[IF977] Engenharia de Software

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

AssertLab
Advanced Software and Systems Engineering Research Technologies
Centro de Informática
Universidade Federal de Pernambuco

Licença do material

Este Trabalho foi licenciado com uma Licença

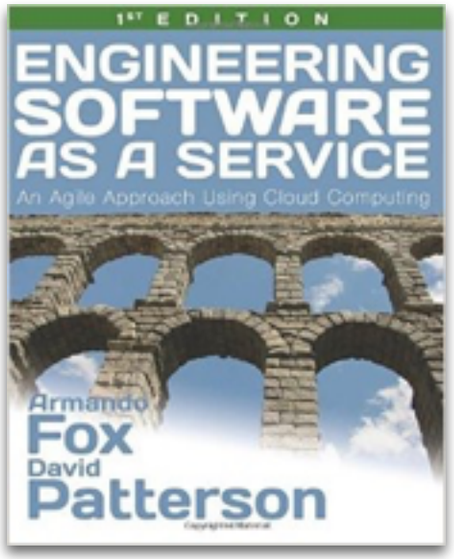
Creative Commons - Atribuição-NãoComercial-Compartilhalgual 3.0 Não Adaptada.

Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
  - http://bit.ly/TDOA5L
- SWEBOK
  - Guide to the Software Engineering Body of Knowledge (SWEBOK): http://www.computer.org/web/swebok
- Engineering Software as a Service: An Agile Approach Using Cloud Computing (Beta Edition)
  - http://www.saasbook.info/



Outline

- (ESaaS 3.5) All Programming is Metaprogramming
- (ESaaS 3.6) Blocks, Iterators, Functional Idioms
- (ESaaS 3.7) Mixins and Duck Typing
- (ESaaS 3.8) Yield

Toda programação é Metaprogramação

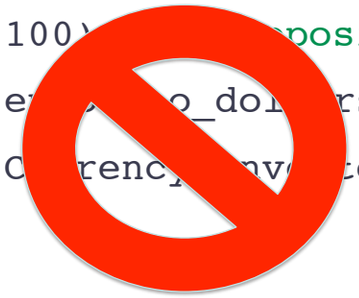
(Engineering Software as a Service §3.5)

Metaprogramação & Reflexão

- Reflexão nos permite perguntar a um objeto coisas sobre ele mesmo e que ele se modifique
- Metaprogramação nos permite definir um “novo código” em tempo de execução
- Como podemos deixar nosso código mais “DRY”ficado, mais conciso ou legível?
  - Ou são apenas palavras para me fazer parecer mais inteligente

Um conta de banco internacional

```
acct.deposit(100)          # deposit $100
acct.deposit(euro_dollars(20)) # about $25
acct.deposit(CurrencyConverter.new(:euros, 20))
```



Think for 1 minute why second one is not preferred

Um conta de banco internacional

```
acct.deposit(100)     # deposit $100
acct.deposit(20.euros) # about $25
```

Sem problemas com classes abertas...

```
class Numeric
  def euros; self * 1.292; end
end
```

http://pastebin.com/f6WuV2rC

Mas e sobre...

```
acct.deposit(1.euro)
```

http://pastebin.com/WZGBhXci

O Poder do method_missing

- Mas e se supormos que queremos dar suporte a
  - `acct.deposit(1000.yen)` http://pastebin.com/agjb5qBF
  - `acct.deposit(3000.rupees)`
- Há alguma maneira DRY de fazer isso?

http://pastebin.com/HJTvUid5

Add the currency 'peso' with exchange rate 1 peso = 0.076 dollars

O Poder do method_missing

```ruby
class Numeric
  @@currencies = {'yen' => 0.013, 'euro' => 1.292, 'rupee' => 0.019}
  def method_missing(method_id, *args, &block)  # capture all args in case have to call super
    singular_currency = method_id.to_s.gsub( /s$/, '')
    if @@currencies.has_key?(singular_currency)
      self * @@currencies[singular_currency]
    else
      super
    end
  end
end
```

```ruby
def self.add_currency(name,conversion)
  @@currencies[name.to_s] = conversion
end
```

Outro exemplo prático

```
class Teste

  def method_missing(method, *args)

    print "Método #{method} chamado na classe Teste,
com os argumentos #{args.join(',')}\n"

  end

end


t = Teste.new()

t.imprimir

t.qualquer_coisa 1, 2, 3, "asd", :teste => 1
```


qual é o resultado da execução deste programa?

Problemas

- Não é uma boa prática usar o method_missing o tempo todo
  - Tudo funciona sem problemas, o que pode criar um objeto com comportamento inconsistente
- Melhor abordagem é utilizar o define_method, conforme o exemplo a seguir

Outro exemplo prático 2

```
module Propriedades
  def propriedade(nome)
    ivarname = "@#{nome}".to_sym
    self.send :define_method, nome do
      sel.instance_variable_get ivarname
    end
    self.send :define_method, "#{nome}=".to_sym do |val|
      self.instance_variable_set ivarname, val
    end
  end
end

class Teste
  extend Propriedades
  propriedade :nome
end

t = Teste.new()
t.nome = "vinicius"
puts t.inspect
```

Reflexão e Metaprogramação

- Você pode “perguntar” aos objetos Ruby questões sobre eles mesmo em tempo de execução
- Você pode usar esta informação para gerar novos códigos (métodos, objetos, classes) em tempo de execução
- Você pode “reabrir” qualquer classe em qualquer momento e adicionar coisas a elas

Examples:
- ask a class what its superclass is
- ask an object what its class is
- ask if an object is a kind_of?(somethign)
- ask what methods an object responds to
- ask whether it responds to some method

Use class_eval to create a new instance method
Lead up to attr_accessor. Homework: impelment attr_accessor,reader,writer

Pergunta

Suponha que a gente queira manipular
`5.euros.in(:rupees)`
Que mudança seria mais apropriada ser feita em Numeric?

A. Mudar Numeric.method_missing para detectar chamadas ao 'in' com os argumentos apropriados

B. Mudar Numeric#method_missing para detectar chamadas ao 'in' com os argumentos apropriados

C. Definir o método Numeric#in

D. Definir o método Numeric.in

Pergunta

Suponha que a gente queira manipular
`5.euros.in(:rupees)`
Que mudança seria mais apropriada ser feita em Numeric?

A. Mudar Numeric.method_missing para detectar chamadas ao 'in' com os argumentos apropriados

B. Mudar Numeric#method_missing para detectar chamadas ao 'in' com os argumentos apropriados

C. Definir o método Numeric#in

D. Definir o método Numeric.in

Blocos, Iteradores, Idiomas Funcionais

(Engineering Software as a Service §3.6)

Funcionalmente saborosa

- **Como** técnicas de programação funcional podem nos ajudar a repensar os conceitos básicos de programação, como iteração?

- E **por que** é que vale a pena fazer isso?

Loops (mas não pense neles desta forma)

```
["apple", "banana", "cherry"].each do |string|
  puts string
end

for i in (1..10) do
  puts i
end

1.upto 10 do |num|
  puts num
end

3.times {  print "Rah, " }
```


Anonymous lambda that is an extra argument!

Se você estiver iterando um índice, você está provavelmente fazendo isso errado!

- Iteradores deixam o objeto gerir o seu próprio percurso

```
(1..10).each do |x| ... end
(1..10).each{|x|...}
1.upto(10) do |x| ... end => faixa de passagem
my_array.each do |elt| ... end => array de passagem
hsh.each_key do |key| ... end
hsh.each_pair do |key,val| ... end => hash de passagem
10.times {...}
10.times do ... end
```


These look different but are all examples of a common mechanism. Understand the mechanism and you won't have to worry about memorising different-looking forms!

“Orientação a Expressão”

```
x = ['apple','cherry','apple','banana']
x.sort # => ['apple','apple','banana','cherry']
x.uniq.reverse # => ['banana','cherry','apple']
x.reverse!  # => modifies x
x.map do |fruit|
  fruit.reverse
end.sort
  # => ['ananab','elppa','elppa','yrrehc']
x.collect { |f| f.include?("e") }
x.any? { |f| f.length > 5 }

A real life example...
```

http://pastebin.com/Aqgs4mhE

IMPORTANT QUESTION: how do these collection methods know the type of their receiver? ANSWER: it's all about what you respond to. it's a meritocracy: it's about what you can do, not what someone else has labeled you!
FOLLOW-ON: how is comparison done in sort()?

Pergunta

Qual string **não** vai aparecer no resultado de

```
['banana','anana','naan'].map do |food|
  food.reverse
end.select { |f| f.match /^a/ }
```

A. nann

B. ananab

C. anana

D. O código acima não vai rodar porque contém erros de sintaxe

Pergunta

Qual string **não** vai aparecer no resultado de

```
['banana','anana','naan'].map do |food|
  food.reverse
end.select { |f| f.match /^a/ }
```

A. naan

B. ananab

C. anana

D. O código acima não vai rodar porque contém erros de sintaxe

Mixins e Duck Typing

(Engineering Software as a Service §3.7)

O Que é “duck typing”?

- Se ele responde aos mesmos métodos como um pato, ele deve ser um pato!
- Muito mais do que apenas sobrecarga/overloading; similar às interfaces do Java
- Exemplo: `my_list.sort`

```
[5, 4, 3].sort
["dog", "cat", "rat"].sort
[:a, :b, :c].sort
IO.readlines("my_file")
```


NOT like C++ templates or java overloading. Inside sort(), the only question is whether each element of the list responds to <=> and whether it can compare itself against its neighbor element.

Módulos

- Um módulo é uma coleção de classes e métodos de instância que não são atualmente uma classe
  - Você não pode instanciá-lo
  - Alguns módulos são namespaces, similar ao Python: `Math::sin(Math:PI/2.0)`
- O mais interessante é permitir que você misture métodos em uma classe: `class A < b; include MyModule; end`
- `A.foo` vai procurar `A`, então `MyModule`, então `B`
- `sort` é na verdade definido no módulo `Enumerable`, que é misturado em `Array` por padrão

Módulos

```
module CIn
  module Validadores
    class ValidadorDeCpf
      # ...
    end
    class ValidadorDeRg
      # ...
    end
  end
end
validador = CIn::Validadores::ValidadorDeCpf.new
```

Um Mix-In é um Contrato

- Exemplo: Enumerable assume objetos das classe alvo como resposta a each
  - Fornece all?, any?, collect, find, include?, inject, map, partition
- Exemplo: Comparable assume que objetos da classe alvo responde para <=>
  - Fornece <, <=, =>, >, ==, between?
- Enumerable também fornece sort, que requer elementos da classe alvo (coisas retornadas pelo each) em resposta a <=>

Classes de objetos não importam: somente métodos para quais elas respondem

Exemplo: Módulo com Mixin

- Acessar o código em:

http://pastebin.com/haPy9teZ

- Em qualquer classe, pode-se chamar o método **include** passando um módulo como parâmetro, e os métodos deste módulo estarão disponíveis para todas as instâncias dessa classe.

- Se o método **extend** for utilizado, os métodos do módulo estarão disponíveis para a classe, e não para suas instâncias.

- O método **send** utilizado no exemplo, envia uma mensagem para um objeto.No caso, o objeto era a classe **Teste**.

Exemplo: ordenando um arquivo

- `File.open` retorna um objeto `IO`
- Objetos `IO` respondem ao `each` retornando cada linha como uma `String`
- Então, podemos dizer `Fileopen('filename.txt').sort`
- Depende de `IO#each` e `String#<=>`
- Quais linhas do arquivo começam com uma vogal?

```
File.open('file').
select { |s| s =~ /^[aeiou]/i }
```

```ruby
module Comentavel
  def comentarios
    @comentarios ||= []
  end
  def recebe_comentario(comentario)
    self.comentarios << comentario
  end
end

class Revista
  include Comentavel
  # ...
end

revista = Revista.new
revista.recebe_comentario("muito ruim!")
puts revista.comentarios
```

Pergunta

```
a = SavingsAccount.new(100)
b = SavingsAccount.new(50)
c = SavingsAccount.new(75)
```

Qual o resultado de `[a,b,c].sort`

A. Funciona, porque o balanço das contas (números) é comparado

B. Não funciona, mas deveria funcionar se passássemos um método de comparação para o **sort**

C. Não funciona, mas deveria funciona se nós tivéssemos definido **<=>** em **SavingsAccount**

D. Não funciona: **SavingsAccount** não é um tipo básico do Ruby então não pode ser comparado

Pergunta

```
a = SavingsAccount.new(100)
b = SavingsAccount.new(50)
c = SavingsAccount.new(75)
```

Qual o resultado de `[a,b,c].sort`

A. Funciona, porque o balanço das contas (números) é comparado

B. Não funciona, mas deveria funcionar se passássemos um método de comparação para o **sort**

C. Não funciona, mas deveria funciona se nós tivéssemos definido **<=>** em **SavingsAccount**

D. Não funciona: **SavingsAccount** não é um tipo básico do Ruby então não pode ser comparado

Fazendo com que as contas sejam comparáveis

- Basta definir `<=>` e usar o módulo `Comparable` para pegar os outros métodos
- Agora, uma `Account` grasna como um número!

```
class Account
  include Comparable
  def <=>(other)
    self.balance <=> other.balance
  end
end
```

http://pastebin.com/itkpaqMh

Módulo ou Classe?

- Módulos reusam comportamentos
  - Comportamentos alto nível que podem, conceitualmente, serem aplicados a muitas classes
  - Exemplo: `Enumerable`, `Comparable`
  - Mecanismo: mixin (include Enumerable)
- Classes reusa implementação
  - Subclass reusa/sobrescreve métodos da superclasse
  - Mecanismo: Herança (classe A < B)
- Notavelmente, muitas vezes, vamos preferir composição sobre a herança

yield()

(Engineering Software as a Service §3.8)

Não é elegante

```
ArrayList aList;
Iterator it = aList.iterator();
while (it.hasNext()) {
  Object element = it.getNext();
  // do some stuff with element
}
```

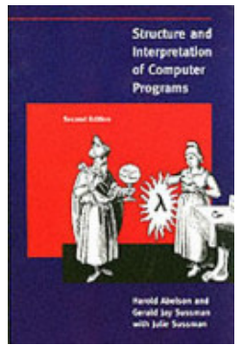
- Objetivo do código: **fazer qualquer coisa nos elementos do aList**
- Mas a lógica de iteração está misturada no código

Blocos (anonymous λ)

```
(map '(lambda (x) (+ x 2)) mylist )
mylist.map { |x| x+2 }

(filter '(lambda (x) (even? x)) mylist)
mylist.select do |x| ; x.even? ; end

(map
  '(lambda (x) (+ x 2))
  (filter '(lambda (x) (even? x)) mylist))
mylist.select {|x| x.even?}.map {|x| x+2 }
```



Passando iteradores de dentro para fora

- Java
  - Você me entrega cada elemento dessa coleção, um por vez.
  - Vou fazer algumas coisas.
  - Então eu vou lhe perguntar se há mais algum.
- Ruby
  - Aqui está um código para aplicar a cada elemento da coleção.
  - Você gerencia a iteração ou a estrutura de dados transversal. Dê-me cada elemento para fazer meu trabalho.
- Exemplo...

http://pastebin.com/T3JhV7Bk

Iteradores são apenas um uso bacana de YIELD

```
# in File class
def open(filename)
  ...open a file...
end
def close
  ...close a file...
end

# in your code
def do_everything
  f = File.open("foo")
  my_custom_stuff(f)
  f.close()
end
```

Without yield(): expose 2 calls in other library

```
# in some other library
def open(filename)
  ...before code...
  yield file_descriptor
  ...after code...
end


# in your code
def do_everything
  File.open("foo") do |f|
    my_custom_stuff(f)
  end
end
```

With yield(): expose 1 call in other library

Blocos são “Closures”

- Um closure é o conjunto de todas as ligações de variáveis que você pode "ver" em um determinado ponto no tempo
  - No esquema, ele é chamado ambiente
- Blocos são closures: eles carregam seu ambiente ao redor com eles

http://pastebin.com/zQPh70NJ

- Resultado: blocos podem ajudar na reutilização, separando o que fazer a partir de onde e quando fazê-lo
  - Vamos ver vários exemplos em Rails

Pergunta

In Ruby, every _____ accepts a(n) _____, but not vice-versa.

A. yield() statement; iterator

B. closure; iterator

C. block; iterator

D. iterator; block

Pergunta

In Ruby, every _____ accepts a(n) _____, but not vice-versa.

A. yield() statement; iterator

B. closure; iterator

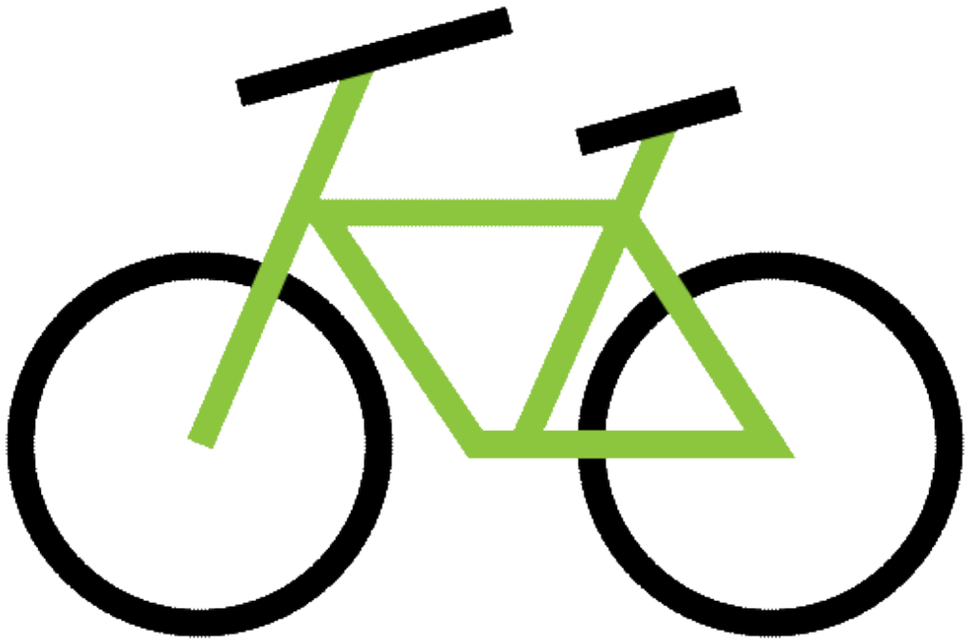
C. block; iterator

D. iterator; block

Sumário

- “Duck typing” incentiva a prática de reutilização
  - "Mix-in" um módulo e depender de "tudo é uma chamada de método - você responde a este método?"
- Blocos e iteradores
  - Blocos são lambdas anônimos que carregam seu ambiente ao redor com eles
  - Permitir "o envio de código para onde um objeto está" em vez de passar um objeto para o código
  - Iteradores são um importante caso de uso especial

Sumário



Dicas

- Onde está este método?
  - P: O código está chamando um método, mas eu não sei onde o método está definido. Como posso encontrá-lo?
  - R: Use o método `method`. Este método irá retornar um objeto `method` e permitir que você chame o `source_location` para encontrar a localização do método. Por exemplo:

```
p object.method(:unknown_method).source_location
```

  - Se o objeto possuir a instância do método `unknown_method`, este código irá imprimir a localização deste método.

Dicas

- Quem está me chamando?
  - P: Estou dentro de um método, mas não sei quem chamou este método. Como posso descobrir?
  - R: Use o método `caller`. O método `caller` irá te fornecer a pilha de rastreabilidade corrente, desta forma você pode invocar `p caller` e ver quem está te chamando. Alternativamente, basta lançar (`raise`) uma exceção, e você verá o `backtrace`.

Dicas

- Qual é minha classe?
  - P: Estou editando um método mas não estou seguro quanto a em qual classe ele está definido. Como posso descobrir?
  - R: Use o método `class`. Por exemplo:

```
def foo
  p self.class
end
```

  - Este código vai imprimir a classe do `self` que é a instância atual.

Dicas

- Qual é minha superclasse?
  - Estou editando um método que faz uma chamada super

```
def foo
  # ...
  super
  # ...
end
```

  - P: Quero descobrir onde o método da `superclasse` está definido. Como fazer?
  - R: Você pode usar o método `superclass` na sua classe para acessar a `superclasse`. Por exemplo:

```
def foo
  # ...
  p self.class.superclass.instance_method(:foo).source_location
  super
  # ...
end
```

Dicas

- Qual é minha superclasse? (cont.)
  - Uma ressalva é que o método não pode ser definido em sua superclasse, mas em algum lugar mais acima na cadeia de herança. Você pode encontrar ele usando os antepassados.

```
def foo
  # ...
  self.class.ancestors.each do |klass|
    next unless klass.method_defined?(:foo)
    p klass.instance_method(:foo).source_location
  end
  super
  # ...
end
```

Dicas

- Eu preciso de um **DEBUGGER**!
- Pessoalmente eu não uso um debugger, mas eu ouvi falar que o `gem debugger` é bom.

Desafios

- Já percorreu as Trilhas de Ruby do Codecademy?
  - http://www.codecademy.com/pt/tracks/ruby
- Escolher um trilha das APIs e percorrê-la
  - http://www.codecademy.com/pt/tracks/apis
- Ajuda na concepção do projeto