

Coisas que aprendi e quero passar adiante

Lucas Húngaro
@lucashungaro

GoNow Tecnologia

Lucas

Culinária

Apple

Ruby

Café

Tecnologia

Home Cinema

Cinema

Drinks

Futebol

Software

Games

Música

Duas partes

Serviços e bibliotecas

O que usei e recomendo

Gems e Plugins

bullet
query_reviewer
rails_indexes } db performance
& optimization

kasket } caching
cachy }

oink } profiling

Oink

```
Aug 12 11:31:15 Iron-Man rails[9541]: Memory usage: 101938 | PID: 9541  
Aug 12 11:31:15 Iron-Man rails[9541]: Instantiation Breakdown: Total:  
39 | User: 20 | FeedItem: 10 | Tag: 9
```

Cachy

versionamento

evita o “dog pile effect”

caches interdependentes

várias outras features muito interessantes

<http://github.com/grosser/cachy>

Master/Slave

Para ActiveRecord, DBCharmer

Filas: Resque

Simple, fácil e eficiente

Resque

Overview

Working

Failed

Queues

Workers

Stats

Queues

The list below contains all the registered queues with the number of jobs currently in the queue. Select a queue from above to view all jobs currently pending on the queue.

Name	Jobs
activities	0
activity_feeds	5
failed	0

1 of 1 Workers Working

The list below contains all workers which are currently running a job.

	Where	Queue	Processing
	Iron-Man.local:4314	ACTIVITY_FEEDS	ActivityFeedProcessor 3 days ago

Powered by [Resque](#) v1.10.0

Connected to Redis namespace resque on redis://localhost:6379/0

Resque

monitoramento

plugins

cuidado: json e símbolos

cuidado: tarefas antes seriais passam a ser paralelas

cuidado: locale dos workers (sobem outro contexto)

O Resque adiciona um componente à sua infra-estrutura, mas é feito para ser facilmente monitorado, estendido e distribuído. Muitas vantagens a um preço muito baixo (e “improvisar” filas no banco de dados da sua app pode custar bem mais caro)

Arquitetura

Web App == Database View

Utilize filas, caching, server push (Comet, Sockets etc)

Processamento síncrono mínimo

Cloud Computing

Muito cuidado com o hype

Hype não é algo ruim por si só, pois leva a mais inovação mas, sem o devido cuidado, podemos “errar por empolgação”

Amazon

Noisy neighbors

Latência e I/O

Superpopulação

Caso a caso

Evite o hype

Benchmark

Amazon

Staging/QA/Homologação

Processamento paralelo/distribuído

CDN

Aumento instantâneo de throughput



The Good

The Bad



Banco de Dados

I/O intensiva

Amazon

<http://goo.gl/0fgw>

Código eficiente

E como padrões ineficientes podem te atrapalhar

Maturidade

O conhecimento, tanto individual, quanto o do grupo, evolui de forma lenta

Caching

Excesso de observers/callbacks

Micro-gerenciamento

Ineficiente

Bug prone

Smart Keys FTW!

aka Versionamento

```
class User < ActiveRecord::Base
  has_many :search_terms, :dependent => :destroy,
  :order => "term ASC"

  def save_search_term(term)
    self.update_attribute(:last_saved_search_at,
Time.now.utc)
    search_terms.create(:term => term)
  end

  def remove_search_term(term_id)
    self.update_attribute(:last_saved_search_at,
Time.now.utc)
    self.search_terms.destroy(term_id)
  end
end
```

```
def saved_search_terms
  Rails.cache.fetch(cache_key_for_saved_searches,
:expires_in => 30.days) do
    self.search_terms
  end
end

private

def cache_key_for_saved_searches
  timestamp = self[:last_saved_search_at].to_s.gsub("
", "_").gsub(":", "-")
  "User/#{self[:id]}/search_terms-#{timestamp}"
end
end
```

cache_proxy

http://github.com/lucashungaro/cache_proxy

Programação defensiva

Não assumas coisas

Principalmente sobre entrada

Postel's Law

(robustness principle)

*“Be conservative in what you send;
be liberal in what you accept.”*

Programação defensiva

Gather input

Perform work

Deliver results

Handle errors

Law of Demeter

“Each unit should only talk to its friends; don't talk to strangers.”

```
class Company < ActiveRecord::Base
  has_many :addresses

  def formatted_city
    self.addresses.first.city.name
  end
end
```

```
def formatted_city
  main_address.city.name
end

private
def main_address
  self.addresses.first
end
```

SOLID

Jim Weirich
Amanhã, 17h

Single Responsibility Principle

Uma e apenas uma razão para mudar

Fat Models

Negócios + Persistência

Esconde o domínio

Falta: flexibilidade, isolamento



Camadas



Tudo junto e misturado

```

class ActivityService
  def initialize(actor)
    @actor = actor
    @activities = []
  end

  def register(type, action, options = {})
    raise ArgumentError("Every action needs a valid
object") unless options[:object]

    (...) # complex treatment of the activity data
  end

  def process(queue_manager = Resque)
    queue_manager.enqueue(ActivityProcessor, @activities,
I18n.locale)
  end

  (...)
end

```

SOLID:

SRP: a razão para mudar se torna única – apenas a formatação das opções. Quem cuida da busca é o wrapper.

OCP: fechada para modificação da implementação, mas aberta para extensão via modificação do wrapper e injeção da dependência

DIP: duck typing + injeção de dependência – a dependência é gerenciada pelo cliente, liberando o código e tornando-o mais flexível

```
class SearchService
  def initialize(classes = [])
    @classes = classes
  end

  def add_class(klass)
    @classes << klass
  end

  def execute(query, options = {}, wrapper = ThinkingSphinx)
    (...) # options processing (defaults and so on)

    wrapper.search(query, options_for_engine)
  end

  (...)
end
```

SOLID:

SRP: a razão para mudar se torna única – apenas a formatação das opções. Quem cuida da busca é o wrapper.

OCP: fechada para modificação da implementação, mas aberta para extensão via modificação do wrapper e injeção da dependência

DIP: duck typing + injeção de dependência – a dependência é gerenciada pelo cliente, liberando o código e tornando-o mais flexível

```
let(:service) { SearchService.new([User]) }

it "should pass default options to the search wrapper" do
  default_options = {
    :page => 1,
    :per_page => 20,
    :match_mode => :extended
  }

  ThinkingSphinx.expects(:search)... # suppressed
  service.execute("teste")
end
```

Sem utilizar a injeção de dependência, acabamos manipulando o comportamento da classe diretamente, revelando a falta de flexibilidade do código.

```
let(:service) { SearchService.new([User]) }

it "should pass default options to the search wrapper" do
  default_options = {
    :page => 1,
    :per_page => 20,
    :match_mode => :extended
  }

  wrapper = mock('search_wrapper')
  wrapper.expects(:search)... # suppressed
  service.execute("teste", {}, wrapper)
end
```

Com a injeção da dependência, podemos utilizar objetos fake para especificar o comportamento esperado sem setup adicional, graças ao duck typing.
Também conseguimos o efeito descrito em “Mock roles, not object states” (<http://www.infoq.com/news/2008/08/Mock-Roles-Pryce-and-Freeman>)

Conclusões

Guidelines, not laws

Mantra

Se é difícil de testar em
isolamento, está mal projetado*

* 99,99% de chance ;)

Checklists

Uma boa forma de adquirir bons hábitos

Lógica importante deve:

ser explícita

ter um nome

ter uma localização clara

Ao finalizar spec e objeto:

aplica DRY?

tem apenas uma responsabilidade?

depende de componentes que mudam
menos que ele?

Zen of Python

```
>>> import this
```

Obrigado

Referências

<http://gist.github.com/524462>