loopstud.io

Dr. Jose Scoseria 2605
Montevideo, 11300
www.loopstud.io

Ruby Concurrent Execution Queue

We need to build an execution queue that takes different commands from multiple clients (concurrently) and execute them one by one according to the command type. It needs to accept 3 types of commands:

1. Execute the job in a synchronous way and return the result (this job skips the queue).
2. Enqueue a job and return its unique identifier.
3. Enqueue a job in at least x seconds from now and return its unique identifier.

The server has to print the result of each job in the server log.

Task

Implement a multithreaded server that accepts multiple TCP connections. Clients can connect to the server and enqueue different kinds of jobs (defined in the server) that will run, according to the command used, in a synchronous or asynchronous way following a first come- first served criteria.

Constraints

- There has to be at least 2 job types.
- No Ruby gems allowed except RSpec.
- We recommend using Telnet to connect to the server.
- Unit Tests for all classes (RSpec).
- Add a Readme file documenting the features and how to use them.

Example client interface

Confidencial 2018

- perform_now JobClass job_params
- perform_later JobClass job_params
- perform_in seconds_from_now JobClass job_params

Example job classes

```ruby
class HelloWorldJob
  def perform
    'Hello World'
  end
end

class HelloMeJob
  def perform(first_name, last_name)
    "Hello #{first_name} #{last_name}"
  end
end
```

Example client usage

```
➜ ~ telnet localhost 2000
Trying ::1...
Connected to localhost.

perform_now HelloWorldJob
> Hello World
perform_later HelloWorldJob
> 123456
perform_in 10 HelloWorldJob
> 456789
```

Bonus

- Add a command called perform_at that takes a UNIX timestamp and execute the job at least at that date time.
- Add job retry logic. When a job fails (raises an exception) it should go back to the queue in at least 60 seconds, if it fails 3 times the job should be discarded.
- Support multiple queues. A queue name could be defined in each Job class and the jobs should be enqueued under those queues.
- Add a command called postpone that takes a number of seconds and a job id. The command should postpone the execution of that job to at least that amount of seconds.
- Add a command called get_info that returns info of a specific job. This should return information for jobs that are enqueued or waiting to be enqueued. The information should include params and execution time if applicable.

Confidencial 2018