EventMachine

fast, simple event-processing library for Ruby programs

EdgeCase

Performance

EdgeCase

Scalability

EdgeCase

Ruby Web Servers

One Process per Request

EdgeCase

Concurrency

EdgeCase

Processes

- Add VMs or Real Machines to handle load
- All Rubies are running deep clones
- Application code is responsible for managing scale

EdgeCase

Threads

- Reuse system resources (VMs, images)
- Doesn't translate well to Windows
- really need MRI 1.9+ to take full advantage of OS.
- Application code is responsible for managing scale

EdgeCase

Evented I/O

- Reactor Pattern
- Pub/Sub to events
- Focus on behavior of the server not scale
- Application doesn't need to worry about scale, internally

EdgeCase

Reactor Pattern

- Resources (Supplies I/O)
- Synchronous Event Demuliplexer (Loop)
- Dispatcher (Handles Pub/Sub)
- Request Handler
- Single Threaded by design

EdgeCase

Reactor.kind_of?
EventLoop #=> true

- Single Threaded
- while loop
- Application code "reacts" to events
- handlers can "block" the Event Loop

EdgeCase

NEVER BLOCK THE REACTOR

EdgeCase

Asynchronous
vs.
Synchronous

EdgeCase

Synchronous Ruby

- code that uses return values
- even blocks/procs should be stored and executed out of process

```
result = my_method
my_other_method(result)
```

EdgeCase

Asynchronous Ruby

- Heavy use of blocks
- Callbacks
- Execute blocks as late as possible to prevent blocking the loop

```
my_method do |result|
  my_other_method(result)
end
```

EdgeCase

EM.run

```
EM.run{
  puts “Reactor is reacting!”
}

# EventMachine is in change now.
# RUN IS BLOCKING!
```

You can always check: EM.reactor_running?
You can always stop it: EM.stop

EdgeCase

Iterating the EM way

```
count = 0
job = proc{
  if count < 1000
    perform_operation_at_index(count)
    count += 1
  end
  EM.next_tick(&job)
}
EM.next_tick(&job)
```

- EM.next_tick
- split work on multiple passes of the loop

EdgeCase

Iterating the EM way (WIN)

```
count = 0
job = EM.tick_loop do
  if count < 1000
    perform_operation_at_index(count)
    count += 1
  else
    :stop
  end
end
job.on_stop{ puts “finished being awesome!” }
```

- EM.tick_loop
- Handles recursion, proc scheduling
- fires events!

EdgeCase

EM::Iterator

- most control (concurrency)
- map/inject
- auto scheduling of job in loop

```
EM::Iterator.new(0..1000).each{ |n,iterator| iterator.next }
```

```
EM::Iterator.new(0..1000, 2).each do |n,iterator|
  # pause without blocking and in (2) chunks per tick!
  EM.add_timer(1){ iterator.next }
end
```

EdgeCase

EM in the REAL

- Thin
- Rainbows
- async_sinatra
- YOUR CUSTOM SERVER!

EdgeCase

DEMO - Echo Server

EdgeCase

```
require 'eventmachine'

module EchoServer
  def post_init
    puts "-- someone connected to the echo server!"
  end

  def receive_data data
    send_data ">>>you sent: #{data}"
    close_connection if data =~ /quit/i
  end

  def unbind
    puts "-- someone disconnected from the echo server!"
  end
end

# Note that this will block current thread.
EventMachine.run {
  EventMachine.start_server "127.0.0.1", 8081, EchoServer
}
```

EdgeCase

Installation

- gem install eventmachine
- require ‘eventmachine’

EdgeCase

Supported Rubies

- C++ Reactors:
  - MRI 1.8
  - MRI 1.9
  - Rubinius
- JRuby Reactors:
  - JRuby
- Pure Ruby version exists as well.

EdgeCase

EM is extensible!

- em-http-request
- em-websocket
- em-zeromq
- em-mysql
- em-mongo

EdgeCase

Battle Tested

- EngineYard
- Heroku
- Github
- 37Signals (campfire)
- RightScale
- Smule

EdgeCase

Resources

- EventMachine Code: https://github.com/eventmachine/eventmachine/
- Aman Gupta's awesome EM presentation http://timetobleed.com/eventmachine-scalable-non-blocking-io-in-ruby/

EdgeCase