Ruby Concurrency

Jerry D'Antonio
@jerrydantonio

```ruby
require 'rest_client'
require 'json'

class Finance

  CALLBACK_TOKEN = 'YAHOO.Finance.SymbolSuggest.ssCallback'
  BASE_URL = 'http://d.yimg.com/autoc.finance.yahoo.com/autoc'
  BASE_QS = "query=%s&callback=YAHOO.Finance.SymbolSuggest.ssCallback"

  attr_reader :suggested_symbols

  def initialize(query_string)
    @url = "%s?%s" % [BASE_URL, BASE_QS % query_string.downcase]
    @suggested_symbols = []
  end

  def update
    data = RestClient.get(@url)
    data = data.gsub(/^#{CALLBACK_TOKEN}\(\s*/, '').gsub(/\)\s*$/, '')
    data = JSON.parse(data)
    @suggested_symbols = data['ResultSet']['Result']
    return self
  end
end

yahoo = Finance.new('YAHOO')
yahoo.update.suggested_symbols #=> [{"symbol"=>"YHOO", "name"=>"Yahoo! Inc."...
```


CALIFORNIA
OUTATIME

```ruby
require 'concurrent'

yahoo = Finance.new('YAHOO')
shock = Concurrent::Future.new { yahoo.update.suggested_symbols }
shock.state #=> :pending
shock.value(0) #=> nil (call blocks for 0 seconds)

# do important stuff...

shock.state #=> :fulfilled
shock.value #=> [{"symbol"=>"YHOO", "name"=>"Yahoo! Inc."...
            #   (call blocks indefinitely)

bogus = Finance.new('this creates a bogus URL')
awe = Concurrent::Future.new { bogus.update.suggested_symbols }
awe.state #=> :pending
awe.value(0) #=> nil (call blocks for 0 seconds)

# do important stuff...

awe.state  #=> :rejected
awe.reason #=> #<URI::InvalidURIError: bad URI(is not URI?)...
           #   (call blocks indefinitely)
```

```ruby
require 'concurrent'

class Ticker
  Stock = Struct.new(:symbol, :name, :exchange)

  def update(time, value, reason)
    ticker = value.collect do |symbol|
      Stock.new(symbol['symbol'], symbol['name'], symbol['exch'])
    end

    output = ticker.join("\n")
    print "#{output}\n"
  end
end

yahoo = Finance.new('YAHOO')
future = Concurrent::Future.new { yahoo.update.suggested_symbols }
future.add_observer(Ticker.new)

# do important stuff...

#>> #<struct Ticker::Stock symbol="YHOO", name="Yahoo! Inc.", exchange="NMS">
#>> #<struct Ticker::Stock symbol="YHO.DE", name="Yahoo! Inc.", exchange="GER">
#>> #<struct Ticker::Stock symbol="YAHOY", name="Yahoo Japan Corporation", exchange="PNK">
#>> #<struct Ticker::Stock symbol="YAHOF", name="YAHOO JAPAN CORP", exchange="PNK">
#>> #<struct Ticker::Stock symbol="YOJ.SG", name="YAHOO JAPAN", exchange="STU">
#>> #<struct Ticker::Stock symbol="YHO.SG", name="YAHOO", exchange="STU">
#>> #<struct Ticker::Stock symbol="YHOO.BA", name="Yahoo! Inc.", exchange="BUE">
#>> #<struct Ticker::Stock symbol="YHO.DU", name="YAHOO", exchange="DUS">
#>> #<struct Ticker::Stock symbol="YHO.HM", name="YAHOO", exchange="HAM">
#>> #<struct Ticker::Stock symbol="YHO.BE", name="YAHOO", exchange="BER">
```

```ruby
require 'concurrent'

ticker = Concurrent::Agent.new([])
ticker.value #=> []

yahoo = Finance.new('YAHOO')
ticker.post{|suggested_symbols| suggested_symbols + yahoo.update.suggested_symbols }
ticker.value.length #=> 0

# wait for it...
ticker.value.length #=> 10

ms = Finance.new('Microsoft')
ticker.post{|suggested_symbols| ms.update.suggested_symbols + suggested_symbols }
ticker.value.count #=> 10

# wait for it...
ticker.value.count #=> 20

ticker.post{|suggested_symbols| raise StandardError }
ticker.value.count #=> 20

# wait for it...
ticker.value.count #=> 20
```


Microsoft

```ruby
require 'concurrent'
require 'hamster'

ticker = Concurrent::Agent.new(Hamster.vector)
ticker.value #=> []

yahoo = Finance.new('YAHOO')
ticker.post do |suggested_symbols|
  yahoo.update.suggested_symbols.each do |symbol|
    suggested_symbols = suggested_symbols.cons(symbol)
  end
  suggested_symbols
end
ticker.value.length #=> 0

# wait for it...
ticker.value.length #=> 10

# -- or --

require 'concurrent'
require 'thread_safe'

ticker = Concurrent::Agent.new(ThreadSafe::Array.new)
ticker.value #=> []

yahoo = Finance.new('YAHOO')
ticker.post{|suggested_symbols| suggested_symbols + yahoo.update.suggested_symbols }
ticker.value.length #=> 0

# wait for it...
ticker.value.length #=> 10
```


For all eternit
signed,

```ruby
require 'concurrent'

ticker = Concurrent::Promise.new([]) {|suggested_symbols|
  suggested_symbols + Finance.new('YAHOO').update.suggested_symbols
}.then {|suggested_symbols|
  suggested_symbols + Finance.new('Microsoft').update.suggested_symbols
}
ticker.pending? #=> true

# wait for it...
ticker.pending?     #=> false
ticker.value.length #=> 20

# --------

ticker = Concurrent::Promise.new([]) {|suggested_symbols|
  suggested_symbols + Finance.new('YAHOO').update.suggested_symbols
}.then {|suggested_symbols|
  raise ArgumentError.new("You're a bad monkey Mojo Jojo")
}.rescue(StandardError) {|ex|
  print ex
}
ticker.pending?  #=> true

# wait for it...
ticker.rejected? #=> true
ticker.reason    #=> => #<ArgumentError: You're a bad monkey Mojo Jojo>
```


MISSION TIME
02:19
Guard your HE Cha
00:30
ARKBIOTECH
XP +01524
Mills Kautz
Cowley Manson
35
175

```ruby
require 'concurrent'

task = Concurrent::ScheduledTask.new(2) do
  'What does the fox say?'
end
task.pending?      #=> true
task.schedule_time #=> 2013-11-07 12:20:07 -0500

sleep(3) # wait for it...

task.fulfilled? #=> true
task.value      #=> 'What does the fox say?'

# --------

t = Time.now + 2
task = Concurrent::ScheduledTask.new(t) do
  raise StandardError.new('Call me maybe?')
end
task.pending?      #=> true
task.schedule_time #=> 2013-11-07 12:22:01 -0500

sleep(3) # wait for it...

task.rejected?  #=> true
task.reason     #=> #<StandardError: Call me maybe?>
```

```ruby
require 'concurrent'

observer = Class.new{
  def update(time, value, reason)
    puts "The task completed at #{time} with value:\n\t'#{value}'"
  end
}.new

task = Concurrent::ScheduledTask.new(2) do
  'What does the fox say?'
end
task.add_observer(observer)
task.pending?      #=> true
task.schedule_time #=> 2013-11-07 12:20:07 -0500

sleep(3) # wait for it...

#=> The task completed at 2013-11-07 17:30:41 -0500 with value:
#=>   'What does the fox say?'

# --------

task = Concurrent::ScheduledTask.new(10) do
  raise StandardError.new('Call me maybe?')
end
sleep(1)
task.cancel #=> true
```

```ruby
require 'concurrent'

task = Concurrent::TimerTask.new(execution: 5, timeout: 5) do
  print "Boom!\n"
end

task.execution_interval #=> 5; default 60
task.timeout_interval   #=> 5; default 30
task.run!

# wait 5 seconds...
#=> 'Boom!'

# wait 5 seconds...
#=> 'Boom!'

# wait 5 seconds...
#=> 'Boom!'

task.stop #=> true
```

```ruby
require 'concurrent'

class TaskObserver
  def update(time, result, ex)
    if result
      print "(#{time}) Execution successfully returned #{result}\n"
    elsif ex.is_a?(Concurrent::TimeoutError)
      print "(#{time}) Execution timed out\n"
    else
      print "(#{time}) Execution failed with error #{ex}\n"
    end
  end
end

task = Concurrent::TimerTask.new(execution_interval: 1){ 42 }
task.add_observer(TaskObserver.new)
task.run!

#=> (2013-10-13 19:08:58 -0400) Execution successfully returned 42
#=> (2013-10-13 19:08:59 -0400) Execution successfully returned 42
#=> (2013-10-13 19:09:00 -0400) Execution successfully returned 42
task.stop
```

```ruby
require 'concurrent'

timer_task = Concurrent::TimerTask.new(execution_interval: 1) do |task|

  task.execution_interval.times{ print 'Boom! ' }
  print "\n"
  task.execution_interval += 1

  if task.execution_interval > 5
    puts 'Stopping...'
    task.stop
  end
end

timer_task.run # blocking call - this task will stop itself
#=> Boom!
#=> Boom! Boom!
#=> Boom! Boom! Boom!
#=> Boom! Boom! Boom! Boom!
#=> Boom! Boom! Boom! Boom! Boom!
#=> Stopping...
```

```ruby
require 'concurrent'

class EchoActor < Concurrent::Actor
  def act(*message)
    puts "#{message} handled by #{self}"
  end
end

echo = EchoActor.new
echo.run!

echo.post("Don't panic") #=> true
#=> ["Don't panic"] handled by #<EchoActor:0x007fc8014d0668>

# --------

mailbox, pool = EchoActor.pool(5)
pool.each{|echo| echo.run! }

10.times{|i| mailbox.post(i) }
#=> [0] handled by #<EchoActor:0x007fc8014fb8b8>
#=> [1] handled by #<EchoActor:0x007fc8014fb890>
#=> [2] handled by #<EchoActor:0x007fc8014fb868>
#=> [3] handled by #<EchoActor:0x007fc8014fb890>
#=> [4] handled by #<EchoActor:0x007fc8014fb840>
#=> [5] handled by #<EchoActor:0x007fc8014fb8b8>
#=> [6] handled by #<EchoActor:0x007fc8014fb8b8>
#=> [7] handled by #<EchoActor:0x007fc8014fb818>
#=> [8] handled by #<EchoActor:0x007fc8014fb890>
```

```ruby
require 'concurrent'

class EverythingActor < Concurrent::Actor
  def act(message)
    sleep(5)
    return 42
  end
end

life = EverythingActor.new
life.run!

universe = life.post?('What do you get when you multiply six by nine?')
universe.pending? #=> true

# wait for it...
universe.fulfilled? #=> true
universe.value      #=> 42

life.post!(1, 'Mostly harmless.')

# wait for it...
#=> Concurrent::TimeoutError: Concurrent::TimeoutError
```

```ruby
require 'concurrent'

class ActorObserver
  def update(time, message, result, ex)
    if result
      print "(#{time}) Message #{message} returned #{result}\n"
    elsif ex.is_a?(Concurrent::TimeoutError)
      print "(#{time}) Message #{message} timed out\n"
    else
      print "(#{time}) Message #{message} failed with error #{ex}\n"
    end
  end
end

class SimpleActor < Concurrent::Actor
  def act(*message)
    message
  end
end

actor = SimpleActor.new
actor.add_observer(ActorObserver.new)
actor.run!

actor.post(1)
#=> (2013-11-07 18:35:33 -0500) Message [1] returned [1]

actor.post(1,2,3)
#=> (2013-11-07 18:35:54 -0500) Message [1, 2, 3] returned [1, 2, 3]
```

```ruby
require 'concurrent'

class Ping < Concurrent::Actor

  def initialize(count, pong)
    super()
    @pong = pong
    @remaining = count
  end

  def act(msg)

    if msg == :pong
      print "Ping: pong\n" if @remaining % 1000 == 0
      @pong.post(:ping)

      if @remaining > 0
        @pong << :ping
        @remaining -= 1
      else
        print "Ping :stop\n"
        @pong << :stop
        self.stop
      end
    end
  end
end
```

```ruby
require 'concurrent'

class Pong < Concurrent::Actor

  attr_writer :ping

  def initialize
    super()
    @count = 0
  end

  def act(msg)

    if msg == :ping
      print "Pong: ping\n" if @count % 1000 == 0
      @ping << :pong
      @count += 1

    elsif msg == :stop
      print "Pong :stop\n"
      self.stop
    end
  end
end

pong = Pong.new
ping = Ping.new(10000, pong)
pong.ping = ping

t1 = ping.run!
t2 = pong.run!
sleep(0.1)

ping << :pong
```

```ruby
require 'concurrent'

QUERIES = %w[YAHOO Microsoft google]

class FinanceActor < Concurrent::Actor
  def act(query)
    finance = Finance.new(query)
    print "[#{Time.now}] RECEIVED '#{query}' to #{self} returned #{finance.update.suggested_symbols}\n\n"
  end
end

financial, pool = FinanceActor.pool(5)

timer_proc = proc do
  query = QUERIES[rand(QUERIES.length)]
  financial.post(query)
  print "[#{Time.now}] SENT '#{query}' from #{self} to worker pool\n\n"
end

t1 = Concurrent::TimerTask.new(execution_interval: rand(5)+1, &timer_proc)
t2 = Concurrent::TimerTask.new(execution_interval: rand(5)+1, &timer_proc)

overlord = Concurrent::Supervisor.new

overlord.add_worker(t1)
overlord.add_worker(t2)
pool.each{|actor| overlord.add_worker(actor)}

overlord.run! # the #run method blocks, #run! does not

#>> [2013-10-18 09:35:28 -0400] SENT 'YAHOO' from main to worker pool
#>> [2013-10-18 09:35:28 -0400] RECEIVED 'YAHOO' to #<FinanceActor:0x0000010331af70>...
#>> [2013-10-18 09:35:30 -0400] SENT 'google' from main to worker pool
#>> [2013-10-18 09:35:30 -0400] RECEIVED 'google' to #<FinanceActor:0x0000010331ae58>...

overlord.stop #=> true
```

Write Code!

github.com/jdantonio
@jerrydantonio