Redis Keyspace Notifications

IMPORTANT Keyspace notifications is a feature available since 2.8.0

Feature overview

Keyspace notifications allow clients to subscribe to Pub/Sub channels in order to receive events affecting the Redis data set in some way.

Examples of events that can be received are:

- All the commands affecting a given key.
- All the keys receiving an LPUSH operation.
- All the keys expiring in the database 0.

Events are delivered using the normal Pub/Sub layer of Redis, so clients implementing Pub/Sub are able to use this feature without modifications.

Because Redis Pub/Sub is *fire and forget* currently there is no way to use this feature if your application demands **reliable notification** of events, that is, if your Pub/Sub client disconnects, and reconnects later, all the events delivered during the time the client was disconnected are lost.

In the future there are plans to allow for more reliable delivering of events, but probably this will be addressed at a more general level either bringing reliability to Pub/Sub itself, or allowing Lua scripts to intercept Pub/Sub messages to perform operations like pushing the events into a list.

Type of events

Keyspace notifications are implemented by sending two distinct types of events for every operation affecting the Redis data space. For instance a DEL operation targeting the key named `mykey` in database `0` will trigger the delivering of two messages, exactly equivalent to the following two PUBLISH commands:

```
PUBLISH __keyspace@0__:mykey del
PUBLISH __keyevent@0__:del mykey
```

It is easy to see how one channel allows us to listen to all the events targeting the key `mykey` and the other channel allows to obtain information about all the keys that are target of a `del` operation.

The first kind of event, with `keyspace` prefix in the channel is called a **Key-space notification**, while the second, with the `keyevent` prefix, is called a **Key-event notification**.

In the above example a `del` event was generated for the key `mykey`. What happens is that:

- The Key-space channel receives as message the name of the event.
- The Key-event channel receives as message the name of the key.

It is possible to enable only one kind of notification in order to deliver just the subset of events we are interested in.

Configuration

By default keyspace event notifications are disabled because while not very sensible the feature uses some CPU power. Notifications are enabled using the `notify-keyspace-events` of redis.conf or via the **CONFIG SET**.

Setting the parameter to the empty string disables notifications. In order to enable the feature a non-empty string is used, composed of multiple characters, where every character has a special meaning according to the following table:

```
K     Keyspace events, published with __keyspace@<db>__ prefix.
E     Keyevent events, published with __keyevent@<db>__ prefix.
g     Generic commands (non-type specific) like DEL, EXPIRE, RENAME,
$     String commands
l     List commands
s     Set commands
h     Hash commands
z     Sorted set commands
t     Stream commands
d     Module key type events
x     Expired events (events generated every time a key expires)
e     Evicted events (events generated when a key is evicted for maxm
m     Key miss events (events generated when a key that doesn't exist
A     Alias for "g$lshztxed", so that the "AKE" string means all the
```

At least K or E should be present in the string, otherwise no event will be delivered regardless of the rest of the string.

For instance to enable just Key-space events for lists, the configuration parameter must be set to `Kl`, and so forth.

The string `KEA` can be used to enable every possible event.

Events generated by different commands

Different commands generate different kind of events according to the following list.

- DEL generates a `del` event for every deleted key.
- RENAME generates two events, a `rename_from` event for the source key, and a `rename_to` event for the destination key.
- MOVE generates two events, a `move_from` event for the source key, and a `move_to` event for the destination key.
- COPY generates a `copy_to` event.
- MIGRATE generates a `del` event if the source key is removed.
- RESTORE generates a `restore` event for the key.
- EXPIRE and all its variants (PEXPIRE, EXPIREAT, PEXPIREAT) generate an `expire` event when called with a positive timeout (or a future timestamp). Note that when these commands are called with a negative timeout value or timestamp in the past, the key is deleted and only a `del` event is generated instead.
- SORT generates a `sortstore` event when `STORE` is used to set a new key. If the resulting list is empty, and the `STORE` option is used, and there was already an existing key with that name, the result is that the key is deleted, so a `del` event is generated in this condition.
- SET and all its variants (SETEX, SETNX,GETSET) generate `set` events. However SETEX will also generate an `expire` events.
- MSET generates a separate `set` event for every key.
- SETRANGE generates a `setrange` event.
- INCR, DECR, INCRBY, DECRBY commands all generate `incrby` events.
- INCRBYFLOAT generates an `incrbyfloat` events.
- APPEND generates an `append` event.
- LPUSH and LPUSHX generates a single `lpush` event, even in the variadic case.
- RPUSH and RPUSHX generates a single `rpush` event, even in the variadic case.
- RPOP generates an `rpop` event. Additionally a `del` event is generated if the key is removed because the last element from the list was popped.
- LPOP generates an `lpop` event. Additionally a `del` event is generated if the key is removed because the last element from the list was popped.
- LINSERT generates an `linsert` event.
- LSET generates an `lset` event.

- LREM generates an `lrem` event, and additionally a `del` event if the resulting list is empty and the key is removed.
- LTRIM generates an `ltrim` event, and additionally a `del` event if the resulting list is empty and the key is removed.
- RPOPLPUSH and BRPOPLPUSH generate an `rpop` event and an `lpush` event. In both cases the order is guaranteed (the `lpush` event will always be delivered after the `rpop` event). Additionally a `del` event will be generated if the resulting list is zero length and the key is removed.
- LMOVE and BLMOVE generate an `lpop/rpop` event (depending on the wherefrom argument) and an `lpush/rpush` event (depending on the whereto argument). In both cases the order is guaranteed (the `lpush/rpush` event will always be delivered after the `lpop/rpop` event). Additionally a `del` event will be generated if the resulting list is zero length and the key is removed.
- HSET, HSETNX and HMSET all generate a single `hset` event.
- HINCRBY generates an `hincrby` event.
- HINCRBYFLOAT generates an `hincrbyfloat` event.
- HDEL generates a single `hdel` event, and an additional `del` event if the resulting hash is empty and the key is removed.
- SADD generates a single `sadd` event, even in the variadic case.
- SREM generates a single `srem` event, and an additional `del` event if the resulting set is empty and the key is removed.
- SMOVE generates an `srem` event for the source key, and an `sadd` event for the destination key.
- SPOP generates an `spop` event, and an additional `del` event if the resulting set is empty and the key is removed.
- SINTERSTORE, SUNIONSTORE, SDIFFSTORE generate `sinterstore`, `sunionstore`, `sdiffstore` events respectively. In the special case the resulting set is empty, and the key where the result is stored already exists, a `del` event is generated since the key is removed.
- ZINCR generates a `zincr` event.
- ZADD generates a single `zadd` event even when multiple elements are added.
- ZREM generates a single `zrem` event even when multiple elements are deleted. When the resulting sorted set is empty and the key is generated, an additional `del` event is generated.
- ZREMBYSCORE generates a single `zrembyscore` event. When the resulting sorted set is empty and the key is generated, an additional `del` event is generated.
- ZREMBYRANK generates a single `zrembyrank` event. When the resulting sorted set is empty and the key is generated, an additional `del` event is generated.
- ZDIFFSTORE, ZINTERSTORE and ZUNIONSTORE respectively generate `zdiffstore`, `zinterstore` and `zunionstore` events. In the special case the resulting sorted set is

empty, and the key where the result is stored already exists, a `del` event is generated since the key is removed.

- XADD generates an `xadd` event, possibly followed an `xtrim` event when used with the `MAXLEN` subcommand.
- XDEL generates a single `xdel` event even when multiple entries are deleted.
- `XGROUP CREATE` generates an `xgroup-create` event.
- `XGROUP CREATECONSUMER` generates an `xgroup-createconsumer` event.
- `XGROUP DELCONSUMER` generates an `xgroup-delconsumer` event.
- `XGROUP DESTROY` generates an `xgroup-destroy` event.
- `XGROUP SETID` generates an `xgroup-setid` event.
- `XSETID` generates an `xsetid` event.
- XTRIM generates an `xtrim` event.
- PERSIST generates a `persist` event if the expiry time associated with key has been successfully deleted.
- Every time a key with a time to live associated is removed from the data set because it expired, an `expired` event is generated.
- Every time a key is evicted from the data set in order to free memory as a result of the `maxmemory` policy, an `evicted` event is generated.

IMPORTANT all the commands generate events only if the target key is really modified. For instance an SREM deleting a non-existing element from a Set will not actually change the value of the key, so no event will be generated.

If in doubt about how events are generated for a given command, the simplest thing to do is to watch yourself:

```
$ redis-cli config set notify-keyspace-events KEA
$ redis-cli --csv psubscribe '__key*__:*'
Reading messages... (press Ctrl-C to quit)
"psubscribe","__key*__:*",1
```

At this point use `redis-cli` in another terminal to send commands to the Redis server and watch the events generated:

```
"pmessage","__key*__:*","__keyspace@0__:foo","set"
"pmessage","__key*__:*","__keyevent@0__:set","foo"
...
```

Timing of expired events

Keys with a time to live associated are expired by Redis in two ways:

- When the key is accessed by a command and is found to be expired.
- Via a background system that looks for expired keys in the background, incrementally, in order to be able to also collect keys that are never accessed.

The `expired` events are generated when a key is accessed and is found to be expired by one of the above systems, as a result there are no guarantees that the Redis server will be able to generate the `expired` event at the time the key time to live reaches the value of zero.

If no command targets the key constantly, and there are many keys with a TTL associated, there can be a significant delay between the time the key time to live drops to zero, and the time the `expired` event is generated.

Basically `expired` events **are generated when the Redis server deletes the key** and not when the time to live theoretically reaches the value of zero.

Events in a cluster

Every node of a Redis cluster generates events about its own subset of the keyspace as described above. However, unlike regular Pub/Sub communication in a cluster, events' notifications **are not** broadcasted to all nodes. Put differently, keyspace events are node-specific. This means that to receive all keyspace events of a cluster, clients need to subscribe to each of the nodes.

History

- `>= 6.0`: Key miss events were added.
