Rozwiązania szkieletowe w tworzeniu aplikacji www

Tomasz Łukaszuk

Katedra Oprogramowania
Wydział Informatyki
Politechnika Białostocka

Temat wykładu:

Język programowania Ruby

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
NCBR Narodowe Centrum Badań i Rozwoju

Materiały przygotowane w ramach projektu "PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji" (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

Język programowania Ruby: Agenda

1. Wstęp
2. Podstawy
3. Przykłady

Wprowadzenie

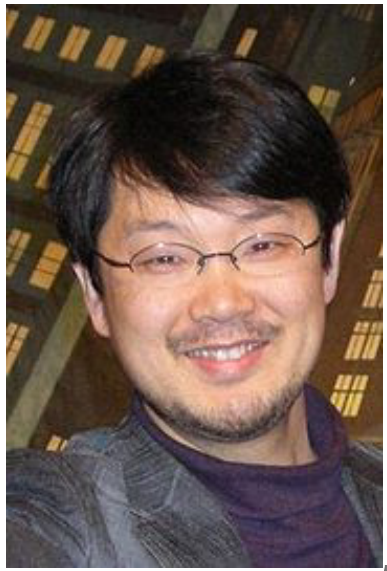
Ruby

- interpretowany, w pełni obiektowy i dynamicznie typowany język programowania
- bazuje na językach Perl, Smalltalk, Eiffel, Ada, i Lisp
- standardowa implementacja napisana jest w języku C, jako jednoprzebiegowy interpretowany język
- nie posiada jednoznacznej specyfikacji, implementacje języka Ruby: YARV, JRuby, Rubinius, IronRuby, MacRuby and HotRuby, każda stosuje inne podejście

Historia

Yukihiro Matsumoto

"I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language"



Historia

- **24 luty 1993** - rozpoczęcie prac nad językiem Ruby przez Yukihiro Matsumoto

- **21 grudzień 1995** - pierwsze wydanie 0.95
- **25 grudzień 1996** - Ruby version 1.0
- **wrzesień 2000** - pierwsza książka o języku Ruby po angielsku ”Programming Ruby”
- **dzisiaj** - 3.4.2 / 14 lutego 2025

Cechy szczególne

- prosta składnia
- automatyczne odśmiecanie pamięci
- iteratory
- przeciążanie operatorów
- obsługa wyjątków
- wyrażenia regularne wbudowane w składnię
- liczby całkowite o dowolnych rozmiarach
- dodawanie metod do instancji klasy
- bloki i lambdy (closures) - wygodne przekazywanie funkcji jako parametrów
- "Duck typing" - rozpoznawanie typów na podstawie ich zachowania, a nie deklaracji
- moduły - rodzaj wielodziedziczenia
- możliwość zmiany praktycznie wszystkiego

Obiektowy język programowania

- Wszystko jest obiektem ... nawet `nil`, nawet klasa (instancja klasy `Class`)
- `Object` jest domyślną superklasą
- Wszystko jest wyrażeniem i zwraca wartość
- Wszystko poza `nil` i `false` jest wartościowane jako prawda

Zmienne i stałe

- zmienne: zaczynają się od małej litery lub znaku _, słowa oddzielane przez _
  `my_variable, _var`
- stałe: zaczynają sie wielką literą, najlepiej całość wielkimi literami
  `MY_CONSTANT`
- zmienne globalne: zaczynają się od znaku `$`
  `$global_variable`
- symbole: zaczynają się od znaku `:`
  `:my_symbol`

```
1  f1,f2 = :ruby,:ruby
2  f1.object_id
3  f2.object_id #the same object
4
5  f1,f2 = "ruby","ruby"
6  f1.object_id
7  f2.object_id #two different objects
```

Komentarze

- Jednoliniowy komentarz

```
1  #one-line comment
```

- Wieloliniowy komentarz

```
1  =begin
2    multiline comment
3    multiline comment
4    multiline comment
5  =end
```

Operatory

- arytmetyczne: `+ - * / % **`
- przypisania: `= += -= *= /= %= **= |= &= >>= <<= ||= &&=`
- porównania: `== .eql? .equal? === != < > >= <= <=> =~ !~`
- logiczne: `&& || ! and or not`
- bitowe: `~ | & ^ << >>`
- inne: `[] []= .. ... ! not`

```
1  5 == 5.0      # => true
2  5.eql? 5.0    # => false, (the same type and value)
3  5.equal? 5.0  # => false, (the same object_id)
4  5.equal? 5    # => true, (for fixnums and symbols)
5
6  a = false or 5 # => 5, a == false
7  a = false || 5 # => 5, a == 5
8
9  a, b = b, a
10 a, b, c = get_something()
```

Instrukcje warunkowe

- `if ... [then|:] ... [elsif ...] [else ...] end`
- `... if ...`
- `unless = if not`
- `... ? ... : ...`
- `case ... when ... [else ...] end`

```
1  x = if a > 100
2      5000
3  elsif a > 5
4      5
5  else
6      1
7  end
8
9  puts "Hello" if a > 100
```

```
1  x = case x
2      when 0...5
3          1
4      when 5..100
5          50
6      else
7          10**9
8      end
```

Pętle

- `while ... [do] ... end`
- `... while ...`
- `until = while not`
- `for ... in ... [do] ... end`
- `break, next, redo, retry`
- `times, upto, downto`

```
1  while a > 10
2      a /= 3
3  end
4
5  puts "Iteration #{i+=1}" while i < 10
6
7  for i in 1..8
8      puts i
9  end
```

Funkcje (lub metody)

- wywołanie funkcji:
  `my_function` lub `my_function()`
- wywołanie z argumentami:
  `my_function a, b, c` lub `my_function(a, b, c)`
- wywołanie z wykorzystaniem zwracanej wartości:
  `a = my_function(a, b, c);puts my_function(a, b, c)`
- definiowanie

```
1  def my_function(a, b, c)
2      do_something
3      statement_return_sth
4  end
```

- domyślne argumenty:
  `def my_function(a, b = true) ...`
- lista argumentów:
  `def my_function(a, *other_args) ...`

Funkcje (lub metody)

- Bang methods
  - kończą się na `!`
  - potencjalnie niebezpieczne
  - modyfikują obiekty
  - mogą posiadać niemodyfikujące odpowiedniki tworzące nowe obiekty
  - np. `sort!`, `upcase!`, `reverse!`
- Asking methods
  - kończą się na `?`
  - zwykle zwracają `true` lub `false`
  - np. `empty?`, `include?`, `nil?`

Klasy

- definiowanie `class ... end`
- nazwa zaczyna się od wielkiej litery - CamelCase
- metoda `initialize` - konstruktor
- metoda `inspect` - <anObject:0x83678>
- metoda `to_s`
- instance variables - zaczynają się od `@`, np. `@inst_variable`
- class variables - begins with `@@`, e.g. `@@class_variable`
- accessors - `attr_reader`, `attr_writer`, `attr_accessor`
- instance method - `def method_name ... end`
- class method - `def self.method_name ... end`
- poziom dostępu do metod: public, protected, private
- metoda `method_missing`

Klasy

```ruby
class MyClass
    attr_reader :value
    def initialize(value)
        @value = value
    end
    private
    def introduce
        "My value is #{@value}"
    end
    public
    def self.public_method(arg)
        ...
    end
    def inspect
        "My id is #{object_id}"
    end
    def to_s
        "My type is #{self.class}"
    end
    def method_missing(method_id)
        puts "No method #{method_id}!"
    end
end
```

```ruby
a = MyClass.new(3)

puts a.introduce
# No method introduce

p a
# My id is 54765890

puts a
# My type is MyClass

puts a.b
# No method b
```

Klasy - dziedziczenie

- tylko jedno-bazowe
- Object - root
- `self`
- `super`

```
1  class MyClass
2      def introduce_yourself
3          puts "My name is " + self.class.to_s
4      end
5  end
6
7  class YourClass < MyClass
8      def introduce_yourself
9          super
10         puts "Something from YourClass"
11     end
12     def to_s
13         "YourClass"
14     end
15 end
```

Bloki

- `do ... end`
- `{ ... }`
- mogą być przekazywane do funkcji (jak obiekty)
- `yield`

```
1  10.times do |i|
2     puts i
3  end
4
5  10.times {|i| puts i}
6
7  def give_me_something
8     sth = Random.new.rand.to_s
9     yield(sth)
10 end
11 give_me_something { |x| puts "I've got #{x}" }
```

Wbudowane typy danych

- numbers: Integer, Fixnum, Bignum, Float, Rational
  ```
  7, 1245, -45.0
  ```
- strings
  ```
  "string example", 'other example'
  ```
- ranges
  ```
  (4..34)
  ```
- arrays
  ```
  [3, 6, "text", ['a', 78]]
  ```
- hashes
  ```
  { :water => 'wet', :fire => 'hot', :iron =>
  'cold' }
  ```
- regular expressions
  ```
  /regex*/
  ```

Hello world

Listing 1: hello.rb

```
1  #!/usr/bin/ruby
2  print "Hello World\n"
```

Listing 2: run in a Ruby shell irb

```
1  puts "Hello World!"
```

Some basic Ruby code

```ruby
# Everything, including a literal, is an object, so this works:

-199.abs
# 199

"ruby is cool".length
# 12

"Your mother is nice.".index("u")
# 2

"Nice Day Isn't It?".downcase.split("").uniq.sort.join
# " 'acdeinsty"
```

Terminal IO

```ruby
#!/usr/bin/ruby
print "This is the first half of Line 1. "
print "This is the second half.", "\n"
puts "This is line 2, no newline necessary."

printf "There were %7d people at the %s.\n", 439, "Auditorium"

print "Name please=>"
name = gets
print "Your name is ", name, "\n"
```

```
This is the first half of Line 1. This is the second half.
This is line 2, no newline necessary.

There were     439 people at the Auditorium.

Name please=>Jon Green
Your name is Jon Green
```

Strings

```
1  a = "\nThis is a double quoted string\n"
2  a = %{\nThis is a double quoted string\n}
3  a = %Q{\nThis is a double quoted string\n}
4  a = <<BLOCK
5
6  This is a multi-line double quoted string
7  BLOCK
8  a = %/\nThis is a double quoted string\n/
9
10 a = 'This is a single quoted string'
11 a = %q{This is a single quoted string}
```

Strings

string assignment and concatination

```
1  #!/usr/bin/ruby
2  myname = "Jon Green"
3  myname_copy = myname
4  print "myname      = ", myname, "\n"
5  print "myname_copy = ", myname_copy, "\n"
6  print "\n=========================\n"
7  myname << "-Red"
8  print "myname      = ", myname, "\n"
9  print "myname_copy = ", myname_copy, "\n"
```

the double less than sign (<<) is a Ruby String overload for concatination

```
1  myname      = Jon Green
2  myname_copy = Jon Green
3
4  =========================
5  myname      = Jon Green-Red
6  myname_copy = Jon Green-Red
```

Strings

the `String.new()` method

```
1  #!/usr/bin/ruby
2  myname = "Jon Green"
3  myname_copy = String.new(myname)          # <--------------
4  print "myname      = ", myname, "\n"
5  print "myname_copy = ", myname_copy, "\n"
6  print "\n=========================\n"
7  myname << "-Red"
8  print "myname      = ", myname, "\n"
9  print "myname_copy = ", myname_copy, "\n"
```

```
1  myname      = Jon Green
2  myname_copy = Jon Green
3
4  =========================
5  myname      = Jon Green-Red
6  myname_copy = Jon Green
```

Strings

the Ruby String class works like an array of characters

```
1  #!/usr/bin/ruby
2  myname = "Jon was here"
3  print myname[6, 3], "\n"
4  myname[6, 3] = "is"
5  print myname, "\n"
```

```
1  was
2  Jon is here
```

Strings

- the addition (+) sign means to add strings together (strings concatenation)
- the multiplication (*) sign means string together multiple copies
- the % method works like the `sprintf()` command in C
- strings comparision with `<=>`

```
1  mystring = "Jon" + " " + "was" + " " + "here"
2  print mystring, "\n"
3  mystring = "Cool " * 3
4  print mystring, "\n"
5  mystring = "There are %6d people in %s" % [1500, "the Ballroom"]
6  print mystring, "\n"
7  print "frank" <=> "frank", "\n"
8  print "frank" <=> "fred", "\n"
9  print "frank" <=> "FRANK", "\n"
```

```
1  Jon was here
2  Cool Cool Cool
3  There are   1500 people in the Ballroom
4  0
5  -1
6  1
```

Loops

The elipses (...) indicate the range through which to loop. The `for` is terminated by an `end`. You don't need braces for a loop.

```
1  for ss in 1...5
2      print ss, " hello\n";
3  end
```

```
1  1 hello
2  2 hello
3  3 hello
4  4 hello
```

The `1...5` means 1 TO BUT NOT INCLUDING 5
The `1..5` means 1 through 5

Loops

```
1  presidents = ["Ford","Carter","Reagan","Bush1","Clinton","Bush2"]
2  for ss in 0...presidents.length
3      print ss, ": ", presidents[ss], "\n";
4  end
```

```
1  0: Ford
2  1: Carter
3  2: Reagan
4  3: Bush1
5  4: Clinton
6  5: Bush2
```

Backwards iteration doesn't work in Ruby – it must iterate up.

```
1  presidents = ["Ford","Carter","Reagan","Bush1","Clinton","Bush2"]
2  for ss in 0...presidents.length
3      print ss, ": ", presidents[-ss-1], "\n";
4  end
```

array[-1] is the last item, array[-2] is the second to last, etc.

Loops

`while` loops

```
1  #!/usr/bin/ruby
2  ss = 4
3  while ss > 0
4      puts ss
5      ss -= 1
6  end
7  puts "======================"
8  while ss < 5
9      puts ss
10     ss += 1
11     break if ss > 2
12 end
```

```
1  4
2  3
3  2
4  1
5  ======================
6  0
7  1
8  2
```

Iterators and blocks

Another way to loop through an array is to use an iterator (`each`) and a block (`{|prez| puts prez}`).

```
1  presidents = ["Ford","Carter","Reagan","Bush1","Clinton","Bush2"]
2  presidents.each {|prez| puts prez}
3
4  (1..5).each do |i|
5      puts i
6  end
```

```
1  Ford
2  Carter
3  Reagan
4  Bush1
5  Clinton
6  Bush2
7  1
8  2
9  3
10 4
11 5
```

Iterators and blocks

The examples of other ruby iterators:

```ruby
[1, 2, 3].collect { |element| element + 1}        #=> [2, 3, 4]

a = [1, 2, 3]                                     #=> [1, 2, 3]
a.collect! { |element| element + 1 }              #=> [2, 3, 4]
a                                                 #=> [2, 3, 4]

[1, 2, 3, 4, 5, 6].delete_if { |i| i%2 == 0 }     # => [1, 3, 5]

(36..100).detect { |i| i%7 == 0 }                 #=> 42

9.downto(0) { |i| print i }                       #=> 9876543210

[3, 6, -5].each_index { |i| print i.to_s + " " } #=> 0 1 2

(0..30).find_all { |i| i%9 == 0 }                 #=> [0, 9, 18, 27]

(1..6).partition { |i| i%2 == 0 }        #=> [[2, 4, 6], [1, 3, 5]]

5.times { |i| print "#{i} "}                       #=> 0 1 2 3 4

1.upto(3) { |i| print i }                         #=> 123
```

Branching

```
1  democrats = ["Carter", "Clinton"]
2  republicans = ["Ford", "Reagan", "Bush1", "Bush2"]
3  party = ARGV[0]
4  if party == nil
5      print "Argument must be \"democrats\" or \"republicans\""
6  elsif party == "democrats"
7      democrats.each { |i| print i, " "}
8  elsif party == "republicans"
9      republicans.each { |i| print i, " "}
10 else
11     print "All presidents were either Democrats or Republicans"
12 end
13 #-----------------------------------------------------------
14 if party != nil
15     democrats.each { |i| print i, " "} if party == "democrats"
16     republicans.each { |i| print i, " "} if party == "republicans"
17     print "All presidents were either Democrats or Republicans"\
18         if (party != "democrats" && party != "republicans")
19 end
```

The `if` keyword must be on the same line as the action

Only a single action can precede the `if` keyword

Collections

Constructing and using an array

```
1  a = [1, 'hi', 3.14, 1, 2, [4, 5]]
2
3  a[2]                  # 3.14
4  a.[](2)               # 3.14
5  a.reverse             # [[4, 5], 2, 1, 3.14, "hi", 1]
6  a.flatten.uniq        # [1, "hi", 3.14, 2, 4, 5]
7  a.pop                 # [4, 5]
8  a                     # [1, "hi", 3.14, 1, 2]
9  a.push('seven')       # [1, "hi", 3.14, 1, 2, "seven"]
10 a.shift(2)            # [1, "hi"]
11 a                     # [3.14, 1, 2, "seven"]
12 a.unshift(4, 'abc')   # [4, "abc", 3.14, 1, 2, "seven"]
13 a[1..3]               # ["abc", 3.14, 1]
14 a[2,4]                # [3.14, 1, 2, "seven"]
15 a[2,3] = []           # []
16 a                     # [4, "abc", "seven"]
```

Collections

Constructing and using an associative array (called hashes in Ruby)

```ruby
hash = { :water => 'wet', :fire => 'hot', :iron => 'cold' }

hash[:fire]           # "hot"

hash.each do |key, value|
     puts "#{key} is #{value}"
end
# water is wet
# fire is hot
# iron is cold

hash.select {|key,value| value!='hot'}
# [[:water, "wet"], [:iron, "cold"]]
hash.delete :water
# {:fire => "hot", :iron => "cold"}
hash.delete_if {|key,value| value=='hot'}
# {:iron => "cold"}
```

Regular Expressions

```ruby
string1 = "Ruby regular expressions"
print "yes" if string1 =~ /r.*l/   # yes
print "yes" if string1 !~ /e.*z/   # yes
print "yes" if string1 =~ /^[A-Z]/ # yes

string1 = "I will drill for a well in walla walla washington."
regex = Regexp.new(/w.ll/)
matchdata = regex.match(string1)
while matchdata != nil
    puts matchdata[0]
    string1 = matchdata.post_match
    matchdata = regex.match(string1)
end
# will
# well
# wall
# wall

string1 = "I will drill for a well in walla walla washington."
string1.gsub!(/(w.ll)/){$1.upcase}
# I WILL drill for a WELL in WALLa WALLa washington.
```

Classes

```ruby
class Person
      attr_reader :name, :age
      def initialize(name, age)
          @name, @age = name, age
      end
      def <=>(person) # Comparison operator for sorting
          @age <=> person.age
      end
      def to_s
          "#@name (#@age)"
      end
end

group = [
    Person.new("Bob", 33),
    Person.new("Chris", 16),
    Person.new("Ash", 23)
]
puts group.sort.reverse
```

```
Bob (33)
Ash (23)
Chris (16)
```

Classes

Inheritance

```
1  class Student < Person
2      attr_accessor :index
3        def initialize(name, age, index)
4            super(name, age)
5            @index = index
6        end
7        def to_s
8            "Student: #@name (#@age, #@index)"
9        end
10 end
11
12 s = Student.new('Frank', 21, 758745)
```

Classes

Open Classes

In Ruby, classes are never closed: you can always add methods to an existing class.

```ruby
# re-open Ruby's Time class
class Time
    def yesterday
        self - 86400
    end
end

today = Time.now # => Thu Aug 14 16:51:50 +1200 2008
yesterday = today.yesterday # => Wed Aug 13 16:51:50 +1200 2008
```

Unconstant number of arguments passed to the function

The last parameter can be started from the mark *, which means that any number of parameters will be transformed into an array.

```
1  def reverse_array(*b)
2      if b.size == 1
3          b
4      else
5          reverse_array(*b[1..-1])+[b[0]]
6      end
7  end
8
9  print reverse_array("!\n","ld","wor",", ","llo","He")
10 # Hello, world!
```