RedisGears Python quick start tutorial

For this tutorial, you need:

- Either:
  - A Redis Enterprise cluster with the RedisGears module and Python plugin installed and enabled on a database
  - An OSS Redis database with the RedisGears module
- redis-cli with connectivity to a Redis database

RedisGears basics

In this quick start guide, we'll see how to use RedisGears to perform batch processing and event processing.

With RedisGears, **batch processing** means processing the data already stored in a Redis database. **Event processing** means processing changes to the Redis key space.

The examples below assume an empty Redis database.

Batch processing

Let's start with the simplest example. From the redis-cli, run the following command:

```bash
redis.cloud:6379> RG.PYEXECUTE "GearsBuilder().run()"
1) (empty array)
2) (empty array)
```

This command doesn't do much; it simply iterates over the keyspace. Let's add a key and run it again:

```bash
redis.cloud:6379> SET message "hello world"
OK
redis.cloud:6379> RG.PYEXECUTE "GearsBuilder().run()"
1) 1) "{event": None, 'key': 'message', 'type': 'string', 'value': 'hello world'}"
2) (empty array)
```

We've added a single string, and you can see that this gears function processes it, even though it does nothing with the data. Let's actually do something with the data. So first, we'll add a few more strings:

```bash
redis.cloud:6379> SET message:2 "hello galaxy"
OK
redis.cloud:6379> SET message:3 "hello universe"
OK
```

We now have three strings in our database. Suppose we want to perform a unique word count on these strings. We can write a RedisGears function to do this just that. So open a file called wordcount.py, and add the following code:
There are two ways to load files into RedisGears. For production deployments, we recommend using the special `gears-cli`. However, for the purpose of this demonstration, the easiest way is to pass the filename through the `redis-cli` command, like so:

```
$ redis-cli rg.pyexecute "\`cat wordcount.py`"
1) 1) "{'key': 'world', 'value': 1}"
   2) "{'key': 'galaxy', 'value': 1}"
   3) "{'key': 'hello', 'value': 3}"
   4) "{'key': 'universe', 'value': 1}"
2) (empty array)
```

The results here show the number of occurrences of each word in all of our strings. So, we've effectively processed the data in our Redis database all at once, in a batch.

### Event processing

You may have noticed that all of the RedisGears functions above end with a call to `run()`. This indicates that the function should be run immediately on the data in the Redis database. But what if you want to process data as it arrives in Redis? In that case, your functions will end with a call to `register()`, which will store the function and apply it as events occur in Redis.

Let's see how to register a function. First, suppose we're writing hashes to our database that represent users. They take the following form:

```
redis.cloud:6379> HSET person:3 name "Summer Smith" age 17
(integer) 2
redis.cloud:6379> HSET person:4 name "James Jameson" age 21
(integer) 2
```

Each hash has two fields, one containing a name and the other an age. Now, suppose we want to keep a record of the maximum age of all users. We can register a RedisGears function to do this. Open up a file called `maxage.py`, and add the following code:
def age(x):
    ''' Extracts the age from a person's record '''
    return int(x['value']['age'])

def compare_and_swap(x):
    ''' Checks and sets the current maximum '''
    k = 'age:maximum'
    v = execute('GET', k)  # read key's current value
    v = int(v) if v else 0  # initialize to 0 if None
    if x > v:  # if a new maximum found
        execute('SET', k, x)  # set key to new value

# Event handling function registration
gb = GearsBuilder()
gb.map(age)  # Extract the 'age' field from each hash
gb.foreach(compare_and_swap)  # Compare the max age to the value stored at age:maximum
gb.register('person:*')  # Only process keys matching the pattern 'person:*'

You can see here that we define two methods: age() and compare_and_swap(). Even if you're not familiar with Python, you should be able to see what the methods do.

Below the method definitions is the RedisGears data flow that we're defining. Notice that at the end we call register() to register the function to listen for events.

To load this function into RedisGears, run the following:

```bash
$ redis-cli RG.PYEXECUTE "`cat maxage.py`"
```

Now start the redis-cli, and create a couple of hashes:

```
redis.cloud:6379> HSET person:5 name "Marek Michalski" age 17
(integer) 2
redis.cloud:6379> HSET person:6 name "Noya Beit" age 21
(integer) 2
```

To see if the RedisGears function is working, check the value of age:maximum:

```
redis.cloud:6379> GET age:maximum
"21"
```

Next steps

You should now have a basic idea of how to run RedisGears functions for batch and event processing. But there's a lot more to RedisGears than this. To better understand it, see the RedisGears tutorial. If you're interested in write-behind caching, see our write-behind caching overview.

Updated: February 10, 2022