Pham Huu Luan
Holistic Interview
April 17, 2017

Overview of Ledis - a lightweight Redis

Application: here

Choosing frontend

I quite unfamiliar with implementing a CLI, so first step I googled for some example. Then I discover web-cli. Look at its features I though this is kind of overkill.

Then I remember working with a nice, simple Ruby console: web-console, which is exactly what I need.

After I spend half an hour interate web-console into a Rails application. Web-console is not a front-end plugin I look into its source code, then pick the visual code that I need.

I have to read along the line and remove a bunch of thing Rails related like console session, key-word auto suggestion.

Choosing backend

I consider between Ruby and Elixir. Reason:

- Ruby: I familiar with it
- Elixir: Redis operation seem very functional-programming to me. And with the reputation of concurrency, I though Elixir would give greate performance to Ledis.

I spend nearly 1 day to make decision. But I realize, I don't have enough imfomation to decise. To be honest, I'm lost!!. Then I calm down and make a different approach: research more about Redis architecture

Here are interesting things I found about Redis:

1. Redis is single-threaded

before proper researching about redis, I was to implementing ledis with a multiprocess architecture in mind. The reason behind this is because I mostly use Ruby MRI. So to me, in other to scale throughput, I have to have multiple processes (along with some threading, but the point is I still have to use many processes).

The problem is each process has its own memory and is not shared to each other. So I wonder: how can Redis make sure that every requests have access to the same data storage? And this is when I realize, that Redis is actually single-threaded, and you can have concurrency on single-thread using an I/O multiplexing mechanism and an event loop

=> So I decided to go with Ruby to simpify the architecture

2. Every Redis command is atomic

Every command of Redis is atomic because of the fact that Redis is single-threaded. Ruby MRI have the GIL so that no more than 1 thread is run at a time. But this case is not equivalent to single-thread, because MRI actually have multi-thread, just no more than 1 thread run at a time -> must ensure single-thread. In other to do this, I config Puma server in single mode and have only 1 thread max

Ledis structure

Ledis class

This is the main class. Its objective is to hold a class variable *value_table*, which is hash. This hash will contain all keys and values

This hash is not shared with anything, so Ledis class also the place to implement the core logic.

Ledis::Command class

command_string inside request get parsed into *command* and *params*. A Ledis::Command object will be initialized with these infomation.

Objective of this class is

1. To check if this is a supported Ledis command.

2. Most of Ledis command are type-specifiec. This class will check if current command and key are compatible with each other.

3. Execute Ledis command. Those *command* and *params* will be passed to Ledis to perform command logic

Ledis::DataTypes

Please look at objective No.2 of Command. In other to know command-key capability, a naive way is get value of that key from the main datastore (the class variable hash) and know which type it is. Then look at a mapping between commans is the type it suppose to work with

I find this not very efficient, because:

- main datastore contain many info, so access key is slow
- have to invoke ruby code object.class each time.

Solution: create another class variable hash *type_table* which also have keys, but value is just in enum of data stype

With this solution, I was able to cut as much as half of the time for this command-key checking operation.

- Because type_table is lighter than value_table, It's only store in enum value
- Does not need to use Ruby to check type
- Downside: To cost more memory. But I think this is a good tradeoff, because I think we are going to read a lot more than write, and command-key checking operation is required for most of the command

If I had more time

There are lots of room for Ledis to be improved. If I had more time, I will:

- refactor front-end component to a more appopriate structure to allow static caching. Reason that I can not right now is web-console code is mainly a javascript file, with js.erb handler. Inside this file the author use Rails to render some html code, which is the UI component the CLI. An approach with BackboneJS view or ReactJS can archive the goal
- not using Rails at all. Because, again, this is overkill. a Rack application should be good enough. Right now I'm using Grape for API, which is also a Rack application.

Thankyou

Thank you for the opportunity. I’ve learned alot new things through this assignment, aside with strengthen my knowledge on concurrency.