**一：事务**

1：概述

Redis中的事务（transaction）是一组命令的集合。事务同命令一样都是Redis的最小执行单位，一个事务中的命令要么都执行，要么都不执行。

事务的原理是是先将属于一个事务的所有命令都发送给Redis，然后再让Redis依次执行这些命令。比如：

127.0.0.1**:**6379**>** multi

OK

127.0.0.1**:**6379**>** sadd aset 2

QUEUED

127.0.0.1**:**6379**>** sadd aset 1

QUEUED

127.0.0.1**:**6379**>** exec

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

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

上面的代码演示了事务的使用方式。首先使用multi命令告诉Redis：“下面我发给你的命令属于同一个事务，你先不要执行，而是把它们暂时存起来”。Redis回答：“OK”。

当把所有在同一个事务中要执行的命令都发给Redis后，使用exec命令告诉Redis，将事务队列中的所有命令按照发送顺序依次执行。exec命令的返回值就是这些命令的返回值组成的列表，返回值顺序和命令的顺序相同。

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，即使此后客户端断链也没关系，因为Redis中己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打断。

2：错误处理

a、语法错误，比如：

127.0.0.1**:**6379**>** multi

OK

127.0.0.1**:**6379**>** set a 1

QUEUED

127.0.0.1**:**6379**>** set b

**(**error**)** ERR wrong number of arguments **for** 'set' command

127.0.0.1**:**6379**>** errcommand c

**(**error**)** ERR unknown command 'errcommand'

127.0.0.1**:**6379**>** exec

**(**error**)** EXECABORT Transaction discarded because of previous errors**.**

跟在multi命令后执行了3个命令：一个正确的命令，成功加入到事务队列中；其余两个命令都有语法错误。只要有一个命令有语法错误，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。

b、运行错误。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。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），示例如下：

127.0.0.1**:**6379**>** multi

OK

127.0.0.1**:**6379**>** set a 1

QUEUED

127.0.0.1**:**6379**>** sadd a 2

QUEUED

127.0.0.1**:**6379**>** set a 3

QUEUED

127.0.0.1**:**6379**>** exec

1**)** OK

2**)** **(**error**)** WRONGTYPE Operation against a key holding the wrong kind of value

3**)** OK

127.0.0.1**:**6379**>** get a

"3"

可见虽然命令”sadd a 2”出现了错误，但是”set a 3”依然执行了。

Redis的事务没有关系数据库事务提供的回滚功能。为此开发者必须在事务执行出错后自己收拾剩下的摊子（将数据库复原回事务执行前的状态等）。不过由于Redis不支持回滚功能，也使得Redis在事务上可以保持简洁和快速。

3：watch命令

watch命令可以监控一个或多个键，一旦有一个键被修改（或删除），整个的事务就不会执行。监控一直持续到exec命令。比如：

127.0.0.1**:**6379**>** set b 1

OK

127.0.0.1**:**6379**>** set a 1

OK

127.0.0.1**:**6379**>** watch a

OK

127.0.0.1**:**6379**>** set a 2

OK

127.0.0.1**:**6379**>** multi

OK

127.0.0.1**:**6379**>** set b 2

QUEUED

127.0.0.1**:**6379**>** set a 3

QUEUED

127.0.0.1**:**6379**>** set b 4

QUEUED

127.0.0.1**:**6379**>** exec

**(**nil**)**

127.0.0.1**:**6379**>** get a

"2"

127.0.0.1**:**6379**>** get b

"1"

上例中，在执行watch命令后，事务执行前，修改了a的值，所以，最后事务中的命令”set a 3”没有执行，而且命令”set b 2”和”set b 4”也没有执行。exec命令返回空结果。

执行exec命令后会取消对所有键的监控，也可以用unwatch命令来取消监控。

**二：生存时间**

在Redis中可以使用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命令的使用方法为：”expire key seconds”，其中seconds表示键的生存时间，单位是秒。expire命令返回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）。如果ttl命令返回-2，表明该key不存在，如果ttl返回-1，表明该key存在，但是没有关联超时时间。比如：

127.0.0.1**:**6379**>** set foo 1

OK

127.0.0.1**:**6379**>** expire foo 20

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

127.0.0.1**:**6379**>** ttl foo

**(**integer**)** 18

127.0.0.1**:**6379**>** ttl foo

**(**integer**)** 10

127.0.0.1**:**6379**>** ttl foo

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

127.0.0.1**:**6379**>** set bar 2

OK

127.0.0.1**:**6379**>** ttl bar

**(**integer**)** **-**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。如果生存时间被成功清除则返回1；否则返回0（因为键不存在或键本来就是永久的）。比如：

127.0.0.1**:**6379**>** set foo 1

OK

127.0.0.1**:**6379**>** ttl foo

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

127.0.0.1**:**6379**>** expire foo 20

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

127.0.0.1**:**6379**>** ttl foo

**(**integer**)** 18

127.0.0.1**:**6379**>** persist foo

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

127.0.0.1**:**6379**>** ttl foo

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

除了persist命令之外，使用set或getset命令为键赋值也会清除键的生存时间，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。比如：

127.0.0.1**:**6379**>** set foo 1

OK

127.0.0.1**:**6379**>** expire foo 20

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

127.0.0.1**:**6379**>** ttl foo

**(**integer**)** 18

127.0.0.1**:**6379**>** set foo 2

OK

127.0.0.1**:**6379**>** ttl foo

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

127.0.0.1**:**6379**>** expire foo 20

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

127.0.0.1**:**6379**>** ttl foo

**(**integer**)** 17

127.0.0.1**:**6379**>** incr foo

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

127.0.0.1**:**6379**>** ttl foo

(integer) 12

expire命令的seconds参数必须是整数，所以最小单位是1秒。如果想要更精确的控制键的生存时间应该使用pexpire命令，pexpire命令与expire命令的唯一区别是前者的时间单位是毫秒，即pexpire key 1000与expire key 1等价。

另外，pttl命令以毫秒为单位返回键的剩余时间。

还有两个命令expireat和pexpireat。expireat命令与expire命令的差别在于前者使用绝对时间表示键的生存时间。pexpireat命令与expireat命令的区别是前者的时间单位是毫秒。比如：

127.0.0.1**:**6379**>** set foo 1

OK

127.0.0.1**:**6379**>** expireat foo 1448972556

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

127.0.0.1**:**6379**>** ttl foo

**(**integer**)** 186

127.0.0.1**:**6379**>** pexpireat foo 1448972556000

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

127.0.0.1**:**6379**>** ttl foo

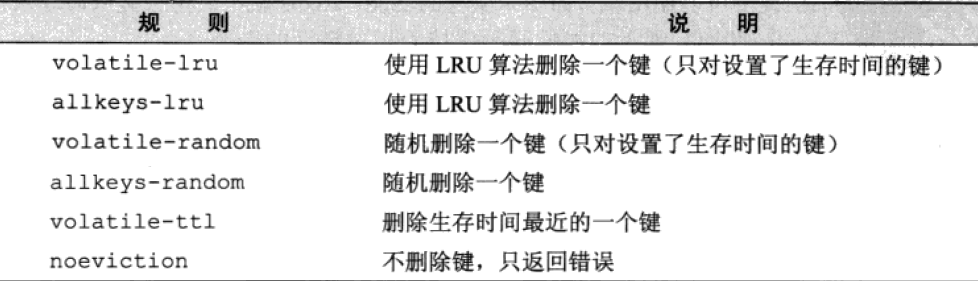
**(**integer**)** 165

将Redis作为缓存使用时，当服务器内存有限时，如果大量地使用缓存键且生存时间设置得过长就会导致Redis占满内存；另一方面如果为了防止Redis占用内存过大而将缓存键的生存时间设得太短，就可能导致缓存命中率过低并且大量内存白白地闲置。

实际开发中会发现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内存。

maxmemory-policy支持的规则如下表所示：



其中的LRU（Least Recently Used）算法即“最近最少使用”，其认为最近最少使用的键在未来一段时间内也不会被用到，当需要空间时这些键是可以被删除的。

**三：排序**

1：sort命令

有序集合常见的使用场景是大数据排序，如游戏的玩家排行榜等。除了使用有序集合外，还可以借助Redis的sort命令，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join查询相类似的任务。比如：

127.0.0.1**:**6379**>** sadd set 1 4 5 2 3 9 10 **-**1

**(**integer**)** 8

127.0.0.1**:**6379**>** lpush list 1 4 5 2 3 9 10 **-**1

**(**integer**)** 8

127.0.0.1**:**6379**>** zadd zset 10.0 1 2.4 4 3.2 5 10.1 2 9.7 3 2.1 9 1.1 10 2.9 **-**1

**(**integer**)** 8

127.0.0.1**:**6379**>** sort set

1**)** "-1"

2**)** "1"

3**)** "2"

4**)** "3"

5**)** "4"

6**)** "5"

7**)** "9"

8**)** "10"

127.0.0.1**:**6379**>** sort list

1**)** "-1"

2**)** "1"

3**)** "2"

4**)** "3"

5**)** "4"

6**)** "5"

7**)** "9"

8**)** "10"

127.0.0.1**:**6379**>** sort zset

1**)** "-1"

2**)** "1"

3**)** "2"

4**)** "3"

5**)** "4"

6**)** "5"

7**)** "9"

8**)** "10"

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，就像这样：

127.0.0.1**:**6379**>** sadd set2 a 2 b z e 3 x f 9

**(**integer**)** 9

127.0.0.1**:**6379**>** sort set2

**(**error**)** ERR One or more scores can't be converted into double

127.0.0.1**:**6379**>** sort set2 alpha

1**)** "2"

2**)** "3"

3**)** "9"

4**)** "a"

5**)** "b"

6**)** "e"

7**)** "f"

8**)** "x"

9**)** "z"

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参数来返回指定范围的结果。用法和sql语句一样，limit offset count，表示跳过前offset个元素，并获取之后的count个元素。比如：

127.0.0.1**:**6379**>** sort set2 alpha desc

1**)** "z"

2**)** "x"

3**)** "f"

4**)** "e"

5**)** "b"

6**)** "a"

7**)** "9"

8**)** "3"

9**)** "2"

127.0.0.1**:**6379**>** sort set2 alpha desc limit 0 3

1**)** "z"

2**)** "x"

3**)** "f"

127.0.0.1**:**6379**>** sort set2 alpha desc limit 2 3

1**)** "f"

2**)** "e"

3**)** "b"

2：by参数

sort命令支持by参数，by参数的语法为“by 参考健”。其中参考键可以是字符串类型键或者是散列类型key的某个field（表示为key->field)。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，而是针对每个元素，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。就像这样：

127.0.0.1**:**6379**>** sadd set a b c d e

**(**integer**)** 5

127.0.0.1**:**6379**>** hset a time 5

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

127.0.0.1**:**6379**>** hset b time 1

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

127.0.0.1**:**6379**>** hset c time 7

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

127.0.0.1**:**6379**>** hset d time 2

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

127.0.0.1**:**6379**>** hset e time 10

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

127.0.0.1**:**6379**>** hset f time 9

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

127.0.0.1**:**6379**>** sort set by **\*->**time

1**)** "b"

2**)** "d"

3**)** "a"

4**)** "c"

5**)** "e"

上例中，对集合set进行排序，但是不在按照集合中元素本身的值进行排序，而是根据by参数，针对set中的每个元素，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。也就是根据散列类型a、b、c、d、e的time字段的值进行排序。

除了散列类型之外，参考键还可以是字符串类型，比如：

127.0.0.1**:**6379**>** lpush list a b c d e

**(**integer**)** 5

127.0.0.1**:**6379**>** mset a 100 b 2 c 1 d 90 e 28 f 19

OK

127.0.0.1**:**6379**>** sort list by **\***

1**)** "c"

2**)** "b"

3**)** "e"

4**)** "d"

5**)** "a"

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）。例如：

127.0.0.1**:**6379**>** sort list by hehe

1**)** "e"

2**)** "d"

3**)** "c"

4**)** "b"

5**)** "a"

例子中hehe是常量键名（甚至hehe键可以不存在），此时sort的结果与lrange的结果相同，没有执行排序操作。在不需要排序但需要借助sort命令获得与元素相关联的数据时，常量键名是很有用的。

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。像这样：

127.0.0.1**:**6379**>** lpush list g

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

127.0.0.1**:**6379**>** set g 2

OK

127.0.0.1**:**6379**>** sort list by **\***

1**)** "c"

2**)** "b"

3**)** "g"

4**)** "e"

5**)** "d"

6**)** "a"

例子中，元素g的参考键：字符串g，和元素b的参考键：字符串b，它们的值相同，都是2，因此，sort命令会再比较”g”和”b”元素本身的大小来决定两者的顺序。

当某个元素的参考键不存在时，会默认参考键的值为0：

127.0.0.1**:**6379**>** lpush list h

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

127.0.0.1**:**6379**>** sort list by **\***

1**)** "h"

2**)** "c"

3**)** "b"

4**)** "g"

5**)** "e"

6**)** "d"

7**)** "a"

3：get参数

sort命令支持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索自身的值，而是get参数指定的键值。get参数的规则和by参数一样，get参数也支持字符串类型和散列类型的键，并使用”\*”作为占位符。比如：

127.0.0.1**:**6379**>** sadd set a b c d e

**(**integer**)** 5

127.0.0.1**:**6379**>** hmset a time 5 title hello

OK

127.0.0.1**:**6379**>** hmset b time 1 title world

OK

127.0.0.1**:**6379**>** hmset c time 7 title this

OK

127.0.0.1**:**6379**>** hmset d time 2 title is

OK

127.0.0.1**:**6379**>** hmset e time 10 title a

OK

127.0.0.1**:**6379**>** hmset f time 9 title redis

OK

127.0.0.1**:**6379**>** sort set by **\*->**time get **\*->**title

1**)** "world"

2**)** "is"

3**)** "hello"

4**)** "this"

5**)** "a"

127.0.0.1**:**6379**>** sort list by **\*** get **\***

1**)** **(**nil**)**

2**)** "1"

3**)** "2"

4**)** "2"

5**)** "28"

6**)** "90"

7**)** "100"

一个sort命令中可以有多个get参数（而by参数只能有一个），有n个get参数，每个元素返回的结果就有n行。如果还需要返回元素本身，则可以使用：”get #”。比如：

127.0.0.1**:**6379**>** sort set by **\*->**time get **\*->**time get **\*->**title get #

1**)** "1"

2**)** "world"

3**)** "b"

4**)** "2"

5**)** "is"

6**)** "d"

7**)** "5"

8**)** "hello"

9**)** "a"

10**)** "7"

11**)** "this"

12**)** "c"

13**)** "10"

14**)** "a"

15**)** "e"

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。 保存后的键的类型为列表类型，如果键己经存在则会覆盖它。加上store参数后sort命令的返回值为结果的个数。比如：

127.0.0.1**:**6379**>** sort set by **\*->**time get **\*->**time get **\*->**title get # store reslist

**(**integer**)** 15

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

1**)** "1"

2**)** "world"

3**)** "b"

4**)** "2"

5**)** "is"

6**)** "d"

7**)** "5"

8**)** "hello"

9**)** "a"

10**)** "7"

11**)** "this"

12**)** "c"

13**)** "10"

14**)** "a"

15**)** "e"

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命令的时间复杂度是O(n + m log m)，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较大的时候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。所以开发中使用sort命令时需要注意以下几点：

a、尽可能减少待排序键中元素的数量（使n尽可能小）；

b、使用limit参数只获取需要的数据（使m尽可能小）；

c、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。

**四：消息通知**

1：任务队列

队列可以用Redis的列表类型实现，比如使用lpush和rpop命令就可以实现队列的概念。要实现任务队列，只需要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。

使用rpop命令时，消费者需要不断的定期轮训队列，查看队列中是否有新的任务，也就是要定期调用一次rpop命令。其实借助brpop命令更方便一些，它是阻塞版本的rpop命令，唯一的区别是当列表中没有元素时，brpop命令会阻塞住连接，直到有新元素加入。

brpop命令接收两个参数，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返回nil。如果超时时间置为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。

举例如下，首先在终端A上调用brpop命令，这里的list甚至可以不存在：

127.0.0.1**:**6379**>** brpop list 0

此时该命令会阻塞住。然后，在终端B中执行lpush命令：

127.0.0.1**:**6379**>** lpush list hehe

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

此时，在终端A上，brpop命令才会有输出：

127.0.0.1**:**6379**>** brpop list 0

1**)** "list"

2**)** "hehe"

使用brpop命令，还可以实现一个优先级队列。brpop命令可以同时接收多个键，其完整的命令格式为：brpop key key [key ...] timeout

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。

举例如下，首先在终端A上调用brpop命令：

127.0.0.1**:**6379**>** brpop list1 list2 0

此时终端A阻塞，然后在终端B上输入命令：

127.0.0.1**:**6379**>** lpush list2 this

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

此时，在终端A上输出：

127.0.0.1**:**6379**>** brpop list1 list2 0

1**)** "list2"

2**)** "this"

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。比如，首先在终端B上调用命令：

127.0.0.1**:**6379**>** lpush list1 l1

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

127.0.0.1**:**6379**>** lpush list2 l2

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

然后，在终端A上，调用brpop命令：

127.0.0.1**:**6379**>** brpop list2 list1 0

1**)** "list2"

2**)** "l2"

可见只返回list2的消息。借此特性可以实现优先级队列，一旦list2中有消息，无论list1中有多少消息，都是首先返回list2的消息。

2：发布和订阅

除了实现任务队列外，Redis还提供了一组命令可以实现“发布/订阅”（publish/subscribe）模式。“发布/订阅”模式也是进程间的消息传递方式，其原理是：“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。

发布者发布消息的命令是publish，用法是publish channel message，比如：

127.0.0.1**:**6379**>** publish channel1 hi

**(**integer**)** 0

这样消息就发出去了。publish命令的返回值表示接收到这条消息的订阅者数量。因为此时没有客户端订阅channel1，所以返回0。发出去的消息不会被持久化，也就是说当有客户端订阅channel1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。

订阅频道的命令是subscribe，可以同时订阅多个频道，用法是：

subscribe channel [channel ...]。比如在终端A上输入：

127.0.0.1**:**6379**>** subscribe channel1

Reading messages**...** **(**press Ctrl**-**C to quit**)**

1**)** "subscribe"

2**)** "channel1"

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

执行subscribe命令后客户端会进入订阅状态（这里表现为阻塞状态，等待接收消息），进入订阅状态后客户端可收到三种类型的回复。每种类型的回复都包含3个值，第一个值是消息的类型，根据消息类型的不同，第二、三个值的含义也不同。消息类型可能的取值有：

a：subscribe，表示订阅成功的反馈信息。此时，比如上面的例子中，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。

b：message，表示接收到的消息。第二个值表示产生消息的频道名称，第三个值是消息的内容。比如，在另一个终端上，输入：

127.0.0.1**:**6379**>** publish channel1 hi

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

向channel1中发布一个消息”hi”，该命令返回2，表示当前有两个客户端订阅该频道，每个客户端的返回的内容都是：

**...**

1**)** "message"

2**)** "channel1"

3**)** "hi

c：unsubscribe，表示成功取消订阅某个频道，此时第二个值是对应的频道名称，第三个值是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。

客户端进入订阅状态之后，就只能执行subscribe/unsubscribe/psubscribe/punsubscribe这4种命令，不能执行“发布/订阅”模式之外的命令。但是在redis-cli中，进入订阅状态后，终端就处于接收消息状态，无法再执行任何命令，因此这种限制应该是针对编程客户端而言的。

使用psubscribe命令，可以指定订阅的规则，规则支持glob风格的通配符格式，比如在终端A中输入：

psubscribe channel**.?\***

Reading messages**...** **(**press Ctrl**-**C to quit**)**

1**)** "psubscribe"

2**)** "channel.?\*"

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

此时，在终端B中执行下列命令：

127.0.0.1**:**6379**>** publish channel.1 "this is channel.1"

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

127.0.0.1**:**6379**>** publish channel.10 "this is channel.10"

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

127.0.0.1**:**6379**>** publish channel.1234 "this is channel.1234"

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

此时，终端A上的输出是：

1**)** "pmessage"

2**)** "channel.?\*"

3**)** "channel.1"

4**)** "this is channel.1"

1**)** "pmessage"

2**)** "channel.?\*"

3**)** "channel.10"

4**)** "this is channel.10"

1**)** "pmessage"

2**)** "channel.?\*"

3**)** "channel.1234"

4**)** "this is channel.1234"

消息的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，第四个值则是消息内容。

使用psubscribe命令可以重复订阅同一个频道，如某客户端执行了”psubscribe channel.? channel.?\*”，这时向channel.2发布消息后该客户端会收到两条消息，而同时publish命令返回的值也是2而不是1。同样的，如果客户端执行了”subscribe channel.10”和”psubscribe channel.?\*”的话，向channel.10发送命令该客户端也会收到两条消息（但是是两种类型，message和pmessage）。

punsubscribe命令可以退订指定的规则，用法是” punsubscribe [pattern pattern]”，如果没有参数则会退订所有规则。

注意使用punsubscribe命令只能退订通过psubscribe命令订阅的规则，不会影响通过subscribe命令订阅的频道；同样unsubscribe命令也不会影响通过psubscribe命令订阅的规则。

**五：管道**

客户端和Redis使用TCP协议连接。不论客户端向Redis发送命令还是Redis向客户端返回命令的执行结果，都需要经过网络传输，在执行多个命令时，即使命令不需要上一条命令的执行结果，每条命令都需要等待上一条命令执行完才能执行。

Redis的底层通信协议对管道提供了支持。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的目的。管道在各种编程语言的客户端中都得到了支持。