V0.8

dRuby hands-on workshop

Masatoshi SEKI

What is dRuby?

- Distributed Object System
- Can invoke methods in different process
- Can send objects between process
- Pure Ruby

Exercise-1 Hello, World. *3*

Exercise-2 Key value store *4*

Exercise-3 Queue *5*

Exercise-4 SSE *7*

About Me *9*

Exercise-1 Hello, World.

Learn dRuby setup and Remote Method Invocation.

➡ list 1-1 hello_server.rb

Bind a URI to an object, and start the dRuby server thread. It's a server, so don't let it finish

```
require 'drb'

class Hello
  def greeting
    puts('Hello, World.')
  end
end

uri = 'druby://localhost:54000'
DRb.start_service(uri, Hello.new)
sleep
```

➡ list 1-2 hello_client.rb

Start the dRuby server thread, Create a proxy object, and send a message.

```
require 'drb'

DRb.start_service
uri = 'druby://localhost:54000'
it = DRbObject.new_with_uri(uri)
it.greeting
```

Step 1.
Run hello_server.rb on terminal 1.

➡ terminal 1

```
$ ruby hello_server.rb
```

Step 2.
Run hello_client.rb on terminal 2.

➡ terminal 2

```
$ ruby hello_client.rb
```

Then, "Hello, World." is displayed on terminal 1.

Q. Terminate the server then run the client. What happens?

Exercise-2 Key value store

Learn to exchange objects. This exercise uses 3 terminals. Output of irb is omitted.

Step 1.
Start the hash server.

➡ terminal 1

```
$ irb -r drb --simple-prompt
>> kvs = Hash.new
>> uri = 'druby://localhost:54320'
>> DRb.start_service(uri, kvs)
```

Step 2.
Store objects into the hash server.

➡ terminal 2

```
$ irb -r drb --simple-prompt
>> DRb.start_service
>> uri = 'druby://localhost:54320'
>> kvs = DRbObject.new_with_uri(uri)
>> kvs['greeting'] = 'hello, world'
>> kvs['stdout'] = $stdout
=> #<IO:<STDOUT>>
```

Step 3.
Retrieve objects.

➡ terminal 3

```
$ irb -r drb --simple-prompt
>> DRb.start_service
>> uri = 'druby://localhost:54320'
>> kvs = DRbObject.new_with_uri(uri)
>> kvs['greeting']
=> "hello, world"
>> kvs['stdout']
=> #<DRb::DRbObject:0x00... @uri="druby://172.17.0.3:46761", @ref=...>
```

Q. Let's store various kind object (eg. String, Integer, Hash, IO).

Q. What happens when kvs['stdout'].puts('hello, again') ?

Exercise-3 Queue

Synchronize process using queue.

Step 1.
Start the queue server.

➡ terminal 1

```
$ irb -r drb --simple-prompt
>> DRb.start_service('druby://localhost:54321', SizedQueue.new(1))
```

Step 2.
Run the consumer and the producer.

➡ terminal 2

```
$ irb -r drb --simple-prompt
>> DRb.start_service
>> uri = 'druby://localhost:54321'
>> queue = DRbObject.new_with_uri(uri)
```

➡ terminal 3

```
$ irb -r drb --simple-prompt
>> DRb.start_service
>> uri = 'druby://localhost:54321'
>> queue = DRbObject.new_with_uri(uri)
```

Step 3.
Let's coordinate two processes.

➡ terminal 3 (cont.)

```
>> queue.push("one")
>> queue.push(2.0) (* Block! queue is full. *)
```

➡ terminal 2 (cont.)

```
>> queue.pop
=> "one"
>> queue.pop
=> 2.0
>> queue.pop (* Block! queue is empty. *)
```

➡ terminal 3 (cont.)

```
>> queue.push(3)
```

➡ terminal 2 (cont.)

```
=> 3
>> queue.pop (* Block! queue is empty. *)
```

➡ terminal 3 (cont.)

```
>> queue.push(:four)
```

➡ terminal 2 (cont.)

```
=> :four
```

Q. Let's increase consumer, producer.

Q. Let's increase size of queue.

Q. Let's store various kind object (eg. String, Integer, Hash, IO).

Exercise-4 SSE

WEBrick, Server Sent Event and dRuby.

➡ list 4-1 sse.rb

```
require 'webrick'
require 'driq'
require 'driq/webrick'
require 'drb'

src = Driq::EventSource.new
svr = WEBrick::HTTPServer.new(:Port => 8086)
body = <<EOS
<!DOCTYPE html>
<html>
  <head>
    <title>SSE</title>
  </head>
  <body>
    <h1>It Works!</h1>
    <ul id="list">
    </ul>
  </body>
  <script>
  var evt = new EventSource('/stream');
  evt.onmessage = function(e) {
    var newElement = document.createElement("li");
    var eventList = document.getElementById('list');

    newElement.innerHTML = "message: " + e.data;
    eventList.appendChild(newElement);
  };
  </script>
</html>
EOS

front = {"src" => src, "webrick" => svr, "body" => body}

DRb.start_service('druby://localhost:54321', front)

svr.mount_proc '/' do |req, res|
  res.body = front['body']
end

svr.mount_proc('/stream') {|req, res|
  last_event_id = req["Last-Event-ID"]
  res.content_type = 'text/event-stream'
  res.chunked = true
  res.body = WEBrick::ChunkedStream.new(Driq::EventStream.new(src, last_event_id))
}
svr.start
```

Step 0.
Install gems

➡ terminal 1

```
$ gem install webrick
$ gem install driq
```

Step 1.
Run Web Server, and open http://localhost:8086 in browser.

➡ terminal 1

```
$ ruby sse.rb
```

Step 2.
Send events, and check the browser.

➡ terminal 2

```
$ irb -r drb --simple-prompt
>> ro = DRbObject.new_with_uri('druby://localhost:54321?src')
>> ro.write('Hello, Again')
>> ro.write(Time.now)
```

About Me

Masatoshi SEKI, www.druby.org, ninja-testing.com, @m_seki, seki, SW-7603-4893-2503, pokémon GO 3600 2703 1425, https://t.co/oramJa8g0p

The dRuby Book (web edition)

`http://www.druby.org/sidruby/`

dRubyによる分散・Webプログラミング (kindle edition)

`https://www.amazon.co.jp/dp/B079YYPGV3/`