Redis的键空间通知(keyspace notifications)功能是自2.8.0版本开始加入的，客户端可以通过订阅/发布(Pub/Sub)机制，接收那些以某种方式改变了Redis数据空间的事件通知。比如：所有改变给定key的命令；所有经过lpush操作的key；所有在0号数据库中过期的key等等。

通知是通过Redis的订阅/发布机制发送的，因此，所有支持订阅/发布功能的客户端都可在无需调整的情况下，使用键空间通知功能。

Redis的发布/订阅目前是即发即弃(fire and forget)模式的，因此无法实现事件的可靠通知。也就是说，如果发布/订阅的客户端断链之后又重连，则在客户端断链期间的所有事件都丢失了。

未来计划支持事件的可靠通知，但是这可能会通过让订阅与发布功能本身变得更可靠来实现，也可能会在Lua脚本中对消息的订阅与发布进行监听，从而实现类似将事件推入到列表这样的操作。

**一：事件的类型**

针对改变Redis数据空间的每个操作，键空间通知都会发送两类不同的事件。

比如在0号数库中，执行del mykey操作，将会触发两个消息，等价于执行下面两个publish命令：

PUBLISH \_\_keyspace@0\_\_**:**mykey del

PUBLISH \_\_keyevent@0\_\_**:**del mykey

一个频道发布0号数据库中，所有针对mykey键执行的操作，这类事件，以keyspace为前缀，称为keyspace通知；

另一个频道发布0号数据库中，所有成功执行del操作的键，这类事件，以keyevent为前缀，称为keyevent通知；

在上面的例子中，当执行del mykey时，会发生：

a：keyspace频道的订阅者会收到消息，消息中包含键执行的操作的名字；

b：keyevent频道的订阅者会收到消息，消息中包含执行某种操作的键的名字；

可以通过配置，使Redis仅发送某一类我们感兴趣的通知。

**二：配置**

因键空间通知功能需要耗费一定的CPU时间，因此默认情况下，该功能是关闭的。可以通过修改配置文件redis.conf，或者通过CONFIG SET命令，设置notify-keyspace-events选项，来启用或关闭该功能。

该选项的值为空字符串时，该功能禁用，选项值为非空字符串时，启用该功能，非空字符串由特定的多个字符组成，每个字符表示不同的意义：

K：keyspace事件，事件以\_\_keyspace@<db>\_\_为前缀进行发布；

E：keyevent事件，事件以\_\_keyevent@<db>\_\_为前缀进行发布；

g：一般性的，非特定类型的命令，比如del，expire，rename等；

$：字符串特定命令；

l：列表特定命令；

s：集合特定命令；

h：哈希特定命令；

z：有序集合特定命令；

x：过期事件，当某个键过期并删除时会产生该事件；

e：驱逐事件，当某个键因maxmemore策略而被删除时，产生该事件；

A：g$lshzxe的别名，因此”AKE”意味着所有事件。

注意，该选项的值中至少需要包含K或者E，否则不会发布任何事件。比如，如果需要开启针对列表的keyspace事件通知，则该选项需要配置为“Kl”；

**三：不同命令产生的事件通知**

DEL 命令为每个被删除的键产生一个 del 事件；

RENAME 产生两个事件：为源键产生一个 rename\_from 事件，并为目标键产生一个 rename\_to 事件；

EXPIRE命令，在设置键的过期时间时产生一个 expire事件；当键因过期而被删除时，产生一个 expired事件；

SORT命令，在带有 STORE 参数时产生一个 sortstore事件。如果 STORE 指示的用于保存排序结果的键已经存在，则原键会被删除，因此还会产生一个 del 事件；

SET 以及它的所有变种(SETEX、SETNX和GETSET)都产生set事件。另外，SETEX命令还会产生expire 事件；

MSET 命令，为每个键产生一个 set 事件；

SETRANGE 产生一个 setrange 事件；

INCR 、DECR、INCRBY和DECRBY都产生 incrby 事件；

INCRBYFLOAT产生incrbyfloat事件；

APPEND产生append事件；

LPUSH和LPUSHX都产生单个 lpush 事件，即使有多个输入元素时，也是如此；

RPUSH 和 RPUSHX 都产生单个rpush事件，即使有多个输入元素时，也是如此；

RPOP 产生 rpop 事件，如果被弹出的元素是列表的最后一个元素，那么还会产生一个 del 事件；

LPOP 产生 lpop 事件，如果被弹出的元素是列表的最后一个元素，那么还会产生一个 del 事件；

LINSERT 产生一个 linsert 事件；

LSET 产生一个 lset 事件；

LREM产生一个lrem事件，如果该命令执行之后，列表键被清空，则还会产生一个 del 事件；

LTRIM 产生一个ltrim事件，如果该命令执行之后，列表键被清空，则还会产生一个 del 事件；

RPOPLPUSH 和 BRPOPLPUSH 产生一个 rpop 事件，以及一个 lpush 事件。两个命令都保证rpop事件在 lpush 事件之前发出。如果弹出元素之后，列表键被清空，则还会产生一个 del 事件；

HSET 、 HSETNX 和 HMSET 都只产生一个 hset 事件；

HINCRBY 产生一个 hincrby 事件；

HINCRBYFLOAT 产生一个 hincrbyfloat 事件；

HDEL 产生一个 hdel 通知。如果执行该命令之后，哈希键被清空，则还会产生一个del事件；

SADD 产生一个 sadd 事件，即使有多个输入元素时，也是如此；

SREM 产生一个 srem 事件，如果执行该命令之后，集合键被清空，则还会产生一个 del 事件；

SMOVE 为源键产生一个 srem 事件，并为目标键产生一个sadd 事件；

SPOP 产生一个 spop 事件。如果执行该命令之后，集合键被清空，则还会产生一个 del 事件；

SINTERSTORE、SUNIONSTORE和SDIFFSTORE分别产生 sinterstore、sunionostore和sdiffstore 三种事件。如果用于保存结果的键已经存在，则还会产生一个 del 事件；

ZINCR产生一个 zincr 事件；

ZADD 产生一个 zadd事件，即使有多个输入元素时，也是如此；

ZREM 产生一个 zrem 通知，即使有多个输入元素时，也是如此。如果执行 ZREM 之后，有序集合键被清空，则还会产生一个 del 事件；

ZREMEBYSCORE 产生一个 zrembyscore事件，如果用于保存结果的键已经存在，则还会产生一个 del 事件。

ZREMBYRANK 产生一个 zrembyrank事件，如果用于保存结果的键已经存在，则还会产生一个 del 事件。

ZINTERSTORE 和 ZUNIONSTORE 分别产生 zinterstore 和 zunionstore 两种事件。如果用于保存结果的键已经存在，那么还会产生一个 del 事件。

每当一个键因为过期而被删除时，产生一个 expired 事件。

每当一个键因为 maxmemory策略而被删除并回收内存时，产生一个 evicted 事件。

注意：所有命令都只在键真的被改动了之后，才会产生事件通知。比如，当srem命令试图删除不存在于集合的元素时，删除操作执行失败，因为没有真正的改动键，所以这一操作不会发送通知。

**四：expired事件通知的发送时间**

Redis 使用以下两种方式删除过期的键：

a：当一个键被访问时，程序会对这个键进行检查，如果键已过期，则删除该键；

b：系统会在后台定期扫描并删除那些过期的键；

当过期键被以上两种方式中的任意一种发现并且删除时，才会产生expired事件通知。

Redis不保证生存时间（TTL）变为 0 的键会立即被删除：如果没有命令访问这个键，或者设置生存时间的键非常多的话，那么在键的生存时间变为0，到该键真正被删除，这中间可能会有一段比较显著的时间间隔。

因此，Redis产生expired事件通知的时间，是过期键被删除的时候，而不是键的生存时间变为 0 的时候。

**五：实践**

1：Redis发布所有通知，客户端A订阅所有消息：

127.0.0.1**:**6379**>** config set notify**-**keyspace**-**events KEA

OK

127.0.0.1**:**6379**>** psubscribe \_\_key**\***@0\_\_**:\***

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

1**)** "psubscribe"

2**)** "\_\_key\*@0\_\_:\*"

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

然后，在客户端B上执行set和del命令：

127.0.0.1**:**6379**>** set msg "hello"

OK

127.0.0.1**:**6379**>** del msg

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

127.0.0.1**:**6379**>** del msg

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

然后客户端A的打印如下：

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "set"

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyevent@0\_\_:set"

4**)** "msg"

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "del"

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyevent@0\_\_:del"

4**)** "msg"

可见，针对每一个操作，客户端A都收到了两种消息，分别是keyspace和keyevent消息。

2：使Redis仅发布keyspace通知，而客户端A订阅所有消息类型：

127.0.0.1**:**6379**>** config set notify**-**keyspace**-**events KA

OK

127.0.0.1**:**6379**>** psubscribe \_\_key**\***@0\_\_**:\***

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

1**)** "psubscribe"

2**)** "\_\_key\*@0\_\_:\*"

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

在客户端B上执行，与上面同样的步骤。此时，客户端A上的打印：

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "set"

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "del"

可见，尽管客户端A订阅了所有消息，但是Redis仅发布了keyspace事件。而且，在客户端B上执行了两次del操作，而只有第一个del成功执行了，从而产生了一个事件。

3：Redis发布所有通知，客户端A仅订阅keyspace消息：

127.0.0.1**:**6379**>** config set notify**-**keyspace**-**events KEA

OK

127.0.0.1**:**6379**>** psubscribe \_\_keyspace@0\_\_**:\***

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

1**)** "psubscribe"

2**)** "\_\_keyspace@0\_\_:\*"

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

在客户端B上，执行与上面同样的步骤。然后客户端A的打印如下：

1**)** "pmessage"

2**)** "\_\_keyspace@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "set"

1**)** "pmessage"

2**)** "\_\_keyspace@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "del"

可见，针对每一个操作，客户端A只收到了keyspace消息。

4：Redis仅发布字符串特定类型的通知，客户端A订阅所有类型的消息：

127.0.0.1**:**6379**>** config set notify**-**keyspace**-**events KE$

OK

127.0.0.1**:**6379**>** psubscribe \_\_key**\***@0\_\_**:\***

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

1**)** "psubscribe"

2**)** "\_\_key\*@0\_\_:\*"

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

在客户端B上，分别执行lpush，set和del命令：

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

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

127.0.0.1**:**6379**>** set msg "hello"

OK

127.0.0.1**:**6379**>** del msg

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

此时，客户端A上的打印：

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyspace@0\_\_:msg"

4**)** "set"

1**)** "pmessage"

2**)** "\_\_key\*@0\_\_:\*"

3**)** "\_\_keyevent@0\_\_:set"

4**)** "msg"

可见，针对字符串，产生了keyspace和keyevent类型的消息，而针对列表的lpush命令并没有产生消息，而且，del命令是非特定类型的命令，也没有产生消息。

参考：

http://www.phperz.com/article/15/0904/153987.html

http://redis.io/topics/notifications