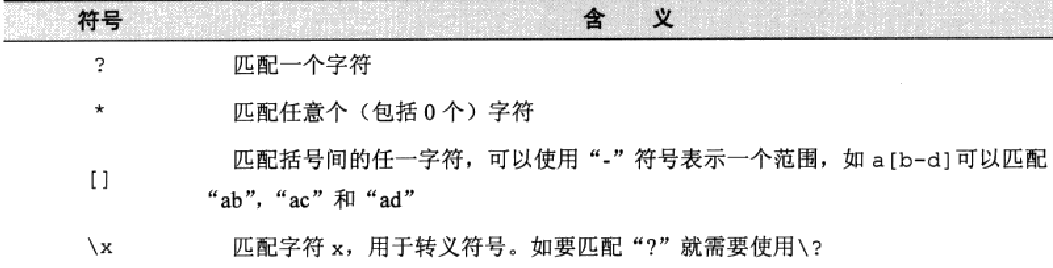
**一：热身**

获得符合规则的健名列表：keys pattern

pattern支持glob风格的通配符，具体规则如下表：



Redis命令不区分大小写。keys命令需要遍历Redis中的所有健，当键的数量较多时会影响性能，不建议在生产环境中使用。

判断一个键是否存在：exists key

如果键存在则返回整数类型1，否则返回0。

删除键：del key [key ...]

返回值是删除键的个数，del命令的参数不支持通配符。

获取键值的数据类型：type key

返回值可能是string（字符串）、hash（哈希表）、list（列表）、set（集合）和zset（有序集合）。

在Redis中每个键都属于一个明确的数据类型，如通过hset命令创建的健是散列类型，通过set命令建立的健是字符串类型等。使用一种数据类型的命令操作另一种数据类型的健会提示错误：”(error) WRONGTYPE Operation against a key holding the wrong kind of value”

所有Redis命令都是原子操作。

**二：字符串**

字符串类型是Redis中最基本的数据类型，它能存储任何形式的字符串，一个字符串类型键允许存储的数据的最大容量是512 MB。

字符串类型是其他4种数据类型的基础，其他数据类型和字符串类型的差别从某种角度来说只是组织字符串的形式不同。例如，列表类型是以列表的形式组织字符串，而集合类型是以集合的形式组织字符串。

赋值与取值：

set key value

get key

取值时，如果key不存在，则返回空结果

递增数字：incr key

当key对应的value是整数形式的字符串时，incr命令可以使其递增，并返回递增后的值。当要操作的key不存在时会默认value为0，所以第一次递增后的结果是1。当value不是整数形式是，Redis会提示错误：(error) ERR value is not an integer or out of range。

增加指定的整数：incrby key increment

incrby命令与incr命令基本一样，只是提供了increment参数指定增加的数值，该命令返回递增后的值。

减少指定的整数：

decr key

decrby key decrement

decr命令与incr命令用法相同，只不过是让value递减。decrby类似，只不过指定了减少的值。

增加指定浮点数：incrbyfloat key increment

使key对应的value增加一个双精度浮点数。

向尾部追加值：append key value

向key对应的键值的末尾追加value。如果键不存在，则将该键的值设置为value，返回值是追加后字符串的总长度。

获取字符串长度：strlen key

strlen命令返回键值的长度，如果key不存在则返回0.

同时设置/获取多个键值

mget key [key...]

mset key value [key value ...]

mget/mset类似于get/set，只是mget/mset可以同时获得/设置多个键的键值。

位操作

getbit key offset

setbit key offset value

bitcount key [start] [end]

bitop operation destkey key [key ...]

一个字节由8个二进制位组成，Redis提供了上述命令直接操作二进制位。

getbit命令可以获得一个字符串类型value指定位置的二进制位的值，索引从0开始，如果需要获取的二进制位的索引offset超出了value的二进制位的实际长度，则返回0。

setbit命令可以设置字符串类型value指定位置的二进制位的值，返回值是该位置的旧值。若要设置的位置超过了value的二进制位的长度，setbit命令会自动将中间的二进制位设置为0，同理设置一个不存在的键的指定二进制位的值会自动将其前面的位赋值为0。

bitcount命令可以获得字符串类型value中值是1的二进制位个数。可以通过参数来限制统计的字节范围，如只希望统计前两个字节，则命令为：bitcount foo 0 1。

bitop命令可以对多个字符串类型value进行位运算，并将结果存储在destkey对应的value中。bitop命令支持的运算操作有and、or、xor和not。

**三：散列类型**

Redis是采用字典结构以key-value的形式存储数据的，而散列类型（hash）的value也是一种字典结构，其存储了字段（field）和字段值（field value）的映射，但字段值只能是字符串，不支持其他数据类型，换句话说，散列类型不能嵌套其他的数据类型。散列类型的key可以包含至多2^32 - 1个字段。

其实，除了散列类型，Redis的其他数据类型同样不支持数据类型嵌套。比如集合类型的每个元素都只能是字符串，不能是另一个集合或散列表等。

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。例如要存储ID为2的汽车对象，可以分别使用名为color，name和price的3个字段来存储该辆汽车的颜色、名称和价格。存储结构如下图所示：



赋值与取值：

hset key filed value

hget key field

hmset key field value [field value ...]

hmget key field [field ...]

hgetall key

hset命令用来给字段赋值，而hget用来取字段的值，hset命令不区分插入和更新操作，因此修改数据时不用事先判断字段是否存在，当执行的是插入操作时（之前字段不存在）hset命令会返回1，当执行的是更新操作时（之前字段己经存在）会返回0。更进一步，当键本身不存在时，hset命令还会自动创建它。比如：

127.0.0.1**:**6379**>** hset car price 500

**(**integer**)** 1

127.0.0.1**:**6379**>** hset car name bmw

**(**integer**)** 1

127.0.0.1**:**6379**>** hget car price

"500"

127.0.0.1**:**6379**>** hget car name

"bmw"

当需要同时设置或获取多个字段的值时，可以使用hmset或hmget命令：

127.0.0.1**:**6379**>** hmset car color black price 300

OK

127.0.0.1**:**6379**>** hmget car name color price

1**)** "bmw"

2**)** "black"

3**)** "300"

如果想获取键中所有字段和字段值却不知道键中有哪些字段时，可以使用hgetall命令：

127.0.0.1**:**6379**>** hgetall car

1**)** "price"

2**)** "300"

3**)** "name"

4**)** "bmw"

5**)** "color"

6**)** "black"

判断字段是否存在：hexists key field

如果字段存在，则返回1，否则返回0，如果键不存在，也返回0.

当字段不存在时赋值：hsetnx key field value

hsetnx命令与hset命令类似，区别在于如果字段己经存在，hsetnx命令不执行任何操作。该命令返回1表示在hash中新建了一个字段并赋值value，返回0表示字段已经存在，且未执行任何操作。hsetnx命令时原子操作。

增加数字：hincrby key field increment

使字段值增加指定的整数。该命令返回增加后的值。如果key或者field不存在，则该命令自动创建该key或field，并赋值为increment。

删除字段：hdel key field [field ...]

hdel命令删除一个或多个字段，返回值是被删除的字段个数。

只获取字段名或字段值：

hkeys key

hvals key

获取字段数量： hlen key

**四：列表类型**

列表类型list可以存储多个字符串，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。不过使用链表的代价是通过索引访问元素比较慢。借助列表类型，Redis还可以作为队列使用。与散列类型的key最多能容纳的字段数量相同，列表类型的key最多能容纳2^32 - 1个元素。

向列表两端添加元素：

lpush key value [value ...]

rpush key value [value ...]

lpush命令用来向列表左边增加若干元素，rpush则用于向列表右边增加元素，返回值表示增加元素后列表的长度。

从列表两端弹出元素：

lpop key

rpop key

lpop命令从列表左边弹出一个元素，包括两部操作：一是将列表左边的元素从列表中移除，二是返回被移除的元素值。rpop命令类似，只不过是从列表右端弹出元素。

获取列表中元素的个数

llen key

当key不存在时返回0。llen的时间复杂度是O(1)。

获取列表片段

lrange key start stop

lrange命令返回索引从start到stop之间的所有元素，包含两端的元素。Redis的列表起始索引为0。

lrange命令支持负索引，表示从右边开始计算序数，-1表示最右边的第一个元素，以此类推。因此 lange key 0 -1可以获取列表中的所有元素。

如果start的索引位置比stop的索引位置靠后，则会返回空列表。如果stop大于实际的索引范围，则会返回到列表最右边的元素。

删除列表中指定的值：lrem key count value

lrem命令删除列表中前count个值为value的元素，返回值是实际删除的元素个数。根据count值的不同，lrem命令的执行方式会略有差异：

当count > 0时lrem命令会从列表左边开始删除前count个值为value的元素；

当count < 0时lrem命令会从列表右边开始删除前|count|个值为value的元素；

当count = 0时lrem命令会删除所有值为value的元素。例如：

127.0.0.1**:**6379**>** lpush list 1 1 2 1 3 1 2

**(**integer**)** 7

127.0.0.1**:**6379**>** lrange list 0 **-**1

1**)** "2"

2**)** "1"

3**)** "3"

4**)** "1"

5**)** "2"

6**)** "1"

7**)** "1"

127.0.0.1**:**6379**>** lrem list 3 1

**(**integer**)** 3

127.0.0.1**:**6379**>** lrange list 0 **-**1

1**)** "2"

2**)** "3"

3**)** "2"

4**)** "1"

127.0.0.1**:**6379**>** lrem list **-**3 2

**(**integer**)** 2

127.0.0.1**:**6379**>** lrange list 0 **-**1

1**)** "3"

2**)** "1"

127.0.0.1**:**6379**>** lrem list 0 1

**(**integer**)** 1

127.0.0.1**:**6379**>** lrange list 0 **-**1

1**)** "3"

获取/设置指定索引的元素值：

lindex key index

lset key index value

lindex命令返回指定索引的元素，索引从0开始；如果index为负数，则表示从右边开始计算的索引，最右边的索引是-1。如果index超出了范围，则返回” (nil)”。

lset将索引为index的元素赋值为value。如果index超出了范围，则报错：”(error) ERR index out of range”。

只保留列表指定片段：ltrim key start end

ltrim命令可以删除指定索引范围之外的所有元素，其指定范围的方法和lrange相同。

向列表中插入元素：linsert key BEFORE|AFTER pivot value

linsert命令首先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。linsert命令返回插入后列表的元素个数。

将元素从一个列表转到另一个列表：rpoplpush source destination

rpoplpush先执行rpop，然后执行lpush命令，从source列表的右边弹出一个元素，然后加入到destination列表的左边，并返回这个元素的值。整个过程是原子的。

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

**五：集合类型**

集合（set）中的每个元素都是不同的，且没有顺序。一个集合类型的key可以存储至多2^32 - 1个字符串。集合类型在Redis内部使用值为空的散列表实现的，所以插入、删除元素这些操作的时间复杂度都是O(1)。

增加/删除元素：

sadd key member [member ...]

srem key member [member ...]

sadd命令用来向集合中增加一个或多个元素，如果key不存在则会自动创建。因为在一个集合中不能有相同的元素，所以如果要加入的元素己经存在于集合中就会忽略这个元素。本命令的返回值是成功加入的元素数量（忽略的元索不计算在内）。

srem命令用来从集合中删除一个或多个元素，并返回删除成功的个数。

获得集合中的所有元素：smembers key

smembers命令会返回集合中的所有元素.

判断元素是否在集合中：sismember key member

判断一个元素是否在集合中是一个时间复杂度为O(1)的操作，无论集合中有多少个元素，sismember命令始终可以极快地返回结果。当member存在时该命令返回1，当member不存在或key不存在时返回0。

集合运算：

sdiff key [key ...]

sinter key [key ...]

sunion key [key ...]

sdiff命令用来对多个集合进行差集运算。集合A与集合B的差集表示A-B，代表所有属于A且不属于B的元素构成的集合。比如：

127.0.0.1**:**6379**>** sadd set1 1 2 3

**(**integer**)** 3

127.0.0.1**:**6379**>** sadd set2 2 3 4

**(**integer**)** 3

127.0.0.1**:**6379**>** sdiff set1 set2

1**)** "1"

127.0.0.1**:**6379**>** sdiff set2 set1

1**)** "4"

sinter命令用于多个集合执行交集运算。集合A与集合B的交集，代表所有属于A且属于B的元素构成的集合。

sunion命令用来对多个集合执行并集运算。集合A与集合B的并集，代表所有属于A或属于B的元素构成的集合。

进行集合运算并将结果存储:

sdiffstore destination key [key ...]

sinterstore destination key [key ...]

sunionstore destination key [key ...]

这些命令的功能与sdiff、sinter和sunion一样，唯一的区别就是这些命令会将结果存储在destination键中。

获得集合中元素个数：scard key

随机获得集合中的元素：srandmember key [count]

srandmember命令用来随机从集合中获取元素：

当count为正数时，srandmember会随机从集合里获得count个不重复的元素，如果count的值大于集合中的元素个数，则会返回集合中的全部元素；

当count为负数时，srandmember会随机从集合里获得|count|个的元素，这些元素有可能相同。比如：

127.0.0.1**:**6379**>** sadd letters a b c d

**(**integer**)** 4

127.0.0.1**:**6379**>** srandmember letters 2

1**)** "a"

2**)** "b"

127.0.0.1**:**6379**>** srandmember letters 10

1**)** "c"

2**)** "d"

3**)** "a"

4**)** "b"

127.0.0.1**:**6379**>** srandmember letters **-**2

1**)** "d"

2**)** "b"

127.0.0.1**:**6379**>** srandmember letters **-**2

1**)** "d"

2**)** "d"

127.0.0.1**:**6379**>** srandmember letters **-**10

1**)** "a"

2**)** "d"

3**)** "d"

4**)** "c"

5**)** "a"

6**)** "b"

7**)** "c"

8**)** "b"

9**)** "d"

10**)** "b"

从集合中弹出一个元素：spop key

集合中的元素是无序的，所以spop命令是从集合中随机选择一个元素弹出。

**六：有序集合**

在集合类型的基础上，有序集合类型为集合中的每个元素都关联了一个分数，这使得我们不仅可以完成插入、删除和判断元索是否存在等集合类型支持的操作，还能够获得分数最高（或最低）的前N个元素、获得指定分数范围内的元素等与分数有关的操作。虽然有序集合中每个元素都是不同的，但是它们的分数却可以相同。

有序集合类型是使用散列表和跳跃表（Skip list）实现的，所以即使读取位于中间部分的数据速度也很快，时间复杂度是O(log N)。

增加元素：zadd key score member [score member ...]

zadd命令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用新的分数替换原有的分数。zadd命令的返回值是新加入到集合中的元素个数。

分数可以是整数，也可以是双精度浮点数。+inf和-inf表示正无穷和负无穷。

获得元素的分数：zscore key member

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（包括两端的元素）。与lrange命令类似，索引是从0开始的，负数代表从后向前查找，-1表示最后一个元素。比如：

127.0.0.1**:**6379**>** zadd scoreboard 89 tom 67 peter 100 david

**(**integer**)** 3

127.0.0.1**:**6379**>** zrange scoreboard 0 2

1**)** "peter"

2**)** "tom"

3**)** "david"

127.0.0.1**:**6379**>** zrange scoreboard 1 **-**1

1**)** "tom"

2**)** "david"

如果需要同时获得元素的分数的话，可以在zrange命令的尾部加上withscores参数，比如：

127.0.0.1**:**6379**>** zrange scoreboard 0 **-**1 withscores

1**)** "peter"

2**)** "67"

3**)** "tom"

4**)** "89"

5**)** "david"

6**)** "100"

zrange命令的时间复杂度为O(log n + m)，其中n为有序集合的基数，m为返回的元素个数。如果两个元素的分数相同，Redis会按照元素本身的字典顺序进行排列。

zrevrange命令和zrange唯一不同在于zrevrange命令是按照元素分数从大到小的顺序给出结果的。

获得指定分数范围的元素：

zrangebyscore key min max [withscores] [limit offset count]

zrangebyscore命令按照元素分数从小到大的顺序，返同分数在min和max之间（包含min和max）的元素：

127.0.0.1**:**6379**>** zrangebyscore scoreboard 80 100

1**)** "tom"

2**)** "david"

如果希望分数范围不包含端点值，可以在分数前加“(”符号。比如上例中，如果希望分数的范围是可以含80分，但不包含100分，则：

127.0.0.1**:**6379**>** zrangebyscore scoreboard 80 **(**100

1**)** "tom"

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。比如希望得到所有分数高于80分，且不包含80分的人的名单，则：

127.0.0.1**:**6379**>** zrangebyscore scoreboard **(**80 **+**inf

1**)** "tom"

2**)** "david"

withscores参数的用法与zrange命令一样，不再赘述。

该命令中的limit offset count用法与SQL中的用法基本相同，即在获得的元素列表的基础上向后偏移offset个元素，并且只获取前count个元素。比如要获得分数高于60分的，从第2个人开始的3个人：

127.0.0.1**:**6379**>** zadd scoreboard 56 jerry 92 wendy 67 tvonne

**(**integer**)** 3

127.0.0.1**:**6379**>** zrange scoreboard 0 **-**1 withscores

1**)** "jerry"

2**)** "56"

3**)** "peter"

4**)** "67"

5**)** "tvonne"

6**)** "67"

7**)** "tom"

8**)** "89"

9**)** "wendy"

10**)** "92"

11**)** "david"

12**)** "100"

127.0.0.1**:**6379**>** zrangebyscore scoreboard 60 **+**inf limit 1 3

1**)** "tvonne"

2**)** "tom"

3**)** "wendy"

如果想获取分数低于或等于100分的前3个人，可以使用zrevrangebyscore命令实现：

127.0.0.1**:**6379**>** zrevrangebyscore scoreboard 100 0 limit 0 3

1**)** "david"

2**)** "wendy"

3**)** "tom"

增加某个元素的分数：zincrby key increment member

zincrby命令增加一个元素的分数，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，Redis在执行命令前会先建立它并将它的分数赋为0再执行操作。比如：

127.0.0.1**:**6379**>** zincrby scoreboard 4 jerry

"60"

获得集合中元素数量：zcard key

获得指定分数范围内的元素个数：zcount key min max

删除一个或多个元素：zrem key member [member ...]

该命令返回成功删除的元素数量。

按照排名范围删除元素：zremrangebyrank key start stop

zremrangebyrank命令按照元素分数从小到大的顺序，删除处在指定排名范围内的所有元素，并返回删除的元素个数，比如：

127.0.0.1**:**6379**>** zadd testrem 1 a 2 b 3 c 4 d 5 e 6 f

**(**integer**)** 6

127.0.0.1**:**6379**>** zremrangebyrank testrem 0 2

**(**integer**)** 3

127.0.0.1**:**6379**>** zrange testrem 0 **-**1

1**)** "d"

2**)** "e"

3**)** "f"

按照分数范围删除元素：zremrangebyscore key min max

zremrangebyscore命令会删除指定分数范围内的所有元素，返回值是删除的元素数量。比如：

127.0.0.1**:**6379**>** zremrangebyscore testrem **(**4 5

**(**integer**)** 1

127.0.0.1**:**6379**>** zrange testrem 0 **-**1

1**)** "d"

2**)** "f"

获得元素的排名：

zrank key member

zrevrank key member

zrank命令会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）。zrevrank 命令则相反（分数最大的元素排名为0）。

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键中元素的分数是由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。例如：

127.0.0.1**:**6379**>** zadd sortedsets1 1 a 2 b

**(**integer**)** 2

127.0.0.1**:**6379**>** zadd sortedsets2 10 a 20 b

**(**integer**)** 2

127.0.0.1**:**6379**>** zinterstore sortedsetsres 2 sortedsets1 sortedsets2

**(**integer**)** 2

127.0.0.1**:**6379**>** zrange sortedsetsres 0 **-**1 withscores

1**)** "a"

2**)** "11"

3**)** "b"

4**)** "22"

当aggregate是min时，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最小值。例如：

127.0.0.1**:**6379**>** zinterstore sortedsetsres 2 sortedsets1 sortedsets2 aggregate min

**(**integer**)** 2

127.0.0.1**:**6379**>** zrange sortedsetsres 0 **-**1 withscores

1**)** "a"

2**)** "1"

3**)** "b"

4**)** "2"

当aggregate是max时，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最大值。

zinterstore命令还能够通过weights参数设置每个集合的权重，每个集合在参与计算时，元素的分数会被乘上该集合的权重。例如：

127.0.0.1**:**6379**>** zinterstore sortedsetsres 2 sortedsets1 sortedsets2 weights 1 0.1

**(**integer**)** 2

127.0.0.1**:**6379**>** zrange sortedsetsres 0 **-**1 withscores

1**)** "a"

2**)** "2"

3**)** "b"

4**)** "4"

另外还有一个命令与zinterstore命令的用法一样，名为zunionstore，它的作用是计算集合间的并集，这里不再赘述。