



# Ruby Concurrency

Jerry D'Antonio



```
1 require 'rest_client'
2 require 'json'
3
4 class Finance
5
6   CALLBACK_TOKEN = 'YAHOO.Finance.SymbolSuggest.ssCallback'
7   BASE_URL = 'http://d.yimg.com/autoc.finance.yahoo.com/autoc'
8   BASE_QS = "query=%s&callback=YAHOO.Finance.SymbolSuggest.ssCallback"
9
10 attr_reader :suggested_symbols
11
12 def initialize(query_string)
13   @url = "%s?%s" % [BASE_URL, BASE_QS % query_string.downcase]
14   @suggested_symbols = []
15 end
16
17 def update
18   data = RestClient.get(@url)
19   data = data.gsub(/^#\{CALLBACK_TOKEN}\(\s*/, '').gsub(/\)\s*$/, '')
20   data = JSON.parse(data)
21   @suggested_symbols = data['ResultSet']['Result']
22   return self
23 end
24 end
25
26 yahoo = Finance.new('YAHOO')
27 yahoo.update.suggested_symbols #=> [{"symbol"=>"YHOO", "name"=>"Yahoo! Inc."...}
```



```
1 require 'concurrent'
2
3 yahoo = Finance.new('YAHOO')
4 shock = Concurrent::Future.new { yahoo.update.suggested_symbols }
5 shock.state #=> :pending
6 shock.value(0) #=> nil (call blocks for 0 seconds)
7
8 # do important stuff...
9
10 shock.state #=> :fulfilled
11 shock.value #=> [{"symbol"=>"YHOO", "name"=>"Yahoo! Inc."...
12                 #   (call blocks indefinitely)
13
14 bogus = Finance.new('this creates a bogus URL')
15 awe = Concurrent::Future.new { bogus.update.suggested_symbols }
16 awe.state #=> :pending
17 awe.value(0) #=> nil (call blocks for 0 seconds)
18
19 # do important stuff...
20
21 awe.state #=> :rejected
22 awe.reason #=> #<URI::InvalidURIError: bad URI(is not URI?)...
23                 #   (call blocks indefinitely)
```

```
1 require 'concurrent'
2
3 class Ticker
4   Stock = Struct.new(:symbol, :name, :exchange)
5
6   def update(time, value, reason)
7     ticker = value.collect do |symbol|
8       Stock.new(symbol['symbol'], symbol['name'], symbol['exch'])
9     end
10
11   output = ticker.join("\n")
12   print "#{output}\n"
13 end
14 end
15
16 yahoo = Finance.new('YAHOO')
17 future = Concurrent::Future.new { yahoo.update.suggested_symbols }
18 future.add_observer(Ticker.new)
19
20 # do important stuff...
21
22 #>> #<struct Ticker::Stock symbol="YHOO", name="Yahoo! Inc.", exchange="NMS">
23 #>> #<struct Ticker::Stock symbol="YHO.DE", name="Yahoo! Inc.", exchange="GER">
24 #>> #<struct Ticker::Stock symbol="YAH0Y", name="Yahoo Japan Corporation", exchange="PNK">
25 #>> #<struct Ticker::Stock symbol="YAHOF", name="YAHOO JAPAN CORP", exchange="PNK">
26 #>> #<struct Ticker::Stock symbol="YOJ.SG", name="YAHOO JAPAN", exchange="STU">
27 #>> #<struct Ticker::Stock symbol="YHO.SG", name="YAHOO", exchange="STU">
28 #>> #<struct Ticker::Stock symbol="YH00.BA", name="Yahoo! Inc.", exchange="BUE">
29 #>> #<struct Ticker::Stock symbol="YHO.DU", name="YAHOO", exchange="DUS">
30 #>> #<struct Ticker::Stock symbol="YHO.HM", name="YAHOO", exchange="HAM">
31 #>> #<struct Ticker::Stock symbol="YHO.BE", name="YAHOO", exchange="BER">
```



```
1 require 'concurrent'
2
3 ticker = Concurrent::Agent.new([])
4 ticker.value #=> []
5
6 yahoo = Finance.new('YAHOO')
7 ticker.post{|suggested_symbols| suggested_symbols + yahoo.update.suggested_symbols }
8 ticker.value.length #=> 0
9
10 # wait for it...
11 ticker.value.length #=> 10
12
13 ms = Finance.new('Microsoft')
14 ticker.post{|suggested_symbols| ms.update.suggested_symbols + suggested_symbols }
15 ticker.value.count #=> 10
16
17 # wait for it...
18 ticker.value.count #=> 20
19
20 ticker.post{|suggested_symbols| raise StandardError }
21 ticker.value.count #=> 20
22
23 # wait for it...
24 ticker.value.count #=> 20
```





```
1 require 'concurrent'
2 require 'hamster'
3
4 ticker = Concurrent::Agent.new(Hamster.vector)
5 ticker.value #=> []
6
7 yahoo = Finance.new('YAHOO')
8 ticker.post do |suggested_symbols|
9   yahoo.update.suggested_symbols.each do |symbol|
10     suggested_symbols = suggested_symbols.cons(symbol)
11   end
12   suggested_symbols
13 end
14 ticker.value.length #=> 0
15
16 # wait for it...
17 ticker.value.length #=> 10
18
19 # -- or --
20
21 require 'concurrent'
22 require 'thread_safe'
23
24 ticker = Concurrent::Agent.new(ThreadSafe::Array.new)
25 ticker.value #=> []
26
27 yahoo = Finance.new('YAHOO')
28 ticker.post{|suggested_symbols| suggested_symbols + yahoo.update.suggested_symbols }
29 ticker.value.length #=> 0
30
31 # wait for it...
32 ticker.value.length #=> 10
```

A woman with blonde hair tied back in a bun, wearing a yellow dress, is shown from the side, looking down at an open book she is writing in. She holds a quill pen in her right hand and a small ink bottle in her left hand. The book has a yellow cover with some text and a small illustration of a tree on it. The scene is set against a dark purple background.

For all eternity  
signed,  
Alice

unto  
Witch  
e on

```
1 require 'concurrent'
2
3 ticker = Concurrent::Promise.new([]) { |suggested_symbols|
4   suggested_symbols + Finance.new('YAHOO').update.suggested_symbols
5 }.then { |suggested_symbols|
6   suggested_symbols + Finance.new('Microsoft').update.suggested_symbols
7 }
8 ticker.pending? #=> true
9
10 # wait for it...
11 ticker.pending?      #=> false
12 ticker.value.length #=> 20
13
14 # -----
15
16 ticker = Concurrent::Promise.new([]) { |suggested_symbols|
17   suggested_symbols + Finance.new('YAHOO').update.suggested_symbols
18 }.then { |suggested_symbols|
19   raise ArgumentError.new("You're a bad monkey Mojo Jojo")
20 }.rescue(StandardError) { |ex|
21   print ex
22 }
23 ticker.pending? #=> true
24
25 # wait for it...
26 ticker.rejected? #=> true
27 ticker.reason    #=> => #<ArgumentError: You're a bad monkey Mojo Jojo>
```

MISSION TIME

02:19

Guard your HE Cha

00:30



XP +01524

Mills Kautz

Cowley Manson

35  
175



```
1 require 'concurrent'
2
3 task = Concurrent::ScheduledTask.new(2) do
4   'What does the fox say?'
5 end
6 task.pending?      #=> true
7 task.schedule_time #=> 2013-11-07 12:20:07 -0500
8
9 sleep(3) # wait for it...
10
11 task.fulfilled? #=> true
12 task.value       #=> 'What does the fox say?'
13
14 # -----
15
16 t = Time.now + 2
17 task = Concurrent::ScheduledTask.new(t) do
18   raise StandardError.new('Call me maybe?')
19 end
20 task.pending?      #=> true
21 task.schedule_time #=> 2013-11-07 12:22:01 -0500
22
23 sleep(3) # wait for it...
24
25 task.rejected?   #=> true
26 task.reason      #=> #<StandardError: Call me maybe?>
```

```
1 require 'concurrent'
2
3 observer = Class.new{
4   def update(time, value, reason)
5     puts "The task completed at #{time} with value:\n\t#{value}"
6   end
7 }.new
8
9 task = Concurrent::ScheduledTask.new(2) do
10   'What does the fox say?'
11 end
12 task.add_observer(observer)
13 task.pending?      #=> true
14 task.schedule_time #=> 2013-11-07 12:20:07 -0500
15
16 sleep(3) # wait for it...
17
18 #=> The task completed at 2013-11-07 17:30:41 -0500 with value:
19 #=>   'What does the fox say?'
20
21 # -----
22
23 task = Concurrent::ScheduledTask.new(10) do
24   raise StandardError.new('Call me maybe?')
25 end
26 sleep(1)
27 task.cancel #=> true
```



```
1 require 'concurrent'
2
3 task = Concurrent::TimerTask.new(execution: 5, timeout: 5) do
4   print "Boom!\n"
5 end
6
7 task.execution_interval #=> 5; default 60
8 task.timeout_interval   #=> 5; default 30
9 task.run!
10
11 # wait 5 seconds...
12 #=> 'Boom!'
13
14 # wait 5 seconds...
15 #=> 'Boom!'
16
17 # wait 5 seconds...
18 #=> 'Boom!'
19
20 task.stop #=> true
```

```
1 require 'concurrent'
2
3 class TaskObserver
4   def update(time, result, ex)
5     if result
6       print "(#{time}) Execution successfully returned #{result}\n"
7     elsif ex.is_a?(Concurrent::TimeoutError)
8       print "(#{time}) Execution timed out\n"
9     else
10      print "(#{time}) Execution failed with error #{ex}\n"
11    end
12  end
13 end
14
15 task = Concurrent::TimerTask.new(execution_interval: 1){ 42 }
16 task.add_observer(TaskObserver.new)
17 task.run!
18
19 #=> (2013-10-13 19:08:58 -0400) Execution successfully returned 42
20 #=> (2013-10-13 19:08:59 -0400) Execution successfully returned 42
21 #=> (2013-10-13 19:09:00 -0400) Execution successfully returned 42
22 task.stop
```

```
1 require 'concurrent'
2
3 timer_task = Concurrent::TimerTask.new(execution_interval: 1) do |task|
4
5   task.execution_interval.times{ print 'Boom! ' }
6   print "\n"
7   task.execution_interval += 1
8
9   if task.execution_interval > 5
10    puts 'Stopping...'
11    task.stop
12  end
13 end
14
15 timer_task.run # blocking call - this task will stop itself
16 #=> Boom!
17 #=> Boom! Boom!
18 #=> Boom! Boom! Boom!
19 #=> Boom! Boom! Boom! Boom!
20 #=> Boom! Boom! Boom! Boom! Boom!
21 #=> Stopping...
```



```
1 require 'concurrent'
2
3 class EchoActor < Concurrent::Actor
4   def act(*message)
5     puts "#{message} handled by #{self}"
6   end
7 end
8
9 echo = EchoActor.new
10 echo.run!
11 sleep(0.1)
12
13 echo.post("Don't panic") #=> true
14 #=> ["Don't panic"] handled by #<EchoActor:0x007fc8014d0668>
15
16 # -----
17
18 mailbox, pool = EchoActor.pool(5)
19 pool.each{|echo| echo.run! }
20
21 10.times{|i| mailbox.post(i) }
22 #=> [0] handled by #<EchoActor:0x007fc8014fb8b8>
23 #=> [1] handled by #<EchoActor:0x007fc8014fb890>
24 #=> [2] handled by #<EchoActor:0x007fc8014fb868>
25 #=> [3] handled by #<EchoActor:0x007fc8014fb890>
26 #=> [4] handled by #<EchoActor:0x007fc8014fb840>
27 #=> [5] handled by #<EchoActor:0x007fc8014fb8b8>
28 #=> [6] handled by #<EchoActor:0x007fc8014fb8b8>
29 #=> [7] handled by #<EchoActor:0x007fc8014fb818>
30 #=> [8] handled by #<EchoActor:0x007fc8014fb890>
```

```
1 require 'concurrent'
2
3 class EverythingActor < Concurrent::Actor
4   def act(message)
5     sleep(5)
6     return 42
7   end
8 end
9
10 life = EverythingActor.new
11 life.run!
12 sleep(0.1)
13
14 universe = life.post?('What do you get when you multiply six by nine?')
15 universe.pending? #=> true
16
17 # wait for it...
18 universe.fulfilled? #=> true
19 universe.value      #=> 42
20
21 life.post!(1, 'Mostly harmless.')
22
23 # wait for it...
24 #=> Concurrent::TimeoutError: Concurrent::TimeoutError
```

```
1 require 'concurrent'
2
3 class ActorObserver
4   def update(time, message, result, ex)
5     if result
6       print "(#{time}) Message #{message} returned #{result}\n"
7     elsif ex.is_a?(Concurrent::TimeoutError)
8       print "(#{time}) Message #{message} timed out\n"
9     else
10      print "(#{time}) Message #{message} failed with error #{ex}\n"
11    end
12  end
13 end
14
15 class SimpleActor < Concurrent::Actor
16   def act(*message)
17     message
18   end
19 end
20
21 actor = SimpleActor.new
22 actor.add_observer(ActorObserver.new)
23 actor.run!
24
25 actor.post(1)
26 #=> (2013-11-07 18:35:33 -0500) Message [1] returned [1]
27
28 actor.post(1,2,3)
29 #=> (2013-11-07 18:35:54 -0500) Message [1, 2, 3] returned [1, 2, 3]
```

```
1 require 'concurrent'
2
3 class Ping < Concurrent::Actor
4
5   def initialize(count, pong)
6     super()
7     @pong = pong
8     @remaining = count
9   end
10
11  def act(msg)
12
13    if msg == :pong
14      print "Ping: pong\n" if @remaining % 1000 == 0
15      @pong.post(:ping)
16
17    if @remaining > 0
18      @pong << :ping
19      @remaining -= 1
20    else
21      print "Ping :stop\n"
22      @pong << :stop
23      self.stop
24    end
25  end
26 end
27 end

1 require 'concurrent'
2
3 class Pong < Concurrent::Actor
4
5   attr_writer :ping
6
7   def initialize
8     super()
9     @count = 0
10  end
11
12  def act(msg)
13
14    if msg == :ping
15      print "Pong: ping\n" if @count % 1000 == 0
16      @ping << :pong
17      @count += 1
18
19    elsif msg == :stop
20      print "Pong :stop\n"
21      self.stop
22    end
23  end
24 end
25
26 pong = Pong.new
27 ping = Ping.new(10000, pong)
28 pong.ping = ping
29
30 t1 = ping.run!
31 t2 = pong.run!
32 sleep(0.1)
33
34 ping << :pong
```



```
1 require 'concurrent'
2
3 QUERIES = %w[YAHOO Microsoft google]
4
5 class FinanceActor < Concurrent::Actor
6   def act(query)
7     finance = Finance.new(query)
8     print "[#{Time.now}] RECEIVED '#{query}' to #{self} returned #{finance.update.suggested_symbols}\n\n"
9   end
10 end
11
12 financial, pool = FinanceActor.pool(5)
13
14 timer_proc = proc do
15   query = QUERIES[rand(QUERIES.length)]
16   financial.post(query)
17   print "[#{Time.now}] SENT '#{query}' from #{self} to worker pool\n\n"
18 end
19
20 t1 = Concurrent::TimerTask.new(execution_interval: rand(5)+1, &timer_proc)
21 t2 = Concurrent::TimerTask.new(execution_interval: rand(5)+1, &timer_proc)
22
23 overlord = Concurrent::Supervisor.new
24
25 overlord.add_worker(t1)
26 overlord.add_worker(t2)
27 pool.each{|actor| overlord.add_worker(actor)}
28
29 overlord.run! # the #run method blocks, #run! does not
30
31 #>> [2013-10-18 09:35:28 -0400] SENT 'YAHOO' from main to worker pool
32 #>> [2013-10-18 09:35:28 -0400] RECEIVED 'YAHOO' to #<FinanceActor:0x0000010331af70>...
33 #>> [2013-10-18 09:35:30 -0400] SENT 'google' from main to worker pool
34 #>> [2013-10-18 09:35:30 -0400] RECEIVED 'google' to #<FinanceActor:0x0000010331ae58>...
35
36 overlord.stop #=> true
```



```
1 require 'eventmachine'
2
3 class Echo < EventMachine::Connection
4   def post_init
5     send_data "CONNECTED >>\n"
6   end
7
8   def receive_data(data)
9     data = data.strip
10    if data == 'stop'
11      puts 'Stopping...'
12      EventMachine.stop
13    else
14      send_data ">> #{data}\n"
15      puts "ECHO: #{data}"
16    end
17  end
18
19
20 EventMachine.run {
21   EventMachine.start_server('127.0.0.1', 8081, Echo)
22 }
23
24 #=> ECHO: Say my name.
25 #=> ECHO: Heisenberg
26 #=> ECHO: I am not in danger, Skylar.
27 #=> ECHO: I am the danger.
28 #=> ECHO: No. I am the one who knocks!
29 #=> Stopping...
30 #=> nil
```

```
[22:22:42 Jerry ~]$ telnet 127.0.0.1 8081
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
CONNECTED >>
Say my name.
>> Say my name.
Heisenberg
>> Heisenberg
I am not in danger, Skylar.
>> I am not in danger, Skylar.
I am the danger.
>> I am the danger.
No. I am the one who knocks!
>> No. I am the one who knocks!
stop
Connection closed by foreign host.
```

```
1 require 'eventmachine'
2 require 'amqp'
3 require 'arachni/rpc/em'
4
5 QUEUE_NAME = 'fun.with.eventmachine'
6
7 EventMachine.run do
8   connection = AMQP.connect(:host => '127.0.0.1')
9   channel   = AMQP::Channel.new(connection)
10  queue     = channel.queue(QUEUE_NAME, :auto_delete => true)
11  exchange  = channel.direct('')
12
13  queue.subscribe do |payload|
14    exchange.publish(payload, :routing_key => queue.name)
15  end
16
17  connection.on_error do |conn, connection_close|
18    puts 'AMQP Connection Error!'
19  end
20
21 EventMachine.start_server('127.0.0.1', 8081, Echo)
22
23 EM.add_periodic_timer(10) do
24   puts "[#{Time.now}] The cake is a lie."
25 end
26
27 Signal.trap('TERM'){ EventMachine.stop }
28 Signal.trap('INT'){ EventMachine.stop }
29
30 rpc_server = Arachni::RPC::EM::Server.new(host: 'localhost', port: 7332)
31 rpc_server.add_handler('vector', Array.new)
32 rpc_server.run
33 end
```



```
1 require 'eventmachine'
2 require 'em-http-request'
3 require 'json'
4
5 module EmFinance
6
7   CALLBACK_TOKEN = 'YAHOO.Finance.SymbolSuggest.ssCallback'
8   BASE_URL = 'http://d.yimg.com/autoc.finance.yahoo.com/autoc'
9   BASE_QS = "query=%s&callback=YAHOO.Finance.SymbolSuggest.ssCallback"
10
11 def self.suggested_symbols(query, peer)
12   qs = {query: query, callback: 'YAHOO.Finance.SymbolSuggest.ssCallback'}
13   http = EventMachine::HttpRequest.new(BASE_URL).get(query: qs)
14
15   http.errback { peer.send_data("An error occurred retrieving data for '#{query}'") }
16
17   http.callback {
18     data = http.response.gsub(/^\#{CALLBACK_TOKEN}\(\s*/, '').gsub(/\)\s*$/, '')
19     data = JSON.parse(data)
20     peer.send_data("#{data['ResultSet']['Result']}\n")
21   }
22 end
23 end
24
25 class AsyncFinance < EventMachine::Connection
26   def receive_data(data)
27     data = data.strip
28
29     if data =~ /^goodbye$/i
30       close_connection
31     else
32       EmFinance.suggested_symbols(data, self)
33     end
34   end
35 end
36
37 EventMachine.run do
38   Signal.trap('TERM'){ EventMachine.stop }
39   Signal.trap('INT'){ EventMachine.stop }
40
41   EventMachine.start_server('127.0.0.1', 8081, AsyncFinance)
42 end
```

```
1 require 'eventmachine'
2 require 'socket'
3
4 class AsyncFinance < EventMachine::Connection
5
6   def post_init
7     @peername = "%s:%s" % Socket.unpack_sockaddr_in(get_peername).reverse
8     puts "Connected to #{@peername}"
9   end
10
11  def unbind
12    puts "Disconnected from #{@peername}"
13  end
14
15  def receive_data(data)
16    data = data.strip
17
18    if data =~ /^goodbye$/i
19      close_connection
20    else
21      EventMachine.defer do
22        puts "Getting stock symbol information for '#{data}'"
23        suggested_symbols = Finance.new(data).update.suggested_symbols
24        send_data("#{suggested_symbols.to_s}\n")
25      end
26    end
27  end
28 end
29
30 EventMachine.run do
31   Signal.trap('TERM'){ EventMachine.stop }
32   Signal.trap('INT'){ EventMachine.stop }
33
34   EventMachine.start_server('127.0.0.1', 8081, AsyncFinance)
35 end
```



```
1 require 'celluloid'
2 require 'rest_client'
3 require 'json'
4
5 class FinanceActor
6   include Celluloid
7
8   CALLBACK_TOKEN = 'YAHOO.Finance.SymbolSuggest.ssCallback'
9   BASE_URL = 'http://d.yimg.com/autoc.finance.yahoo.com/autoc'
10  BASE_QS = "query=%s&callback=YAHOO.Finance.SymbolSuggest.ssCallback"
11
12 attr_reader :suggested_symbols
13 attr_reader :url
14
15 def initialize(query_string)
16   @url = "%s?%s" % [BASE_URL, BASE_QS % query_string.downcase]
17   @suggested_symbols = []
18 end
19
20 def update
21   data = RestClient.get(@url)
22   data = data.gsub(/^#{CALLBACK_TOKEN}\(\s*/, '').gsub(/\)\s*$/, '')
23   data = JSON.parse(data)
24   @suggested_symbols = data['ResultSet']['Result']
25   return self
26 end
27 end
28
29 yahoo = FinanceActor.new('YAHOO')
30 yahoo.update.suggested_symbols #=> [{"symbol"=>"YHOO", "name"=>"Yahoo! Inc."...}
31
32 microsoft = FinanceActor.new('Microsoft')
33 microsoft.async.update
34 microsoft.suggested_symbols #=> []
35
36 # do important stuff...
37 microsoft.suggested_symbols #=> [{"symbol"=>"MSFT", "name"=>"Microsoft Corporation"...}
```

```
1 require 'celluloid'
2
3 class Financier
4   include Celluloid
5
6   trap_exit :actor_died
7
8   def actor_died(actor, reason)
9     print "#{actor.inspect} failed with exception #{reason.message}\n"
10 end
11
12 def update(*queries)
13   @minions = queries.collect do |query|
14     minion = FinanceActor.new(query)
15     self.link(minion)
16     minion.async.update
17     minion
18   end
19 end
20
21 def suggested_symbols
22   @minions.reduce([]) do |collection, minion|
23     collection + minion.suggested_symbols
24   end
25 end
26 end
27
28 financier = Financier.new
29 financier.async.update('YAHOO', 'Microsoft', 'google', 'die! die! die!') #=> nil
30
31 # do important stuff...
32 #>> E, [2013-10-17T07:04:11.559228 #16865] ERROR -- : FinanceActor crashed!
33 #>> ... stack trace ...
34 #>> #<Celluloid::ActorProxy(FinanceActor) dead> failed with exception bad URI...
```

```
1 require 'celluloid'
2 require 'rest_client'
3 require 'json'
4
5 class FinanceMinion
6   include Celluloid
7
8   CALLBACK_TOKEN = 'YAHOO.Finance.SymbolSuggest.ssCallback'
9   BASE_URL = 'http://d.yimg.com/autoc.finance.yahoo.com/autoc'
10  BASE_QS = "query=%s&callback=YAHOO.Finance.SymbolSuggest.ssCallback"
11
12  attr_reader :suggested_symbols
13
14  def update(query_string)
15    @url = "%s?%s" % [BASE_URL, BASE_QS % query_string.downcase]
16    data = RestClient.get(@url)
17    data = data.gsub(/^\#\{CALLBACK_TOKEN}\(\s*/, '').gsub(/\)\s*$/, '')
18    data = JSON.parse(data)
19    return data['ResultSet']['Result']
20  end
21 end
22
23 class FinancierGroup < Celluloid::SupervisionGroup
24   pool FinanceMinion, as: :finance, size: 3
25 end
26
27 FinancierGroup.run!
28
29 ['YAHOO', 'Microsoft', 'google'].each do |query|
30   puts Celluloid::Actor[:finance].update(query)
31 end
32
33 # do important stuff...
34
35 #>> {"symbol"=>"YHOO", "name"=>"Yahoo! Inc."...
36 #>> {"symbol"=>"YOJ.SG", "name"=>"YAHOO JAPAN"...
37 #>> {"symbol"=>"YHOO.BA", "name"=>"Yahoo! Inc."...
38 #>> and many more...
```

# Write Code!

[github.com/jdantonio](https://github.com/jdantonio)  
@jerrydantonio