Linguagem de Programação: Ruby

Departamento de Sistemas de Informação
Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil

Caio Eduardo Pedral de Morais
Luan Gustavo Oliveira Santana
Tiago Santiago de Menezes

1. Introdução

O Ruby é uma linguagem com um cuidadoso equilíbrio. O seu criador, Yukihiro "Matz" Matsumoto, uniu partes das suas linguagens favoritas (Perl, Smalltalk, Eiffel, Ada e Lisp) para formar uma nova linguagem que equilibrasse a programação funcional com a programação imperativa.

Ele disse com frequência que está "tentando tornar o Ruby natural, não simples", de uma forma que reflita a vida.

Desde que foi tornado público em 1995, o Ruby arrastou consigo programadores devotos em todo o mundo. Em 2006, o Ruby atingiu aceitação massiva, com a formação de grupos de usuários em todas as principais cidades do mundo e com as conferências sobre Ruby com lotação esgotada.

A Ruby-Talk, a principal lista de e-mails para a discussão sobre a linguagem Ruby, atingiu uma média de 200 mensagens diárias em 2006. Recentemente a média caiu, já que o tamanho da comunidade distribuiu as discussões de uma lista central em muitos grupos menores.

Ruby está posicionado entre no top 10 da maioria dos índices que medem o crescimento da popularidade de linguagens de programação pelo mundo todo (tais como o índice TIOBE). Muito deste crescimento é atribuído à popularidade de softwares escritos em Ruby, em particular o framework de desenvolvimento web Ruby on Rails.

O Ruby também é totalmente livre. Não somente livre de custos, mas também livre para utilizar, copiar, modificar e distribuir.

2. Lexemas do Ruby

Utilizamos, em Ruby, as seguintes regras para criação do identificador:

1. não pode ser uma palavra-reservada (palavra-chave);
2. não pode ser true nem false - literais que representam os tipos lógicos (booleanos);
3. não pode ser null - literal que representa o tipo nulo;
4. não pode conter espaços em brancos ou outros caracteres de formatação;
5. deve ser iniciado por uma letra minúscula ou um underscore;
6. deve ser iniciado com um cifrão caso seja de escopo global;

7. deve ser a combinação de uma ou mais letras e dígitos UNICODE-16. Por exemplo, no alfabeto latino, teríamos:

- letras de A a Z (de \u0041 a \u005a);
- letras de a a z (de \u0061 a \u007a);
- sublinha _ (\u005f);
- cifrão $ (\u0024);
- dígitos de 0 a 9 (de \u0030 a \u0039).

2.1 Comentários

As instruções que não são executadas pelo compilador e interpretador são chamadas de Comentários. Durante a codificação, o uso adequado dos comentários torna a manutenção mais fácil e a localização de bugs facilmente.

Em Ruby existem dois tipos de comentários:

1. Comentários de uma linha:

É representado pelo sinal # . É usado para denotar um comentário de linha única em Ruby. São os comentários digitados mais fáceis. Quando precisamos de apenas uma linha de um comentário em Ruby, podemos usar os caracteres '#' antes do comentário.

2. Comentários multilinhas:

Se nosso comentário se estende em mais de uma linha, então podemos usar um comentário multilinha. Em Ruby, o comentário multilinha começou usando = begin e terminou usando a sintaxe = end.

2.2 Palavras reservadas

alias:

Serve como um apelido para um método já definido.

BEGIN:

É usado para definir um bloco de código que será executado antes mesmo que o método principal.

begin:

É utilizado para iniciar uma seção de código que lança uma exceção na aplicação.

break:

É utilizado para encerrar a execução de um laço.

case:

É utilizado como uma estrutura de decisão.

class:

É utilizado para definir uma classe.

def:

É utilizado para definir um método.

defined:

É utilizado para retornar informações sobre a definição de uma determinada constante, variável, método ou classe.

do:

Utilizado para iniciar uma estrutura de repetição.

else:

Utilizado como parte da estrutura de comandos condicionais na linguagem, normalmente depois de um "**if**" ou "**elsif**".

elsif:

Utilizado de maneira parecida com o "**else**" normalmente após um "**if**", essa palavra reservada faz parte da estrutura de comandos condicionais da linguagem.

END:

É utilizado para definir o bloco de comandos a ser executado, sendo um delimitador.

end:

É utilizado para definir o final das estruturas de controle, como do, if, etc.

ensure:

É utilizado para garantir que um determinado bloco de código seja executado, independentemente se for lançada uma exceção durante a execução do mesmo.

for:

É utilizado para definir uma estrutura de loop, sendo executada “n” vezes.

if:

É utilizado para delimitar um bloco de execuções baseado em uma condição.

in:

É utilizado para determinar se um determinado elemento pertence a um determinado conjunto ou range(definido no for, por exemplo).

module:

É utilizado para definir módulos, que são containers para métodos, classes e constantes.

next:

É utilizado para pular para próxima iteração em um laço como o for, por exemplo.

nil:

É utilizado para representar a ausência de valor.

redo:

É utilizado para realizar uma nova iteração do loop, voltando para o início do loop, sem levar em consideração algum possível flag que possa vir posteriormente.

rescue:

É um bloco de comandos designado a tratar uma determinada exceção.

retry:

É utilizado dentro de um rescue, para volta para o begin, ou seja, o inicio do bloco de comandos de uma determinada exceção.

return:

É utilizado para marcar o final da execução de um método, retornando um valor.

self:

É utilizado para referir-se ao objeto que está sendo utilizado naquele momento pra chamar um método de uma classe, por exemplo.

super:

É utilizado para chamar o método como o mesmo nome que está na classe pai.

then:

É utilizado logo após um if para separar o que deve ser executado da expressão lógica ou valor contido na condição.

undef:

É utilizado para remover a definição de um método, classe ou objeto.

unless:

É utilizado como uma negação de um if, por exemplo, se uma condição for verdadeira, o unless não é executado.

until:

É utilizado como um flag em um loop.

when:

Utilizado em conjunto com o case citado acima, serve como um validador.

while:

É utilizado para testar uma condição e enquanto ela for verdadeira, será executado o loop.

yield:

É utilizado para invocar um bloco de código passado para um método.

2.3 Literais reservados

and:

Operador lógico que retorna um valor booleano a partir de duas expressões, a partir de uma tabela verdade.

false:

É uma expressão lógica utilizada para representar a não verdade de uma determinada

expressão, por exemplo.

not:

É utilizado para reverter o valor booleano, por exemplo, se um valor for true, quando usado o not, ele passará a ser false.

or:

Operador lógico que retorna um valor lógico a partir de uma tabela verdade entre duas expressões, o or tem valor inverso ao and.

true:

É uma expressão lógica utilizada para representar a verossimilidade de uma determinada expressão.

2.4 Operadores e delimitadores

Operador em Ruby é um símbolo que é usado para executar operações. Por exemplo: +, -,*, / etc.

Existem muitos tipos de operadores em Ruby que são fornecidos abaixo:

- Operador aritmético,
- Operador shift,
- Operador relacional,
- Operador bit a bit (bitwise),
- Operador lógico,
- Operador ternário e
- Operador de atribuição.

Precedência de operadores em Ruby

Tipo de Operador	Categoria	Precedência
Aritmética	multiplicativo	* / % **
	aditivo	+ -
Mudança	mudança	>> / <<
Relacional	comparação	> < >= <=

	igualdade	== !=
Bit a bit	bit a bit AND	&
	bit a bit OR exclusivo	|
	bit a bit NOT exclusivo	!
Lógico	AND lógico	and / &&
	OR lógico	or / ||
Ternário	ternário	A==B ? A : B
Atribuição	atribuição	||=

2.5 Literais String

são sequências de caracteres delimitadas por aspas simples ou duplas.

Por exemplo: string = "Olá, mundo!"

puts “Olá, mundo!”

2.6 Literais Numéricos

2.6.1 Literais Integer

É possível representar números do tipo integer de três formas em Ruby: decimal (base 10), octal (base 8) e hexadecimal (base 16).

2.6.1.1 Literal Decimal

Os decimais (base 10) em Ruby, são representados apenas com números, sem prefixo ou sufixo. **Ex.:** idade = 62; // 62 é uma literal decimal

2.6.1.2 Literal Octal

Pode-se representar números inteiros em notação octal usando o prefixo 0 seguido por um número zero.

Ex.:

```
numero = 0o12

numero_decimal = numero.to_i(8) # retorna 10
```

2.6.1.3 Literal Hexadecimal

Hexadecimais, ou hex, são números construídos com 16 símbolos distintos, sendo que do 10 ao 15 são representados pelas letras do alfabeto: a, b, c, d, e, f. Respectivamente, ficando desta forma: 0 1 2 3 4 5 6 7 8 9 a b c d e f.

Ex.:

```
numero = 0xA

numero_decimal = numero.to_i(16) # retorna 10
```

2.6.4 Literais de ponto flutuante

Os números de ponto flutuante são representados usando o tipo de dados "Float". Pode-se atribuir um número de ponto flutuante a uma variável, como mostrado abaixo:

Ex.: numero = 3.14

2.6.5 Literais Booleanas

Os valores booleanos são representados pelos valores "true" e "false". Eles são usados para representar valores lógicos, como verdadeiro ou falso.

Ex.:

```
verdadeiro = true
```

2.6.6 Literais Char:

Um caractere é representado por uma string de tamanho 1. Você pode atribuir um caractere a uma variável.

Ex.: letra = “A”

Além disso, pode-se tratar caracteres como strings, o que significa que você pode realizar operações comuns em strings, como concatenação, repetição, etc. com caracteres.

Ex.:

```
letra = "A"

novo_texto = letra * 3 # retorna "AAA"
```

Referências:

SOBRE o Ruby. Ruby-lang, 2003. Disponível em: https://www.ruby-lang.org/pt/about/. Acesso em: 5 de fevereiro de 2023.

COMENTÁRIOS em Ruby. Acervo Lima. Disponível em: https://acervolima.com/comentarios-em-ruby/. Acesso em: 6 de fevereiro de 2023.