EventMachine

Evented programming i Ruby

Mig!

- Rasmus Rønn Nielsen
- Medejer og udvikler i Playonic/Odia
  - virtualmanager.com - fodboldmanager
  - fargonia.com - real time strategy
- twitter.com/rasmusrn

Min vej til EventMachine

- Fargonia - et RTS
  - 5-15 villagers per bruger
  - bruger sender villager-ordrer (byg hus, gå til)
  - eksekveringen af disse ordrer varer mellem 0 og 60 sekunder
- Krav: Selvstændig daemon som kan behandle mange simultane "ordrer".
  - Hvis 2000 brugere online som giver 10 ordrer på samme tid: 20.000 simultane ordrer
- Hvordan implementerer vi dette?

Threading

- Det "traditionelle" valg
- ActiveRecord og threading virker ikke out of the box
- Ruby og threading - performance-problemer
- Ruby 1.9 - bedre, men stadig ikke "true concurrency" (Global interpreter lock)

Threading overhead

- Thread-skedulering - hvilken tråd skal processes "nu"?
- Blocking IO (mysql, memcached, http osv.)
  - Fortsat skedulering - unødvendigt overhead
- Server med tusindvis af klienter
  - tusindvis af tråde = enormt overhead

Evented programming (threading alternativ)

- Høj concurrency i kun én tråd (slipper for skedulering) via asynkron processering

- Ét "tæt" uendeligt loop: "event loop"

```
# pseudo-eksempel
server = start_server
loop do
    buffer << server.read # blokker ikke
    if buffer.complete?
       do_something buffer
    end
end
```

- Coupling: Netværks-logik og business-logik blandet sammen... hvad dælan gør vi? :-(

EventMachine!

- Event-modellen er ikke indbygget i Ruby. Kan alligevel lade sig gøre takket være blocks.
- Reactor-pattern: Separation af netværks-logik og business-logik
- EventMachine er en implementation af reactor-pattern'et der gør evented programming mulig i Ruby.
- Al kode der eksekveres i EventMachine's reactor skal være asynkron.

Asynkron kode

- Typisk ruby-kode er synkron:

```
result = calculate()
use_result(result)
```

- Asynkron/evented kode bruger blocks:

```
calculate do |result|
    use_result result
end
```

- AJAX

```
// jQuery
$.get('/test', function() { alert(2); });
alert(1);
```

Asynkron kode, ulemper

- Lav læsbarhed i komplekse tilfælde
- Kan ikke bruge exceptions - besværliggør error handling
- Besværlig at teste
- Ruby 1.9's Fibers to the rescue?

Blocking IO er forbudt i EM!

- Man kan ikke bruge "normale" libraries i et event loop
- Al blocking IO er "forbudt" - ActiveRecord, ruby-mysql
- EventMachine.defer { block_io() }
- Non blocking ORM: Cramp::Model (github.com/lifo/cramp)

```
User.first do |user|
    user.name = 'Rasmus'
    user.save do |result|
        puts "Jeg blev gemt!" if result.success?
    end
end
```

HTTP request-eksempel

```
require 'eventmachine'
require 'em-http'

  EM.run {
    http = EventMachine::HttpRequest.new('http://127.0.0.1/').get

    http.callback do
      p http.response_header.status
      p http.response_header
      p http.response
    end
  }
```

Simpel server-eksempel

```
require 'eventmachine'

module EchoServer
  def post_init
    puts "Nogen connected!"
  end

  def receive_data data
    puts "Jeg modtog #{data}"
    send_data "Right back at ya: #{data}"
  end

  def unbind
    puts "Nogen disconnectede!"
  end
end

EM.run {
  EventMachine::start_server "127.0.0.1", 1234, EchoServer
}
```

Kompleks kode, eksempel

```
def capable?(&block)
  map_object do |map_object|
    if map_object.is_a? Villager
      components do |components|
        map_object.has_components? components do |success|
          if success
            tools do |tools|
              map_object.has_tools? tools, block
            end
          else
            block.call false
          end
        end
      end
    else
      block.call false
    end
  end
end
```

```
def capable?
map_object.is_a?(Villager) &&
map_object.has_components?(components) &&
map_object.has_tools(tools)
end
```

EventMachine: Pros and cons

- Positivt
  - Høj performance
  - Skalerbart
  - Behøver ikke tænke på thead-safety
- Negativt
  - Anderledes programmingstil
  - Få libraries virker asynkront
  - Kan ikke bruges sammen med ActiveRecord
- I Fargonia's tilfælde opvejer fordelene ulemperne (tror jeg!)

Real world eksempel

Fargonia-demo!

Spørgsmål?

virtualmanager.com (vi søger udviklere!)
fargonia.com
twitter.com/rasmusrn