令一样，-inf和+inf分别表示负无穷和正无穷。

需要注意的是ZREVRANGEBYSCORE命令不仅是按照元素分数从大往小的顺序给出结果的，而且它的min和max参数的顺序和ZRANGEBYSCORE命令是相反的。

1. **增加某个元素的分数**

ZINCREBY key increment member

返回值是更改后的分数；increment也可以是个负数表示减分；如果指定的元素不存在，会先建立它并将它的分数赋为0再执行操作。

1. **获得指定分数范围内的元素个数**

ZCOUNT key min max

1. **按照排名范围删除元素**

ZREMRANGEBYRANK key start stop

按照元素分数从小到大的顺序（即索引0表示最小的值）删除处在指定排名范围内的所有元素，并返回删除的元素数量。

1. **按照分数范围删除元素**

ZREMRANGEBYSCORE key min max

1. **获得元素的排名**

ZRANK key member

ZREVRANK key member

按照元素分数从小到大的顺序获得指定的元素的排名（从0开始，即分数最小的元素排名为0）；ZREVRANK命令则相反（分数最大的元素排名为0）

1. **计算有序集合的交集**

ZINTERSTORE destination numkeys key [key…] [WEIGHTS weight [weight…]] [AGGREGATE SUM | MIN | MAX ]

ZINTERSTORE命令用来计算多个有序集合的交集并将结果存储在destination键中（同样以有序集合类型存储），返回值为destination键中的元素个数。Destination键中元素的分数由AGGREGATE参数决定的。

当AGGREGATE是SUM时（也就是默认值），destination键中元素的分数是每个参与计算的集合中该元素分数的和。

同样可以通过WEIGHTS参数设置每个集合的权重，每个集合在参与计算是元素的分数会被乘上该集合的权重。

另外，ZUNIONSTORE的作用是计算集合间的并集，与ZINTESTORE命令的用法一样。

## 三、事务

### 1、概述

就是先将属于一个事务的命令发送给redis，然后再让redis依次执行这些命令。



Redis保证一个事务中的所有命令要么都执行，要么都不执行。如果在发送exec命令前客户端断线了，则redis会清空事务队列，事务中的所有命令都不会执行。而一旦客户端发送了exec命令，所有的命令就都会被执行，即使此后客户端断线也没关系，因为redis中已经记录了所有要执行的命令。

除此之外，redis的事务还能保证一个事务内的命令依次执行而不被其他命令插入

### 2、错误处理

语法错误：命令不存在或者命令参数的个数不对。执行exec命令后redis就会直接返回错误，连语法正确的命令也不会执行。

运行错误，指在命令执行时出现的错误，比如使用散列类型的命令操作集合类型的键，这种错误在实际执行之前redis是无法发现的，所以事务里这样的命令是会被redis接受并执行的。如果事务里的一条命令出现了运行错误，事务里其他的命令依然会继续执行（包括出错命令之后的命令）

### 3、Watch命令介绍

Watch命令可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行了。

提示：由于watch命令的作用只是当被监控的键值被修改后阻止之后一个事务的执行，而不能保证其他客户端不修改这一键值，所以我们需要在EXEC执行失败后重新执行整个函数。

·

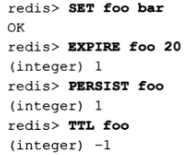
## 四、生存时间

可以使用EXPIRE命令设置一个键的生存时间，到时间后redis会自动删除它

EXPIRE key seconds：其中seconds参数表示键的生存时间，单位是秒；返回1表示设置成功，返回0则表示键不存在或设置失败。

如果想知道一个键还有多久的时间会被删除，可以使用TTL命令；返回值是键的剩余时间（单位是秒）；当键不存在时TTL会返回-1，另外没有为键设置生存时间（即永久存在，这是建立一个键后的默认情况）同样会返回-1.

如果想取消键的生存时间设置（即将键恢复成永久的），可以使用PERSIST命令。



除了PERSIST命令之外，使用SET或GETSET命令为键赋值也会同时清除键的生存时间；使用EXPIRE命令会重新设置键的生存时间。

其他只对键值进行操作的命令（如INCR、LPUSH、HSET、ZREM）均不会影响键的生存时间。

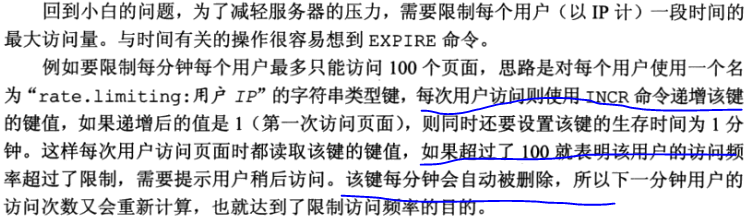
PEXPIRE的时间单位是毫秒，即PEXPIRE key 1000与EXPIRE key 1等价，对应地可以用PTTL命令以毫秒为单位返回键的剩余时间。



另外还有两个相对不太常用的命令：EXPIREAT和PEXPIREAT。使用Unix时间作为第二个参数表示键的生存时间的截止时间；PEXPIREAT的单位是毫秒。

### 1、实现访问频率限制

方式一：



这种方式的一个问题是：如果一个用户在一分钟的最后一秒访问了9次，又在下一分钟的的第一秒访问了10次；这样的话，该用户在2秒钟内就访问了19次博客。

方式二：

对每个用户使用一个列表类型的键来记录他最近10次访问博客的时间。一旦键中的元素超过十个，就判断时间最早的元素距现在的时间是否小于1分钟。如果是则表示最近一分钟的访问次数超过了10次，如果不是就将现在的时间加入到列表中，同时把最早的元素删除。

### 2、实现缓存

实际开发中很难为缓存键设置合理的生存时间，为此可以限制redis能够使用的最大内存，并让redis按照一定的规则淘汰不需要的缓存键，这种方式在只将redis用作缓存系统时非常实用。

具体的设置方法为：修改配置文件的maxmemory参数，限制redis最大可用内存大小（单位是字节），当超出了这个限制时redis会依据maxmemory-policy参数指定的策略来删除不需要的键，直到redis占用的内存小于指定的内存。

|  |  |
| --- | --- |
| 规则 | 说明 |
| volatile-lru | 使用LRU算法删除一个键（只对设置了生存时间的键） |
| allkeys-lru | 使用LRU算法删除一个键 |
| volatile-random | 随机删除一个键（只对设置了生存时间的键） |
| allkeys-random | 随机删除一个键 |
| volatile-ttl | 删除生存时间最近的一个键 |
| noeviction | 不删除键，只返回错误 |

## 五、排序

### 1、SORT命令

SORT命令可以对列表类型、集合类型和有序集合类型键进行排序，并且可以完成与关系数据库中的连接查询相类似的任务。

在对有序集合类型排序时会忽略元素的分数，只针对元素自身的值进行排序。

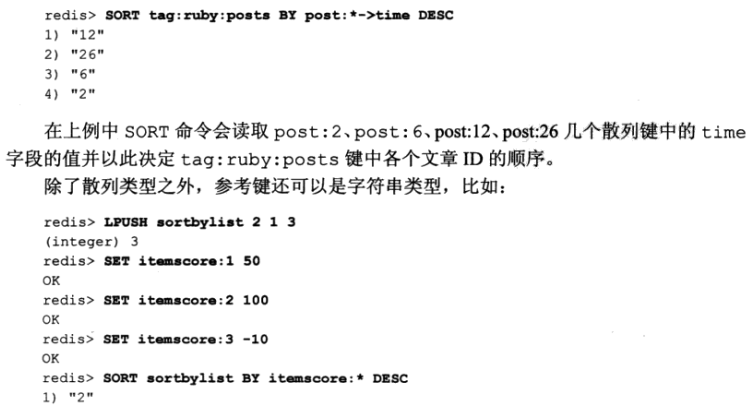
除了可以排列数字外，sort命令还可以通过alpha参数实现按照字典顺序排列非数字元素。如：SORT key ALPHA

SORT命令的DESC参数可以实现将元素按照从大到小的顺序排列。

SORT命令还支持LIMIT参数返回指定范围的结果：SORT key DESC LIMIT offset count

### 2、BY参数

BY参数的语法为“BY参考键”。其中参考键可以是字符串类型键或者是散列类型键的某个字段（表示为键名->字段名）。如果提供了by参数，sort命令将不再依据元素自身的值进行排序，而是对每个元素使用元素的值替换参考键中的第一个“\*”并获取其值，然后依据该值对元素排序。

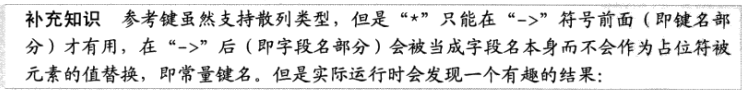


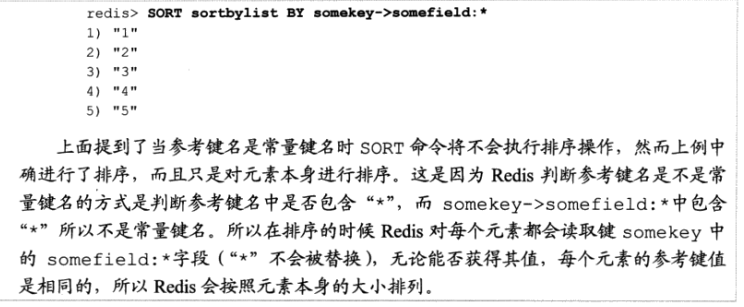
当参考键名不包含“\*”时（即常量键名，与元素值无关），sort命令将不会执行排序操作，因为redis认为这种情况是没有意义的（因为所有要比较的值都一样）：

SORT key BY anytext ——anytext是常量键名（甚至anytext键可以不存在），此时sort的结果与lrange的结果相同。

如果几个元素的参考键值相同，则sort命令会再比较元素本身的值来决定元素的顺序。

当某个元素的参考键不存在时，会默认参考键的值为0.

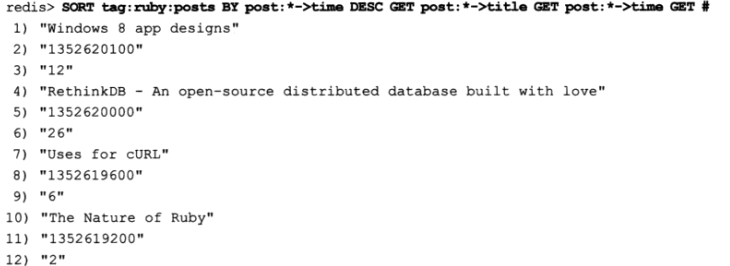




### 3、GET参数

GET参数不影响排序，它的作用是使SORT命令的返回结果不再是元素自身的值，而是GET参数中指定的键值。GET参数的规则和BY参数一样，GET参数也支持字符串和散列类型的键，并使用“\*”作为占位符。例如，要实现在排序后直接返回id对应的文章标题，可以这样写：SORT tag:ruby:posts BY post:\*->time DESC GET post:\*->title

在一个sort命令中可以有多个GET参数（而BY参数只能有一个），而且使用“#”可以返回元素本身的值。



### 4、STORE参数

默认情况下SORT会直接返回排序结果，如果希望保存排序结果，可以使用STORE参数。如希望把结果保存到sort.result键中。

SORT tag:ruby:posts BY post:\*->time DESC GET post:\*->title GET post:\*->time GET # STORE sort.result

保存后的键的类型为列表类型，如果键已经存在则会覆盖它。加上STORE参数后SORT命令的返回值为结果的个数。STORE参数常用来结合EXPIRE命令缓存排序结果。

### 5、性能优化

SORT是Redis中最强大最复杂的命令之一，如果使用不好很容易成为性能瓶颈。SORT命令的时间复杂度O(n+mlogm)，其中n表示要排序的列表（集合或有序集合）中的元素个数，m表示要返回的元素个数。当n较大时sort命令的性能相对较低，并且redis在排序前会建立一个长度为n的容器来存储待排序的元素，虽然是一个临时的过程，但如果同时进行较多的大数据量排序操作则会严重影响性能。

使用SORT命令时需要注意：1）尽可能减少待排序键中元素的数量（使n尽可能小）；2）使用LIMIT参数只获取需要的数据（使m尽可能小）；3）如果要排序的数据数量较大，尽可能使用store参数将结果缓存。

## 六、消息通知

### 1、任务队列

可以使用列表类型实现队列，让生产者将任务使用LPUSH命令加入到某个键中，另一边让消费者不断地使用 RPOP命令从该键中取出任务即可。

同时可以借助BRPOP命令实现一旦有新任务加入到任务队列就通知消费者。

BRPOP和RPOP命令相似，唯一的区别是当列表中没有元素时BRPOP命令会一直阻塞住连接，直到有新元素加入。、

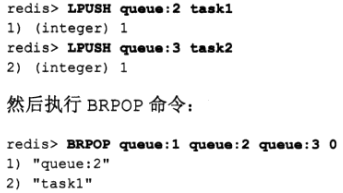
BRPOP接收两个参数，第一个是键名，第二个是超时时间，单位是秒。当超过了此时间仍然没有获得新元素的话就会返回nil。超时时间为“0”，表示不限制等待的时间，即如果没有新元素加入列表就会永远阻塞下去。

当获得一个元素后BRPOP命令返回两个值，分别是键名和元素值。  
除了BRPOP命令外，redis还提供了BLPOP，和BRPOP的区别在于从队列取元素时BLPOP会从队列左边取。

### 2、优先级队列

BRPOP命令可以同时接收多个键，其完整的命令格式为BRPOP key [key…] timeout。意义是同时检测多个键，如果所有键都没有元素则阻塞，如果其中有一个键有元素则会从该键中弹出元素。

如果多个键都有元素则按照从左到右的顺序取第一个键中的一个元素，如下：



借此特性可以实现区分优先级的任务队列。

### 3、“发布/订阅”模式

“发布/订阅”模式中包含两种角色，分别是发布者和订阅者。订阅者可以订阅一个或若干个频道（channel），而发布者可以向指定的频道发送消息，所有订阅此频道的订阅者都会收到此消息。

发布消息：PUBLISH channel message

返回值表示收到这条消息的订阅者数量。发出去的消息不会被持久化，也就是说当有客户端订阅channel后只能收到后续发布到该频道的消息，之前发送的就收不到了。

订阅消息：SUBSCRIBE channel [channel…]

可以同时订阅多个频道；执行subscribe命令后客户端会进入订阅状态，处于此状态下客户端不能使用除SUBSCRIBE/UNSUBSCRIBE/PSUBSCRIBE/PUNSUBSCRIBE这4个属于“发布/订阅”模式的命令之外的命令，否则会报错。

进入订阅状态后客户端可能收到三种类型的回复，每种类型的回复都包含3个值。

1. subscribe类型，表示订阅成功的反馈信息。第二个值是订阅成功的频道名称，第三个值是当前客户端订阅的频道数量。
2. message类型，表示接收到的消息。第二个值表示产生消息的频道名称，第三个是消息的内容
3. unsubscribe类型，表示成功取消订阅某个频道。第二个值是对应的频道名称，第三个值设计当前客户端订阅的频道数量，当此值为0时客户端会退出订阅状态，之后就可以执行其他非“发布/订阅”模式的命令了。

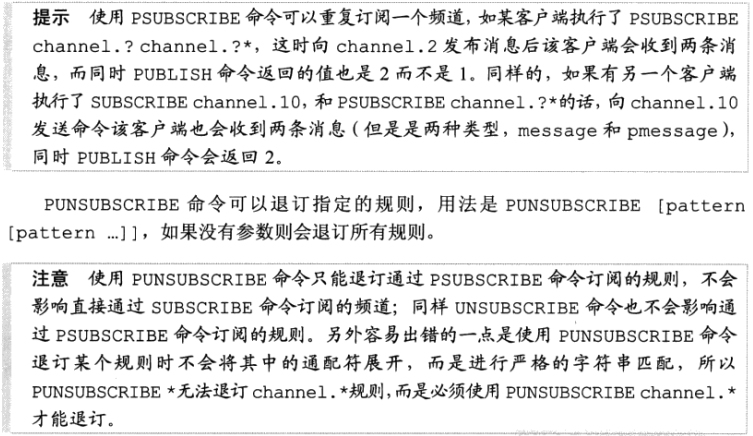
取消订阅指定的频道：unsubscribe [channel…]。如果不指定频道则会取消订阅所有频道。

**按照规则订阅**

可以使用PSUBSCRIBE命令订阅指定的规则，规则支持glob风格通配符格式。例如

PSUBSCRIBE channel.?\*

第一个值表示这条消息是通过PSUBSCRIBE命令订阅频道而收到的，第二个值表示订阅时使用的通配符，第三个值表示实际收到消息的频道命令，第四个则是消息内容。



## 七、管道

客户端和redis使用tcp协议连接。Redis的底层通信协议对管道（pipelining）提供了支持。通过管道可以一次性发送多条命令并在执行完后一次性将结果返回，当一组命令中每条命令都不依赖于之前命令的执行结果时就可以将这组命令一起通过管道发出。管道通过减少客户端与Redis的通信次数来实现降低往返时延累计值的目的。

## 八、节省空间

### 1、精简键名和键值

### 2、内部编码优化

在配置文件中有\*-max-ziplist-entries，\*-max-ziplist-value这样的参数（单位为字节）；当小于这个设置值时，会以ziplist的类型存在，所占空间更小，查找效率也可以接受。当大于此值时，由ziplist转换为\*\*类型，以实现更高的查找效率。

## 九、管理

### 1、持久化

<https://blog.csdn.net/weixin_30273501/article/details/96183635>

#### RDB方式

RDB方式的持久化是通过快照（snapshotting）完成的，当符合一定条件时Redis会自动将内存中的所有数据进行快照并存储在硬盘上。进行快照的条件可以由用户在配置文件中自定义，由两个参数构成：时间和改动的键的个数。当在指定的时间内被更改的键的个数大于指定的数值时就会进行快照。RDB是Redis默认采用的持久化方式

在RDB方式中，save参数指定了快照条件，可以存在多个条件，条件之间是“或”的关系。如下：

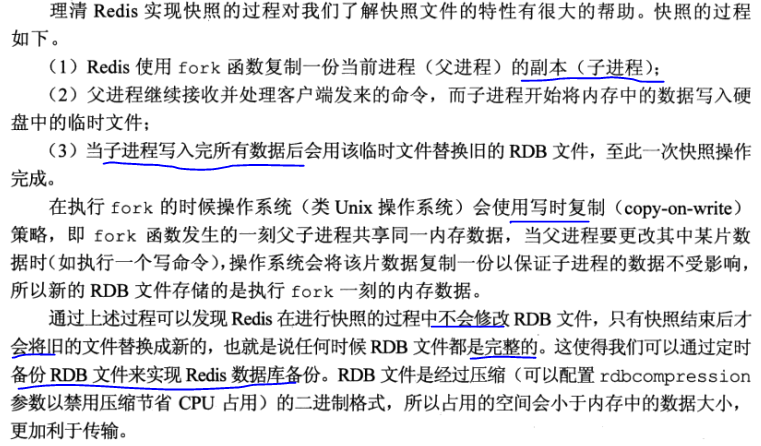
save 900 1

save 300 10

save 60 10000

save 900 1的意思是在15分钟（900秒钟）内有至少一个键被更改则进行快照。如果想要禁用自动快照，只需要将所有的save参数删除即可。

Redis默认会将快照文件存储在当前目录的dump.rdb文件中，可以通过配置dir和dbfilename两个参数分别指定快照文件的存储路径和文件名。



除了自动快照，还可以手动发送save或bgsave命令让redis进行快照，两个命令区别在于，前者是由主进程进行快照操作，会阻塞其他请求，后者会通过fork子进程进行快照操作。

Redis启动后会读取rdb快照文件，将数据从硬盘载入到内存。通常将一个记录一千万个字符串类型键、大小为1gb的快照文件载入到内存中需要花费20~30秒钟（因数据量大小、结构和服务器性能而异）。

通过rdb方式实现持久化，一旦redis异常退出，就会丢失最后一次快照以后更改的所有数据。

#### AOF（append onlyfile）方式

默认情况下reids没有开启AOF方式的持久化，可以通过appendonly参数开启：appendonly yes

开启AOF持久化后每执行一条会更改redis中的数据的命令，redis就会将该命令写入硬盘中的AOF文件。保存位置也是通过dir参数设置的，文件名可以通过appendfilename参数修改：appendfilename appendonly.aof

AOF文件是纯文本文件，其内容正是redis客户端向redis发送的原始通信协议的内容。这个时候就要注意，当频繁操作Redis中的数据时，AOF格式文件所占内存就会越来越大，而且会有很多没用的指令，如连续执行set foo 1，set foo 2，set foo 3，那么前两条指令是没有意义的，Redis很强大，它会自动将没有意义的指令删除。即每当达到一定条件时redis就会自动重写、优化AOF文件，这个条件可以在配置文件中设置：

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

除了让redis自动执行重写外，还可以主动使用BGREWRITEAOF命令手动执行AOF重写。

重写的过程只和内存中的数据有关，和之前的aof文件无关，这与rdb很相似，只不过二者的文件格式完全不同。

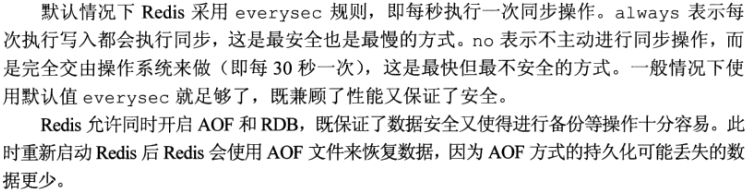
在启动时redis会逐个执行AOF文件中的命令来将硬盘中的数据载入到内存中，载入的速度相较RDB会慢一些。

需要注意的是虽然每次执行更改数据库内容的操作时，AOF都会将命令记录在AOF文件中，但是事实上，由于操作系统的缓存机制，数据并没有真正地写入硬盘，而是进入了系统的硬盘缓存。在默认情况下系统每30秒会执行一次同步操作，以便将硬盘缓存中的内容真正地写入硬盘，在这30秒的过程中如果系统异常退出则会导致硬盘缓存中的数据丢失。一般来讲启用AOF持久化的应用都无法容忍这样的损失，这就需要redis在写入AOF文件后主动要求系统将缓存内容同步到硬盘中。在redis中可以通过appendfsync参数设置同步的时机：

#appendfsync always

appendfsync everysec

#appendfsync no



### 2、复制（即主、从）

为了避免单点故障，可以将数据库复制多个副本以部署在不同的服务器上，即使有一台服务器出现故障其他服务器依然可以继续提供服务。也就是当一台服务器上的数据库更新后，可以自动将更新的数据同步到其他服务器上，redis提供了复制（replication）功能可以自动实现同步的过程。

#### 1）配置

主数据库可以进行读写操作，当发生写操作是自动将数据同步给从数据库。而从数据库一般是只读的，并接受主数据库同步过来的数据。一个主数据库可以拥有多个从数据库，而一个从数据库只能拥有一个主数据库。

只需在从数据库的配置文件中加入“slaveof <masterip> <masterport>”即可，主数据库无需进行任何配置。

加上slaveof参数启动另一个Redis实例作为从数据库，并让其监听6380端口：

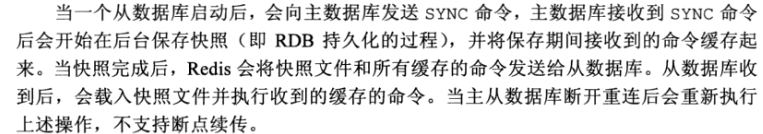
redis-server –port 6380 --slaveof 127.0.0.1 6379

可以通过设置从数据库的配置文件中的slave-read-only为no以使从数据库可写，但是对从数据库的任何更改都不会同步给任何其他数据库，并且一旦主数据库中更新了对应的数据就会覆盖从数据库中的改动。

除了通过配置文件或命令行参数设置slaveof参数，还可以在运行时使用SLAVEOF命令修改：——SLAVEOF 127.0.0.1 6379

如果该数据库已经是其他主数据库的从数据库了，SLAVEOF命令会停止和原来数据库的同步转而和新数据库同步。还可以使用SLAVEOF NO ONE来使当前数据库停止接收其他数据库的同步转成主数据库。

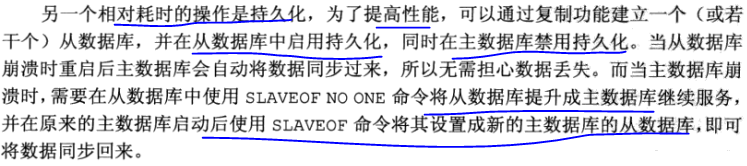
#### 2）原理



**读写分离**

可以通过复制功能建立多个从数据库，主数据库只进行写操作，而从数据库负责读操作。

**从数据库持久化**



### 3、安全

#### 1）Bind参数

Redis的默认配置会接受来自任何地址发送来的请求，要更改这一设置，在配置文件中修改bind参数，如：bind 127.0.0.1

band参数只能绑定一个地址，如果想更自由地访问规则需要通过防火墙来完成。

#### 2）数据库密码

还可以通过配置文件中的requirepass参数为redis设置一个密码。

客户端每次连接到redis时都需要发送密码，否则redis会拒绝执行客户端发来的命令，发送密码需要使用AUTH命令。

注意：配置redis复制的时候如果主数据库设置了密码，需要在从数据库的配置文件中通过masterauth参数设置主数据库的密码，以使从数据库连接主数据库时自动使用AUTH命令认证。

#### 3）命名命令

redis支持在配置文件中将命令重命名，比如将FLUSHALL命令重命名成一个比较复杂的名字，以保证只有自己的应用可以使用该命令：

rename-command FLUSHALL ofkllsafmlfalsfn,kklfl

如果希望直接禁用某个命令可以将命令重命名成空字符串

rename-command FLUSHALL “”

### 通信协议

