Lab-6

Ibraimov Akzhol

RPC

In this laboratory work I implemented scientific calculator in form of RPC server and client. Which does the following tasks: Client enters the number(s) and choose some mathematical operation to do, they are “factorization”,

“is prime”, “factorial”, “exponentiation”. All calculations are done on server side and return answer to client.

This project was written on Ruby on Rails. For working with message queuing I

used RabbitMQ server. And Bunny gem, which is ruby client for working with

RabbitMQ.

Let's analyza code:

This is our client view, which sends messages to the HomeController#lab5(I updated lab5, so action's name remained “lab5”)

#lab5-1/app/views/home/about.html.erb

<div style="width: 400px; margin: 0 auto;">

<select name="type" style="float: left">

<option value="factorization">Factorization</option>

<option value="is\_prime">Is Prime</option>

<option value="factorial">Factorial</option>

<option value="exponentiation">Exponentiation</option>

</select>

<br><br>

<input id="val1" name="value" placeholder="Enter your value" style="float: left" type="number">

<input id="hid" name="value1" placeholder="Enter second value" style="float: left; visibility: hidden" type="number">

<br><br><br>

<button id="sub">Submit</button>

<br><br>

<label>Answer: </label>

<div id="answer"></div>

</div>

<script>

$(document).ready(function(){

$("select").change(function(){

var val = $(this).val()

if (val == 'exponentiation')

{

$("#hid").css('visibility', 'visible')

}

else

{

$("#hid").css('visibility', 'hidden')

}

});

$("#sub").click(function(){

var type = $("select").val();

var val1 = $("#val1").val();

var val2 = $("#hid").val();

$.ajax({

method: "POST",

url: "/lab5",

dataType: "json",

data: { type: type, value: val1, value1: val2 }

})

.success(function(data){

$("#answer").html(data.status)

});

});

});

</script>

HomeController, which creates channel and calls server, answer comes to response

#lab5-1/app/controllers/home\_controller.rb

def lab5

type = params[:type]

value = params[:value]

value1 = params[:value1]

if type == 'exponentiation'

value = value.to\_s+" "+value1.to\_s

end

require "bunny"

require "thread"

conn = Bunny.new(:automatically\_recover => false)

conn.start

ch = conn.create\_channel

client = CalculatorClient.new(ch, "rpc\_queue")

puts " [x] Requesting "+value+""

response = client.call(type.to\_s+"+"+value.to\_s)

puts " [.] Got #{response}"

ch.close

conn.close

@s = {}

@s[:status] = response

render json: @s

end

Here is class which makes calculations on server

#lab5-2/app/services/calculator\_server.rb

class CalculatorServer

def initialize(ch)

@ch = ch

end

def start(queue\_name)

@q = @ch.queue(queue\_name)

@x = @ch.default\_exchange

@q.subscribe(:block => true) do |delivery\_info, properties, payload|

# n = payload.to\_i

my\_arr = payload.split('+')

type = my\_arr[0]

value = my\_arr[1]

if type == "factorization"

r = self.class.factorization(value)

elsif type == "is\_prime"

r = self.class.is\_prime(value)

elsif type == "factorial"

r = self.class.factorial(value)

elsif type == "exponentiation"

r = self.class.exponentiation(value)

end

# r = self.class.fib(n)

@x.publish(r.to\_s, :routing\_key => properties.reply\_to, :correlation\_id => properties.correlation\_id)

end

end

def self.factorization(value)

require 'prime'

@pd = (value.to\_i).prime\_division

return "factorization of "+value.to\_s+": "+@pd.to\_s

end

def self.is\_prime(value)

require 'prime'

if (value.to\_i).prime?

return value.to\_s+" is prime"

else

return value.to\_s+" is not prime"

end

end

def self.factorial(value)

f = 1; for i in 1..value.to\_i; f \*= i; end; f

return "factorial of "+value.to\_s+": "+f.to\_s

end

def self.exponentiation(value)

my\_arr = value.split(' ')

val1 = my\_arr[0].to\_i

val2 = my\_arr[1].to\_i

return "exponentiation of "+val1.to\_s+" to degree "+val2.to\_s+" is equal to "+(val1\*\*val2).to\_s

end

def self.fib(n)

case n

when 0 then 0

when 1 then 1

else

fib(n - 1) + fib(n - 2)

end

end

end