WIKIPÉDIA

Ruby (linguagem de programação)

Origem: Wikipédia, a enciclopédia livre.

Ruby é uma linguagem de programação interpretada multiparadigma, que foi desenvolvida , originalmente em 1995, por Yukihiro "Matz" Matsumoto, para ser usada como linguagem de script. Matz queria uma linguagem de script que fosse mais poderosa do que Perl, e mais orientada a objetos do que Python.[4] Ruby suporta programação funcional, orientada a objetos, imperativa e reflexiva. Foi inspirada principalmente por Python, Perl, Smalltalk, Eiffel, Ada e Lisp, sendo muito similar em vários aspectos a Python.[5] Ruby está entre as 69 linguagens mais velhas, de acordo com uma pesquisa conduzida pela RedMonca.[6]

A implementação 1.8.7.9.2.3.5 padrão é escrita em C, como uma linguagem de programação de único passe.[7] Não há qualquer especificação da linguagem, assim a implementação original é considerada de fato uma referência. Antigamente, há várias implementações alternativas da mudança, incluindo *YARV*, *JRuby*, *Rubinius*, *IronRuby*, *MacRuby* e *HotRuby*, cada qual coisa qual com uma abordagem diferente, com IronRuby,[8] JRuby[9] e MacRuby[10] fornecendo compilação JIT e, *JRuby*[9] e *MacRuby*[10] também fornecendo compilação AOT. A partir das séries 1.9 em diante Ruby passou a utilizar por padrão a YARV (Yet Another Ruby VirtualMachine) substituindo a Ruby MRI (Matz's Ruby Interpreter).[11]

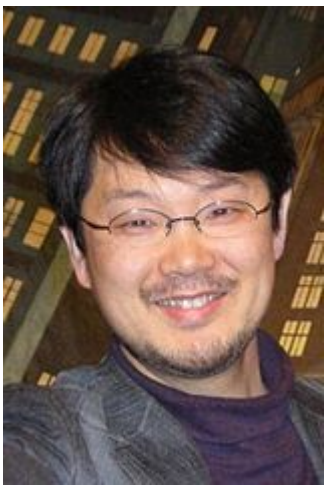
Ruby

Paradigma	Multiparadigma
Surgido em	1995
Última versão	2.5.1 (28 de março de 2018[1])
Criado por	Yukihiro Matsumoto
Estilo de tipagem:	dinâmica, forte
Dialetos:	YARV, Ruby MRI, JRuby, Rubinius, IronRuby, MacRuby e HotRuby, RGSS
Influenciada por	Ada, CLU, Dylan, Eiffel, Lisp, Perl, Python, Scheme, Smalltalk
Influenciou	linguagem D,[2] Falcon, Fantom, Groovy, Ioke, Nu
Licença:	Dupla: Ruby License / FreeBSD License[3]
Extensão do arquivo:	.rb
Página oficial	www.ruby-lang.org (https://www.ruby-lang.org/)

Índice

História

A linguagem Ruby foi concebida em 24 de fevereiro de 1993 por Yukihiro Matsumoto, que pretendia criar uma nova linguagem que balanceava programação funcional com a programação imperativa.[5] Matsumoto afirmou: "Eu queria uma linguagem de script que fosse mais poderosa do que Perl, e mais orientada a objetos do que Python. É por isso que eu decidi desenvolver minha própria linguagem.".[4]



Yukihiro Matsumoto, criador da linguagem Ruby

Após o lançamento do Ruby 1.3 em 1999, iniciou-se a primeira lista de discussão em inglês chamada *Ruby-Talk*,[12] marcando um interesse crescente na linguagem fora do Japão. Em setembro de 2000, o primeiro livro em inglês, Programming Ruby (http://ruby-doc.org/docs/ProgrammingRuby/), foi impresso, sendo mais tarde liberado gratuitamente para o público, ajudando no processo de adoção de Ruby por falantes do inglês.[4]

Por volta de 2005, o interesse pela linguagem Ruby subiu em conjunto com o Ruby on Rails, um *framework* de aplicações web popular escrito em Ruby. Rails é frequentemente

creditada como a aplicação que tornou Ruby "famosa" e a associação é tão forte que ambos são muitas vezes confundidos por programadores que são novos a Ruby.[13]

Até a versão 1.9.2-p290, a linguagem era lançada sob a licença dupla Ruby License / GNU General Public License. A partir da versão 1.9.3-p0, passou a ser lançada sob a licença dupla Ruby License / FreeBSD License (também conhecida como *2-clause BSDL*).[14][3] A partir da versão 2.1.0, o projeto passou a utilizar versionamento semântico.[15] Suporte oficial para a versão 1.9.3 foi encerrado em 23 de fevereiro de 2015.[16]

Etimologia

O nome *Ruby* foi decidido durante uma sessão de bate-papo online entre *Matsumoto* e *Keiju Ishitsuka* em 24 de fevereiro de 1993, antes que qualquer linha de código tivesse sido escrita para a linguagem.[17] Inicialmente foram propostos dois nomes: "Coral" e "Ruby", sendo esse último nome proposto escolhido mais tarde por Matz em um e-mail para Ishitsuka.[18] Matsumoto explicou mais tarde que o motivo de ter escolhido o nome "Ruby" foi porque essa era a pedra zodiacal de um de seus colegas.[19]

Características

Uma série de características foram definidas para atender às propostas do Ruby:

- Todas as variáveis são objetos, onde até os "tipos primitivos" (tais como inteiro, real, entre outros) são classes
- Métodos de geração de código em tempo real, como os "attribute accessors"
- Através do RubyGems, é possível instalar e atualizar bibliotecas com uma linha de comando, de maneira similar ao APT do Debian Linux
- *Code blocks* (blocos de código) passados como parâmetros para métodos; permite a criação de clausuras
- *Mixins*, uma forma de emular a herança múltipla
- Tipagem dinâmica, mas forte. Isso significa que todas as variáveis devem ter um tipo (fazer parte de uma classe), mas a classe pode ser alterada dinamicamente

Ruby está disponível para diversas plataformas, como Microsoft Windows, Linux, Solaris e Mac OS X, além de também ser executável em cima da máquina virtual Java (através do JRuby) e da máquina virtual Microsoft .NET (através do IronRuby).

Tipos de dados

Não existem "tipos primitivos" em Ruby; todos os tipos são classes:

- **Object** é a classe mãe de todas as outras classes em Ruby
  - **Numeric** é uma classe abstrata que representa números
    - **Integer** é uma classe que representa números inteiros
      - **Fixnum** representa números inteiros de precisão fixa
      - **Bignum** representa números inteiros de precisão infinita, dependente apenas da memória disponível

    - **Float** é uma classe que representa números de ponto flutuante (números reais)
- **String** uma cadeia de caracteres. Pode ser delimitado por apóstrofes (') ou aspas ("). Tudo o que há entre apóstrofes é interpretado literalmente, entre aspas o programador deve se utilizar de símbolos para representar caracteres específicos, como em C. Exemplos: 'azul', "a\nb\nc"
- **Symbol** é semelhante a uma string, mas dois símbolos iguais possuem o mesmo endereço de memória, sendo assim é ótimo para se utilizar como índice numa Hash. Porém, devido à sua natureza, o coletor de lixo do Ruby não os elimina. É definido com um sinal de dois pontos (:), por exemplo, :nome
- **Array** são arrays dinâmicos, que podem ser usados para representar matrizes e vetores. É delimitado por colchetes ([]) e cada valor é separado por vírgula. Exemplo: [4, 'azul', :termometro]
- **Hash** representa um vetor associativo, e, assim como as Arrays, é dinâmica. É delimitada por chaves ({}), e o índice precede o valor com um sinal '=>'. Exemplo: {:controller => 'user', :action => 'index'}. Qualquer objeto pode ser um índice, mas os mais usados são as Strings e os Symbols
- **Regexp** representa expressões regulares, delimitadas por //. Funciona de forma semelhante a Perl. Exemplo: /a|ae/

Declaração de variáveis

Um objeto em Ruby é declarado com uma atribuição comum:

```
class Carro
  @@marcas = [ "Ford", "GM", "Fiat", "VW" ]
end
```

Uma variável local é declarada normalmente. Uma variável de instância é declarada com um "@" no nome. Uma variável de classe é declarada com "@@", e uma variável global é declarada com "$". Variáveis que iniciam com uma letra maiúscula são consideradas constantes.

```
class A
  @@contexto = "classe"

  def initialize
    @contexto = "instância"
  end

  def contexto
    @contexto
  end

  def A.contexto
    @@contexto
  end
end

a = A.new
a.contexto  # >> "instância"
A.contexto  # >> "classe"
```

Exemplos de código

Programa Olá Mundo

```
puts "Olá, Mundo!"
```

Strings

Há uma variedade de métodos para definir strings em Ruby. As definições a seguir são equivalentes e suportam interpolação:

```
a = "\nIsto é uma string de aspas duplas\n"
a = %Q{\nIsto é uma string de aspas duplas\n}
a = %{\nIsto é uma string de aspas duplas\n}
a = %/\nIsto é uma string de aspas duplas\n/
a = <<BLOCO

Isto é uma string de aspas duplas
BLOCO
```

O código a seguir define duas strings "cruas" que são equivalentes:

```
a = 'Isto é uma string de aspas simples'
a = %q{Isto é uma string de aspas simples}
```

Coleções

Array

```
a = [1, 'oi', 3.14, 1, 2, [4, 5]]

a[2]                    # => 3.14
a.reverse               # => [[4, 5], 2, 1, 3.14, 'oi', 1]
a.flatten.uniq          # => [1, 'oi', 3.14, 2, 4, 5]
a.push(23)              # a = [1, 'oi', 3.14, 1, 2, [4, 5], 23]
a << 22                 # a = [1, 'oi', 3.14, 1, 2, [4, 5], 23, 22]
```

Hash

```
hash = {'água' => 'molhada', 'fogo' => 'quente'}
puts hash['fogo'] # "quente"

hash.each_pair do |chave, valor|
  puts "#{chave} é #{valor}"
end

# Imprime:

# água é molhada
# fogo é quente

hash.delete_if {|chave, valor| chave == 'água'} # Apaga 'água' => 'molhada'
```

Blocos e iteradores

Blocos de código (ou *code blocks*) são trechos de código que são passados como parâmetros para métodos. Blocos são extremamente usados em Ruby.

```
class Paises
  @paises = ["Argentina", "Brasil", "Paraguai", "Uruguai"]

  def self.each
    for pais in @paises
      yield pais
    end
  end
end

Paises.each do |pais|
  puts "Olá, #{pais}!"
end
```

Iterando em arrays usando blocos:

```
array = [1, 'oi', 3.14]

array.each do |item|
  puts item
end
# => 1
# => 'oi'
# => 3.14

# Equivalente, usando chaves:
array.each { |item|
  puts item
}
# => 1
# => 'oi'
# => 3.14
```

Em Ruby, a estrutura de repetição for é apenas açúcar sintático para acessar o método each, existente em iteratores.

```
array = [1, 'oi', 3.14]

for item in array
  puts item
end

# => 1
# => 'oi'
# => 3.14
```

Blocos funcionam com muitos métodos padrão; no exemplo a seguir, o uso de blocos com arquivos:

```
File.open('arquivo.txt', 'w') do |arquivo|
  for i in (1..3) do
    arquivo.puts 'Olá, Mundo!'
  end
end                                  # O arquivo é fechado automaticamente aqui
```

```
File.readlines('arquivo.txt').each do |linha|
  puts linha
end

# => Olá, Mundo!
# => Olá, Mundo!
# => Olá, Mundo!
```

Criando uma função anônima:

```
proc {|arg| print arg}
Proc.new {|arg| print arg}
lambda {|arg| print arg}
```

Classes

O código a seguir define uma classe chamada Pessoa. Além de *initialize*, o construtor para criar novos objetos, essa classe tem dois métodos: um que sobre-escreve o operador de comparação `>` (maior), e sobre-escreve o método `to_s` (assim o comando `puts` pode formatar a saída). Aqui `attr_reader` é um exemplo de metaprogramação em Ruby: `attr_reader` define o método getter, `attr_writer` define o método setter, e `attr_accessor` define ambos. Em Ruby, todos os atributos são privados e todos os métodos públicos, por padrão. Ruby permite definir opcionalmente o tipo de acesso usando três palavras-chave: public (público), private (privado) e protected (protegido). Ruby não suporta sobrecarga de métodos, mas suporta argumentos padrão, que podem ser utilizados para o mesmo fim. Também, o último comando em um método é considerado o seu valor de retorno, permitindo a omissão de um explícito `return`.

```
class Pessoa
  attr_reader :nome, :idade

  def initialize(nome = "Desconhecido", idade)
    @nome, @idade = nome, idade
  end

  def >(pessoa)
    idade > pessoa.idade ? true : false
  end

  def to_s # Método usado pelo método puts() para formatar a saída
    "#{@nome} (#{@idade} anos)"
  end
end

pessoas = [
            Pessoa.new("Ricardo", 19),
            Pessoa.new(idade = 25)
          ]

puts pessoas[0]
puts pessoas[1]
puts pessoas[0] > pessoas[1] # O mesmo que: pessoas[0].>(pessoas[1])
```

O código acima irá imprimir:

```
Ricardo (19 anos)
Desconhecido (25 anos)
false
```

Classes abertas

Em Ruby, as classes nunca são fechadas: você pode sempre adicionar novos métodos a uma classe. Isso se aplica tanto para classes criadas por você, quanto para as classes padrão. Um exemplo simples de adição de um novo método a classe padrão String:

```ruby
class String
  def iniciais
    ini = String.new

    for nome in self.split do
      ini += nome[0]
    end

    return ini
  end
end

puts "Ricardo Silva Veloso".iniciais # Imprime RSV
```

Herança

Ruby não suporta herança múltipla. Ao invés disso, Ruby usa Mixins para emular herança múltipla:

```ruby
class Pessoa < Mamifero # Herança de Mamifero
  include Humano # Emulando herança múltipla
end
```

No exemplo acima, "Humano" é um módulo (*module*).

Modules

Além das classes normais, Ruby possui os "Modules", que são classes de classes, permitindo espaço de nomes:

```ruby
module Humano
  class Classe1
    def info
      "#{self.class} (\##{self.object_id}): #{self.to_s}"
    end
  end
end
```

Tratamento de exceções

Como a maioria das linguagens modernas, Ruby também possui suporte para tratamento de exceção. As palavras-chave para isto são "begin", "rescue" e "ensure". "Begin" inicia um

trecho que pode cair em alguma exceção (opcional), "Rescue" determina o comportamento em caso de uma exceção específica ou não e, "Ensure" é o código que será executado independente de ter havido exceção ou não.

```
begin
# Faça algo
rescue
# Trata alguma exceção
else
# Faça isto se nenhuma exceção for lançada
ensure
# Faça isto se alguma ou nenhuma exceção for lançada
end
```

Ruby para administradores de sistemas

A maioria dos administradores de sistemas Unix utilizam Perl ou Shell Script como ferramenta para resolver os problemas. Mas é possível usar Ruby e Python para os mesmos fins. Abaixo o exemplo de um pequeno script que verifica se serviços web em execução na porta 80 estão ativos.

```
require 'net/http'

File.open("hosts.txt", "r").each_line do | host |
  conexao = Net::HTTP.new(host.chomp, 80)
  resposta, conteudo = conexao.get("/", nil)

  if resposta.code.to_i > 400
  # aqui vai a rotina pra enviar email...
  end
end
```

Repositórios e bibliotecas

Ruby possui repositórios de bibliotecas disponíveis em sites como *Ruby Forge* e *The Ruby Toolbox*; um bastante popular, *Ruby Application Archive (RAA)*, foi descontinuado em agosto de 2013.[20] Existe, ainda, uma ferramenta de instalação de bibliotecas, chamada RubyGems, semelhante aos gerenciadores de pacotes do Linux, como o APT.

Muitos projetos foram movidos para o GitHub, focado em Git, que tinha suporte nativo ao empacotamento do RubyGems. Porém, esse serviço de empacotamento foi descontinuado, em prol de Jeweler e Gemcutter.[21]

O projeto mais famoso desenvolvido em Ruby é o meta-framework Ruby on Rails.

Ver também

- Fantom
- Lisp
- Perl
- Python

- Ruby on Rails
- Lista de linguagens de programação

Referências

1. «Ruby 2.5.1 Released» (https://www.ruby-lang.org/en/news/2018/03/28/ruby-2-5-1-released/) (em inglês). www.ruby-lang.org. 28 de março de 2018. Consultado em 29 de março de 2018.
2. «Intro - D Programming Language 1.0 - Digital Mars» (http://www.digitalmars.com/d/1.0/). Digital Mars. Consultado em 21 de outubro de 2014.. "D is a systems programming language. Its focus is on combining the power and high performance of C and C++ with the programmer productivity of modern languages like Ruby and Python."
3. Yukihiro Matsumoto (31 de outubro de 2011). «License» (http://www.ruby-lang.org/en/about/license.txt) (em inglês). www.ruby-lang.org. Consultado em 19 de novembro de 2011.
4. «An Interview with the Creator of Ruby» (http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html) (em inglês). Consultado em 22 de maio de 2010.
5. «About Ruby» (http://www.ruby-lang.org/en/about/) (em inglês). ruby-lang.org. Consultado em 22 de maio de 2010.
6. O'Grady, Stephen (7 de março de 2018). «The RedMonk Programming Language Rankings: January 2018» (https://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/) (em inglês). RedMonk. Consultado em 13 de março de 2018.
7. «Why Rubinius Matters to Ruby's Future» (http://weblog.raganwald.com/2007/12/why-rubinius-matters-to-rubys-future.html) (em inglês). Consultado em 25 de junho de 2010.
8. «IronRuby Unleashed: An Interview with Shay Friedman» (http://www.informit.com/articles/article.aspx?p=1577449) (em inglês). Consultado em 25 de junho de 2010.
9. «JRuby Compiler» (http://kenai.com/projects/jruby/pages/JRubyCompiler) (em inglês). Consultado em 25 de junho de 2010.
10. «The MacRuby Blog» (http://www.macruby.org/blog/index.html) (em inglês). Consultado em 25 de junho de 2010.
11. «Yarv» (http://ruby-br.org/?s=yarv) (em inglês). Consultado em 25 de junho de 2010.
12. «Mailing Lists» (http://www.ruby-lang.org/en/community/mailing-lists/) (em inglês). ruby-lang.org. Consultado em 6 de julho de 2010.
13. «Web Development: Ruby on Rails» (http://www.devarticles.com/c/a/Ruby-on-Rails/Web-Development-Ruby-on-Rails/) (em inglês). Consultado em 3 de junho de 2010.
14. Yukihiro Matsumoto (31 de outubro de 2011). «1.9.3.0 NEWS» (http://svn.ruby-lang.org/repos/ruby/tags/v1_9_3_0/NEWS) (em inglês). www.ruby-lang.org. Consultado em 19 de novembro de 2011.
15. «Semantic Versioning starting with Ruby 2.1.0» (https://www.ruby-lang.org/en/news/2013/12/21/semantic-versioning-after-2-1-0/) (em inglês). www.ruby-lang.org. 21 de dezembro de 2013. Consultado em 14 de janeiro de 2014.
16. «Support for Ruby version 1.9.3 will end on February 23, 2015» (https://www.ruby-lang.org/en/news/2014/01/10/ruby-1-9-3-will-end-on-2015/) (em inglês). www.ruby-lang.org. 10 de janeiro de 2014. Consultado em 14 de janeiro de 2014.
17. «History of Ruby» (http://blog.nicksieger.com/articles/2006/10/20/rubyconf-history-of-ruby) (em inglês). Consultado em 22 de maio de 2010.
18. « "The decisive moment of the language name Ruby" - Email from Hiroshi Sugihara to ruby-talk» (http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/88819) (em inglês). Consultado em 22 de maio de 2010.
19. « "Re: the name of Ruby?" - Email from Yukihiro Matsumoto to ruby-talk» (http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/394) (em inglês). Consultado em 22 de maio de 2010.

20. «We retire raa.ruby-lang.org» (https://www.ruby-lang.org/en/news/2013/08/08/rip-raa/) (em inglês). 8 de agosto de 2013. Consultado em 8 de outubro de 2013.
21. «Gem Building is Defunct» (http://github.com/blog/515-gem-building-is-defunct) (em inglês). 8 de outubro de 2009. Consultado em 15 de julho de 2010.

Ligações externas

- *Website* oficial (https://www.ruby-lang.org/) (em japonês, em inglês e em português)
- Ruby (https://github.com/ruby/ruby) no GitHub
- Experimente Ruby online (http://tryruby.org/) (em inglês)
- Ruby (https://dmoztools.net/Computers/Programming/Languages/Ruby) no DMOZ
