XINFO GROUPs enhancements in Redis 7

XINFO GROUPS Enhancements in Redis 7

Redis 7 introduced two useful enhancements to the output of the **`XINFO GROUPS`** command. Let's take a moment to look at these. The new data items help us answer these questions about the state of a consumer group:

How many entries has each consumer group read?

Sometimes it it useful to know how many entries consumers in the consumer group have collectively read. This information is now available in Redis 7.

What's the current lag for each consumer group?

Think of a consumer group's lag as the number of entries that exist in the stream which have yet to be read by any consumer that is a member of the group.

We can observe changes in this value over time to determine whether entries are being produced faster than they are being consumed. In this case, we might want to use this information to add further consumer instances to the consumer group, so that it can catch up.

Once the lag falls beyond a certain point, we could then scale down the consumer group by removing some consumer instances from it.

Example

In the example that follows, we can see how to get the number of entries read and lag for a consumer group, and how these change as the stream is manipulated. To see the correct output from the **`XINFO GROUPS`** you must be using Redis 7. If you're using the course Docker Compose file, you have a Redis 7 instance.

First, check your Redis server version with the **`INFO SERVER`** command:

```
127.0.0.1:6379> INFO SERVER

# Server

redis_version:7.0.8

redis_git_sha1:00000000
```

```
redis_git_dirty:0
redis_build_id:c869ebfd8f51f71c
redis_mode:standalone
...
```

Here, I'm running Redis 7.0.8 so we're good to go!

First, let's use the `XADD` command to add 1000 entries to a stream. They'll all have the same data inside them, but that doesn't matter for this example:

```
127.0.0.1:6379> 1000 XADD mystream * hello world
"1677530033562-0"
"1677530033564-0"
"1677530033566-0"
"1677530033568-0"
"1677530033570-0"
"1677530033571-0"
```

```
...
```

If you choose to try this yourself then your stream entry IDs will of course differ, as they are timestamps generated by the Redis server at the point that **`XADD`** is run. This doesn't matter for this example, what's important is that we have a stream with `1000` entries in it. Let's verify that this is what we have just created:

```
127.0.0.1:6379> XLEN mystream

(integer) 1000
```

Next, we need to create a consumer group for this stream. Let's call it **`consumers`** and have it start at the beginning of the stream:

```
127.0.0.1:6379> XGROUP CREATE
mystream consumers 0

OK
```

Now we have a consumer group, we can run **`XINFO GROUPS`** and see what Redis 7 tells us about the state of the stream's consumer groups:

```
127.0.0.1:6379>  XINFO GROUPS
mystream
1)  1) "name"
    2) "consumers"
    3) "consumers"
    4) (integer) 0
    5) "pending"
    6) (integer) 0
    7) "last-delivered-id"
    8) "0-0"
    9) "entries-read"
   10) (nil)
   11) "lag"
   12) (integer) 1000
```

Here we can see the two additional data points that were added in Redis 7. **`entries-read`** is currently set to **`(nil)`** as no entries have been read (the counter is yet to be initialized). **`lag`** is currently set to **`1000`** as the consumer group **`consumers`** is currently 1000 messages behind in its processing of the stream.

If multiple consumer groups existed for this stream, we'd see similar output for each.

Let's have two consumers read some entries from the stream, then see how the output of **`XINFO GROUPS`** is affected...

```
127.0.0.1:6379> XREADGROUP GROUP
consumers consumer1 COUNT 1
STREAMS mystream >
1) 1) "mystream"
   2) 1) 1) "1677530033562-0"
         2) 1) "hello"
            2) "world"
127.0.0.1:6379> XREADGROUP GROUP
consumers consumer1 COUNT 1
STREAMS mystream >
```

```
1) 1) "mystream"
   2) 1) 1) "1677530033564-0"
         2) 1) "hello"
            2) "world"
127.0.0.1:6379> XREADGROUP GROUP
consumers consumer2 COUNT 1
STREAMS mystream >
1) 1) "mystream"
   2) 1) 1) "1677530033566-0"
         2) 1) "hello"
            2) "world"
127.0.0.1:6379> XINFO GROUPS
mystream
1)  1) "name"
    2) "consumers"
    3) "consumers"
    4) (integer) 2
    5) "pending"
```

```
     6) (integer) 3
     7) "last-delivered-id"
     8) "1677530033566-0"
     9) "entries-read"
    10) (integer) 3
    11) "lag"
    12) (integer) 997
```

Now we can see that the **`entries-read`** counter has updated to reflect the 3 entries that the consumers in the group have read between them. We also observe that the **`lag`** value has reduced by 3 to a new value of 997. We could monitor the value of **`lag`** over time and use it to determine whether or not we need to ramp up our throughput by adding more consumer instances to the consumer group for a while.

Note that the consumers do not need to acknowledge each stream entry read with the **`XACK`** command before the **`lag`** and **`entries-read`** counters are updated.

Further Resources

For more information, see the XINFO GROUPS command page on redis.io.

Optional Video

If you'd like to see the enhancements to `XINFO GROUPS` in action, check out the following optional video from the Simon's Things on Thursdays series that was live streamed to our YouTube channel.

Simon's Things on Thursdays: More Redis Streams! (Episode 2)

Course overview

☐ Lesson

Environment setup

Consumer groups

☐ Lesson

The problem with slow consumers

☐ Assessment

Quiz 1 | Redis streams consumer groups

☐ Lesson

Consumer groups

☐ Assessment

Quiz 2 | Redis streams consumer groups

☐ Lesson

Adding consumers to a group

☐ Assessment

Quiz 3| Redis streams consumer groups