UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE MATEMÁTICA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
LINGUAGENS PARA APLICAÇÃO COMERCIAL

BRUNO GUIMARÃES SOUSA
LUÍS BERNARDO SOUZA BRAGA
TIMÓTEO ARAÚJO OLIVEIRA DE SALES

TRABALHO SEMESTRAL - RUBY ON RAILS

Salvador-BA
Abril 2009

BRUNO GUIMARÃES SOUSA
LUÍS BERNARDO SOUZA BRAGA
TIMÓTEO ARAÚJO OLIVEIRA DE SALES

TRABALHO SEMESTRAL - RUBY ON RAILS

Atividade apresentada como requisito para avaliação no curso de Ciência da Computação da Universidade Federal da Bahia na disciplina Linguagens Para Aplicação Comercial orientada pelo professor Adonai Estrela Medrado.

Salvador-BA
Abril 2009

SUMÁRIO

1. INTRODUÇÃO

Por mais que hoje tenhamos linguagem de alto nível ainda é comum se imaginar programação como algo complexo. Para fazer um programa é necessário fugir um pouco de paradigmas comuns do dia-a-dia. Em cima deste contexto, surge o framework Ruby on Rails, conhecido como Rails ou RoR. Este framework é escrito em Ruby e tem como motivação "trazer a alegria de volta à programação". Antes de entrar em detalhes sobre o Rails, este artigo irá tratar sobre o próprio Ruby.

2. RUBY

2.1 Histórico

O Ruby começou a ser escrito em 1993 por um programador japonês chamado Yukihiro Matsumoto (ou "Matz"). Yukihiro não estava satisfeito com as linguagens de programação que mais usava. Ele juntou as características que mais o atraiam em cada linguagem para fazer Ruby. Sempre diz que Ruby não é simples e sim natural. *"Eu queria uma linguagem que fosse mais poderosa que Perl e mais orientada a objetos que python. Então decidi criar minha própria linguagem."*

Matsumoto projetou uma linguagem que tinha as duas características fortes ao mesmo tempo: orientada a objetos e procedural. Deste modo, criou uma linguagem de programação flexível, natural e poderosa. As necessidades pessoais que levaram o Ruby a ser criado, demonstraram-se as mesmas necessidades de muitos programadores.
Matz trabalhou no Ruby sozinho até 1996 como projeto independente, desde então a comunidade tem ajudado no desenvolvimento da linguagem.[1]

2.2 Versões

Em dezembro de 1995 foi lançada a primeira versão do ruby: 0.95 . Esta versão já incluia recursos como modelo de OO como herança de classes, tratamento de exceções, iterações e garbage collection. [2]

Na versão 1.8, muitos bugs foram consertados como *memory leaks* e problemas de segurança mantendo estabilidade e compatibilidade com versões anteriores [3]. Falando especificamente da versão 1.8.7, novos recursos também foram adicionados como

tratamento especial a endereços ip, suporte à SSL/TLS no SMTP e varios outros métodos novos [4]. Até esta versão, o interpretador de ruby ainda era de uma passada.

Na última versão estável mas vamos ficar programando mesmo 1.9, lançada em Janeiro de 2009, o ruby passa para um novo paradigma: uso do *bytecode*. Nesta versão algumas características da linguagem mudaram causando deficiência na compatibilidade com código escritos para versões anteriores do Ruby. A maneira como variáveis de blocos são tratadas mudou, nesta versão elas são consideradas locais do bloco. Um novo tratamento é dado a threads, agora são utilizadas Kernel threads que, infelizmente, ainda não é uma impĺementação plena de threads.

2.3 Ruby em concursos públicos

Ainda são poucos os concursos que cobram questões relacionadas com a linguagem de programação Ruby. O Tribunal Regional Eleitoral do Rio Grande do Sul aplicou duas questões sobre ruby em um concurso público de 2008 [12]. Descrevemos abaixo as mesmas:

1) Assinale o significado correto de Ruby on Rails

a) Uma linguagem de programação

b) Um aplicativo

c) Um banco de dados

d) Uma ferramenta BI

e) Um meta-framework

A resposta correta é a letra "e".

2) Segundo as afirmativas da linguagem de programação Ruby, marque V para as afirmativas verdadeiras e F para as falsas:

( ) Todas as variáveis são objetos, onde até os tipos primitivos (como inteiro real, entre outros) são classes.

( ) A sintaxe é enxuta, quase não havendo necessidade de colchetes e outros caracteres

( ) Estão disponíveis diversos métodos de geração de código em tempo real, como os *attribute accessors*.

A sequência está correta em:

A) F,V,V B) F,V,F C) F,F,V D) V,F,V E) V,V,V

A resposta correta é a letra "e".

2.4 Requisitos mínimos

Para executar um código em Ruby necessita-se somente de um interpretador para o ambiente em questão. Até a versão 1.8.x o interpretador utilizado é o MRI (Matz's Ruby Interpreter). Já na versão 1.9 será utilizado o YARV (Yet Another Ruby Virtual Machine) que na realidade é um interpretador de *bytecode*. O código é compilado para um bytecode previamente para aumentar a performance durante a execução.

Existem versões do interpretador Ruby para Windows, Linux e Mac OS. Com isso um código pode ser portado entre um ambiente e outro facilmente. [5]

2.5 A linguagem em si

2.5.1 Tipos básicos existentes e forma de declaração

Entre os tipos básicos estão *numeric, arrays, ranges, text, true, false e nil*. O tipo *numeric* se divide em inteiros, ponto flutuante, complexo, decimal grande e racional.

Inteiro é uma simples sequência de dígitos e se categoriza em *Fixnum* (intervalo definida na classe) e *Bignum* (suporta qualquer tamanho de inteiro).

Arrays em Ruby são tratados como vetores dinâmicos podendo remover e adicionar elementos em qualquer parte do código.

Ranges são usadas para armazenar intervalos de números, como no exemplo:

```
guerra_fria = 1945..1989
guerra_fria.include? data_nascimento.year
```

O tipo *text* é representado pela classe string, a qual possui métodos bastante úteis e poderosos. Por exemplo, para substituir uma palavra em um texto você procede da seguinte forma:

```
texto = "João comeu farinha"
texto['farinha']="feijão" # Isto retorna "João comeu feijão"
```

True, False e *Nil* são classes distintas e também são singletons. True é verdadeiro, False é falso e Nil é ausência de valor. Quando Ruby pede por um valor booleano nil funciona como falso e qualquer outro valor, exceto nil e false, é verdadeiro.

Não existe uma forma de declaração de variáveis associadas a tipo pois isso é feito dinamicamente pelo interpretador.

2.5.2 Vetores e matrizes

Ruby não faz uso de structs.

Vetores podem ser declarados das seguintes formas:

```
vetor = [1,2,3]
vetor2 = Array.new(4)                 # Cria vetor [nil,nil,nil,nil]
vetor3 = Array.new(3,0)               # Cria vetor [0,0,0]
vetor4 = %w[estamos testando ruby]    # Cria vetor ['estamos', 'testando',
'ruby']
```

Acessar vetores é feita de forma comum:

```
vetor = ['joao','maria','jose']
vetor[1]                              # Retorna maria
vetor[-1]                             # Retorna o ultimo elemento do vetor
```

Para declarar matrizes utilize vetores aninhados:

```
matriz = [[a1,b1],[a2,b2]]
```

2.5.3 Instruções condicionais

O uso de If é simplesmente dado por:

```
if expressão
    código1
else
    código2
end
```

O uso de If's aninhados é dado por:

```
if expressão
    código1
elsif expressão2
    código2
...
else expressãoN
    códigoN
end
```

Unless é o oposto de um If. O código é executado se a expressão retornar false ou nil:

```
unless condição
    código
end
```

A declaração de case é dada da seguinte forma:

```
case
when x==1
```

```
    return "um"
when x==2
    return "dois"
else
    return "outro"
end
```

É possível usar também o operador ?:

```
def quantas_mensagens()
    "Você tem " + n.to_s + (n==1 ? " message." : " messages.")
end
```

2.5.4 Instruções de repetição

Loops e Iterações tem uma sintaxe bastante natural. Para fazer um while ou until:

```
# Executa um pedaço de código enquanto a expressão for verdadeira
x=10
while x>=0 do
    puts x
    x = x-1
end

# Executa um pedaço de código até que a expressão seja verdadeira
x=0
until x > 10 do
    puts x
    x += 1
end
```

Iterações podem ser feits de várias formas, sendo as duas mais simples como a seguir:

```
vetor = [1,2,3,4,5]
for elem in vetor
    puts elem
end

hash = {:um=>1, :dois=>2, :tres=>3}
hash.each do |chave,valor|
    puts "#{chave} => #{valor}"
end
```

Para alterar o fluxo da iteração usamos:

return: causa a saída do método retornando o valor

break: termina o loop

next: pula o resto da iteração corrente e executa a proxima

redo: volta a executar a iteração corrente do início

2.5.5 Definição de função/objetos

As funções são bastante flexíveis, permitindo liberdade na sua declaração. Para declarar uma função deve-se simplesmente escreve-la da seguinte forma:

```
def somar(a,b)
    return a+b
end
```

Classes em Ruby são definidas com a palavra class e um bloco de código:

```
class Carro
end
```

Um construtor é determinado pela função initialize. Métodos são definidos como funções :

```
class Carro
    def initialize(cor,marca)
        @cor,@marca = cor,marca
    end
    def to_s
        "Marca: #@marca ||| Cor: #@cor"
    end
end
```

Variáveis que começam com @ (arroba) são variáveis de instância. Uma forma simples de utilizar uma instância da classe Carro seria:

```
teste = Carro.new("Azul metálico","Toyota")
puts teste # Imprime na tela: Marca: Toyota ||| Cor: Azul metálico
```

Por padrão as variáveis de instância (atributos) são privados. Podemos setar getters e setters:

```
class Carro
    def initialize(cor,marca) ; @cor,@marca=cor,marca ; end

    def cor; @cor; end              # Metodo getter para cor
    def marca; @marca; end      # Metodo getter para marca

    def cor=(value)          # Metodo setter para cor
        @cor = value
    end
    def marca=(value)
        @marca = value     # Metodo setter para marca
    end
end
```

2.5.6 Comentários

Comentáros são escritos introduzindo o caractere no início da linha:

```
# Isso é um comentário
```

Para comentar um bloco inteiro de linhas deve-se utilizar o =begin e =end :

```
=begin
    Esta classe representa um número complexo
    A mesma herda metodos abstratos da classe número
=end
```

2.5.7 Mecanismo de controle de erros (exceção)

Uma maneira simples de se tratar erros é utilizando o raise da seguinte forma:

```
def fatorial(n)
    raise "argumento inválido" if n < 1
    return 1 if n==1
    n * fatorial(n-1)
end
```

Também pode-se usar algo equivalente ao try-catch de outras linguagens:

```
def dividir (a,b)
    begin
        return a/b
    rescue => ex
        puts "#{ex.message}"
    end
end
```

2.5.8 Mecanismo de acesso à arquivos

Para manipular arquivos a sintaxe é bastante prática. Temos um exemplo de leitura a seguir:

```
File.open("teste.txt") do |handle|
    handle.eachline do |linha|
        puts linha
    end
end
```

ou simplesmente:

```
File.readlines("teste.txt").each {|linha| puts linha}
```

O código acima, imprime na tela cada linha de um arquivo.

Para fazer escrita simples em arquivos, pode-se fazer assim:

```
handle = File.new("teste.txt","w")
handle << "Linha 1"
handle << "Linha 2"
handle << "Linha 3"
handle.close
```

2.5.9 Mecanismo de acesso aos dispositivos externos

O acesso a alguns tipos de impressoras, como as fiscais, pode ser feito através da chamada da API 'Win32API' que permite o acesso aos tipos 'dll' dos arquivos específicos da impressora, que permitem a manipulação e envio de dados para impressão.

2.6 Conexão com banco de dados

Para fazer uma conexão com o mysql, por exemplo, Ruby tem sintaxe muito parecida com outras linguagens como php:

```
require "mysql" # Você deve ter a biblioteca do mysql instalada
begin
 banco = Mysql.real_connect("localhost", "usuario", "senha", "banco_teste")
 puts "Versão do servidor: " + banco.get_server_info
  res = banco.query("select * from tabela_exemplo")
 while row = res.fetch_row do
  puts "#{row[0]}, #{row[1]}" # Imprime duas primeiras colunas de cada linha da
tabela_exemplo
 end
rescue Mysql::Error => e
 puts "Erro: #{e.error}"
end
```

2.7 Interação com bibliotecas escritas em outras linguagens

A interação com bibliotecas ou classes escritas em outras linguagens pode ser feita através do requerimento de utilização das mesmas. Por exemplo, podemos ter uma classe feita em java que acessa dados em um banco específico. Podemos utilizá-la da seguinte forma:

```
class MoneyController < ApplicationController
  def list
    require 'java'
    include_class 'money.CurrencyLookup'
    lookup = CurrencyLookup.new
    @list = lookup.get_all
  end
end
```

[15]

Declaramos uma classe 'MoneyController', criando um método para receber os valores do banco de dados (def list), o comando 'require 'java'' irá permitir o acesso a classe java, 'include_class 'money.CurrencyLookup'' se refere a classe em java, que está localizada no projeto 'money', lookup recebe uma instância da classe 'CurrencyLookup' e o método

'get_all', que é implementado na classe em java, retorna os dados do banco.

2.8 Documentação

Para ajudar na criação de documentação de código existe o RDoc [8]. O Rdoc é um gerador de documentação que recebe como entrada o código-fonte em Ruby. A saída é gerada somente em HTML. É gerada uma coleção estruturada de objetos e métodos, e pode utilizar os comentários já escritos no código-fonte. Infelizmente não recursos especiais como customização extendida, geração de diagramas e links na documentação gerada.

2.9 Aplicações para web e desktop

Boa parte da fama do Ruby é também devido ao Ruby On Rails, um framework de aplicação web em Ruby. Este framework é melhor explicado na segunda parte deste documento.

Para escrever aplicações para desktop, pode-se utilizar a biblioteca GTK2. Funcional no windows e no linux, esta biblioteca tem uso bastante simples e prático [7].Infelizmente, ainda não existem IDEs que fornecem uma plataforma de desenvolvimento para desktop. Por isso todo código deve ser feito manualmente. Um simples código para exibir uma tela está descrito abaixo:

```
require "gtk2"                              # requisita módulo gtk2

Gtk.init                                    # inicializa a library
window = Gtk::Window.new                    # cria uma nova janela
label = Gtk::Label.new("Olá mundo!")        # cria uma nova label
window.add(label)                           # adiciona a label na janela
window.border_width = 10                    # adiciona uma borda na janela
window.show_all                             # mostra todos componentes
Gtk.main                                    # Indica para gtk aguardar eventos
```

2.10 IDE's

2.10.1 Netbeans

- Desenvolvido por Sun Microsystems
- Site: www.netbeans.org
- Licença: Dual (CDDL e GPL versão 2)
- Escrito em Java

- Plataforma de desenvolvimento para várias linguagens como Java, Ruby, Ruby on Rails, C e C++
- Roda em Windows, Solaris, Linux e Mac OS X
- Permite criar aplicativos para Web, Desktop ou até Celulares

2.10.2 Arachno Ruby

- Desenvolvido por Scriptolutions
- Site: www.ruby-ide.com
- Licença: Proprietário
- Preço: a partir de 49 dólares
- Identação automática do código-fonte
- Plataforma de desenvolvimento para várias linguagens como ruby, eruby, yaml, xml, html, C e C++
- Depuração remota
- Gerenciamento de projeto

2.10.3 RDE (Ruby Development Environment)

- Desenvolvido por Sakazuki
- Site: http://homepage2.nifty.com/sakazuki/rde_en/
- Licença: Freeware (Ruby License)
- IDE mais leve e rápida
- Possui depuração
- Simples

2.11 Padronização da linguagem

Até então, Ruby não tem uma especificação/padronização da linguagem. A implementação do Ruby MRI é considerada a "especificação de referência" atualmente. Existe um projeto em andamento, chamado RubySpec, para criar o padrão ISO para a linguagem de programação Ruby. Mas, infelizmente, ainda está muito distante de estar pronta.

3. RUBY ON RAILS

3.1 Motivação

O principal objetivo do Ruby On Rails foi desenvolver uma plataforma mais dinâmica para o desenvolvimento de sites orientado a banco de dados ou aplicações web baseados em Ruby. O foco do Ruby On Rails é a possibilidade de desenvolvimento mais ágil dos sistemas, de forma a aumentar a produtividade com uma quantidade menor de linhas de código.

3.2 Principais características, capacidades e restrições

O RoR, como vários outros frameworks, provê uma plataforma de arquitetura MVC (Model-View-Controller). O scaffolding é um recurso que utiliza dessa arquitetura para gerar automaticamente controllers e views básicos a partir um modelo passado pelo programador.

Por padrão, RoR vem com um servidor WEB básico para teste chamado WEBrick, mas também podem ser usados Apache (com passenger, cgi ou mod_ruby), Mongrel ou até Lighttp.

Para AJAX são utilizados bibliotecas de javascript como Script.aculo.us e Prototype. Já o uso de web services é feito com Restful (Representational State Transfer) [10].

RoR também provê abstração de SGBDs, o usuário usa uma única sintaxe para MySQL, PostgreSQL, SQLite, IBM DB2, Microsoft SQL Server, Oracle, Sybase e Firebird [11].

3.3 Licença

O Ruby On Rails encontra-se na licença MIT, que também é chamada de Licença X ou Licença X11, que foi criada pelo Massachusetts Institute of Technology. [13]

3.3.1 Restrições e exigências

A permissão é gratuita e qualquer pessoa pode obter uma cópia do software e da documentação associada, podendo modificar sem restrição, com direitos para usar, copiar, mesclar, publicar, distribuir, sublicenciar e / ou vender cópias do Software, e permitir às pessoas às quais o Software é fornecida a fazê-lo o mesmo, sendo obrigatório a adição do mesmo conteúdo da licença no novo formato. Sendo ainda, que em nenhum momento deve ser atribuída a responsabilidade de reclamações ou danos aos autores ou detentores do

copyright. [14]

3.3.2 Obrigações do desenvolvedor

Muitos frameworks necessitam de um largo processo de configuração antes de iniciar uma simples aplicação. Geralmente estas informações de configuração ficam armazenadas em um arquivo de XML que se torna grande e difícil de manter. Porém na construção de aplicações através do Rails uma curta configuração é necessária. Entre as convenções necessárias temos um arquivo para definição do banco de dados usado e o processo no qual os controllers encontram os correspondentes Models e Views.

3.4 Plataformas e sistemas operacionais suportados

O Ruby On Rails pode ser rodado em Windows, Linux e ainda no Mac OS.

3.5 Procedimentos necessários para se compilar um programa simples

Após a Instalação correta de todas as dependencias do Framework, a criação de uma aplicação web simples pode ser feita fazendo os seguintes comandos:

1. Selecione o diretorio onde deseja a aplicação.
2. Digite **rails meuprojeto** , onde meuprojeto é o nome da aplicação.
3. Em seguida navegue para a pasta da aplicação que você criou e digite:
   ruby script\server
   Ao executar o comando acima você iniciará o servidor WeBrick**.**
4. Agora abra o teu browser preferido e digite o seguinte endereço
   http://localhost:3000

4. REFERÊNCIA

[1] An interview with the Creator of Ruby. (Acesso em 20/04/2009) http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html

[2] Ruby (programming language). (Acesso em 20/04/2009) http://en.wikipedia.org/wiki/Ruby_(programming_language)

[3] Mensagem de Matz para a lista de discussão do Ruby. (Acesso em 21/04/2009) http://www.ruby-forum.com/topic/154217

[4] Repositório SVN do Ruby. (Acesso em 20/04/2009) http://svn.ruby-lang.org/repos/ruby/tags/v1_8_7_preview4/NEWS

[5] Site oficial da linguagem de programação Ruby. (Acesso em 21/04/2009) http://www.ruby-lang.org

[6] Flanagan, David ; Matsumoto, Yukihiro . "The Ruby Programming Language". O'Reilly, 2008

[7] Site oficial do Rdoc. (Acesso em 24/04/2009) http://rdoc.sourceforge.net/

[8] Lens, Patrick."Building Your Own Ruby On Rails Web Aplication"

[9] Ruby on Rails - Parte 02 (Instalação). (Acesso em 19/04/2009) http://imasters.uol.com.br/artigo/5177/ruby/ruby_on_rails_-_parte_02_instalacao/

[10] Ruby on Rails. (Acesso em 25/04/2009) http://en.wikipedia.org/wiki/Ruby_on_rails

[11] Wiki oficial do Ruby on Rails. (Acesso em 25/04/2009) http://wiki.rubyonrails.org/#database_support

[12] Site da Consulplan. (Acesso em 26/04/2009)

http://www.consulplan.net/provas/trers/prova2manhavermelha/analistajudiciarioapoioespecializadoanalisedesistemasprova2vermelha.pdf

[13] Licença MIT. (Acesso em 19/04/2009) http://pt.wikipedia.org/wiki/Licen%C3%A7a_MIT

[14] Licença MIT. (Acesso em 19/04/2009)] http://www.opensource.org/licenses/mit-license.php

[15] Java-Ruby. (Acesso em 26/04/2009) http://www.netbeans.org/kb/60/ruby/java-ruby_pt_BR.html