Język Ruby

wprowadzenie

Radosław Bułat
http://radarek.jogger.pl
29 luty 2008

O czym prezentacja?

- wstęp do Rubiego
  - to nie jest tutorial
  - raczej przedstawienie możliwości oraz ekosystemu Rubiego
- próba zainteresowania Was Rubym
  - `Hej, może warto to „obczaić“?`
  - pokazanie powodów, dla których warto zainteresować się nim

Może chociaż jedna osoba zrobi tak...


FuniaSte.net

O czym nie jest prezentacja?

- porównanie języków
- wyższości jednego języka nad drugim
  - flame war'om – mówimy stanowcze NIE :-)
- Ruby on Rails
  - Ruby to nie Ruby on Rails

Język Ruby

Cechy Rubiego

- skryptowy
- interpretowany
- dynamiczny
- w pełni obiektowy
- automatyczne odśmiecanie (Garbage Collector)
- bardzo wysokiego poziomu (VHLL)
- zorientowany na umysł człowieka a nie maszynę
- open source
- prosty w pisaniu, prosty w czytaniu
- „fun“

Cechy Rubiego

- prosta składnia, podobna do innych języków
- wiele elementów zaczerpniętych z takich języków jak Smalltalk, Perl, Lisp, Python i innych
- obsługa dowolnie dużych liczb
- wbudowane regexpy
- obsługa wyjątków
- elastyczność, dynamizm, rozszerzalność, ekspresyjność

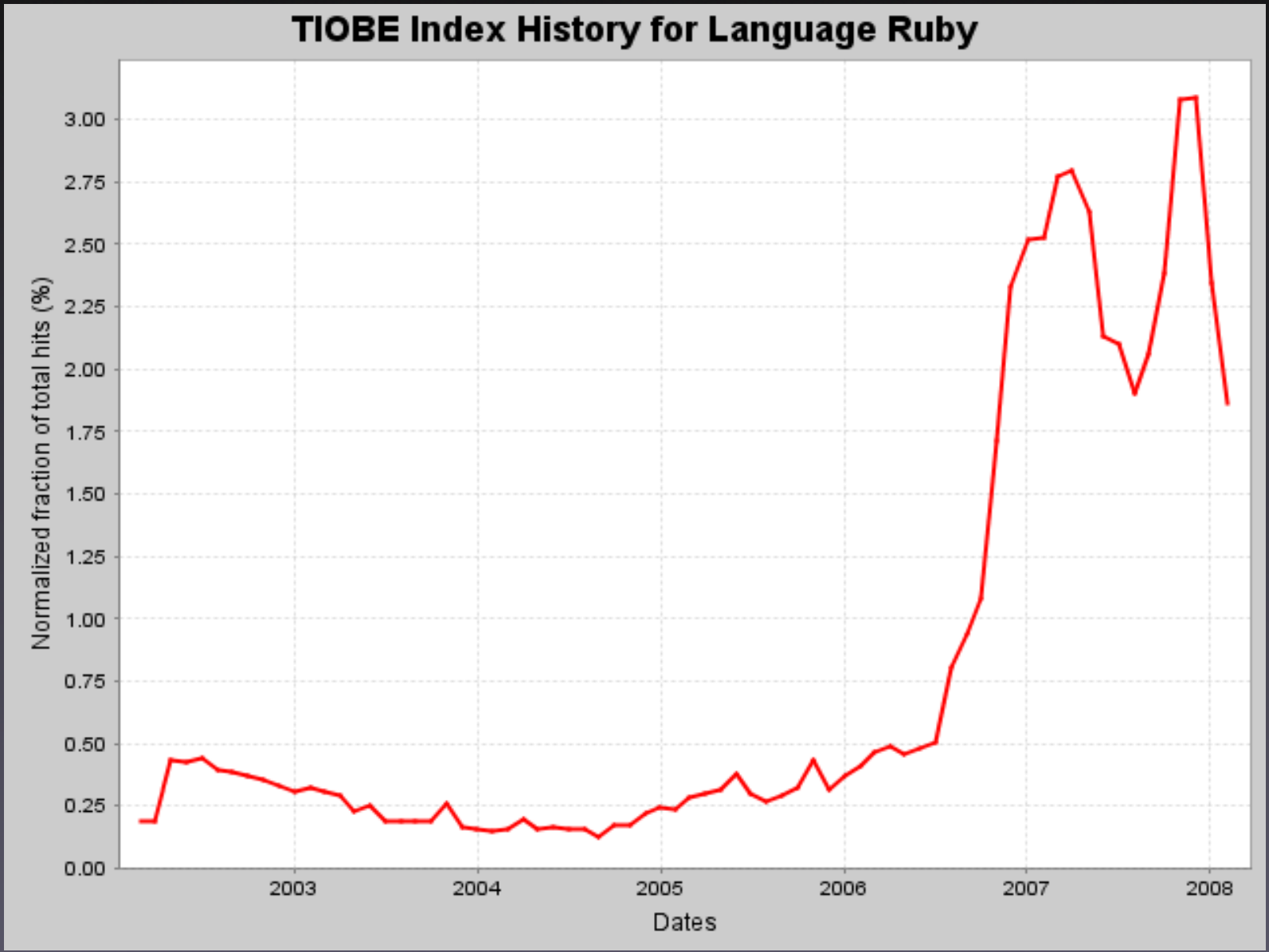
Twórca języka
Yukihiro „Matz“ Matsumoto

http://flickr.com/photos/strangecontext/1850262566/

Historia

- 1993 – Yukihiro Matsumoto (Matz) rozpoczyna prace nad językiem gdyż ówczesne popularne języki nie spełniały jego wymagań. Ruby z założenia ma być połączeniem najlepszych cech takich języków Perl, Python, Lisp, Smalltalk oraz innych
- 1995 – wydanie pierwszej wersja języka, zdobywa popularność w Japonii
- 2004 – pierwsza wersja i od razu ogromny sukces Ruby on Rails, który zostaje killer application
- 2005 – dwie najpopularniejsze książki na temat Ruby i Ruby on Rails w sklepie Amazon były najlepiej sprzedawanymi pozycjami w kategorii Programowanie
- 2006 – Ruby językiem roku według TIOBE*

* http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html



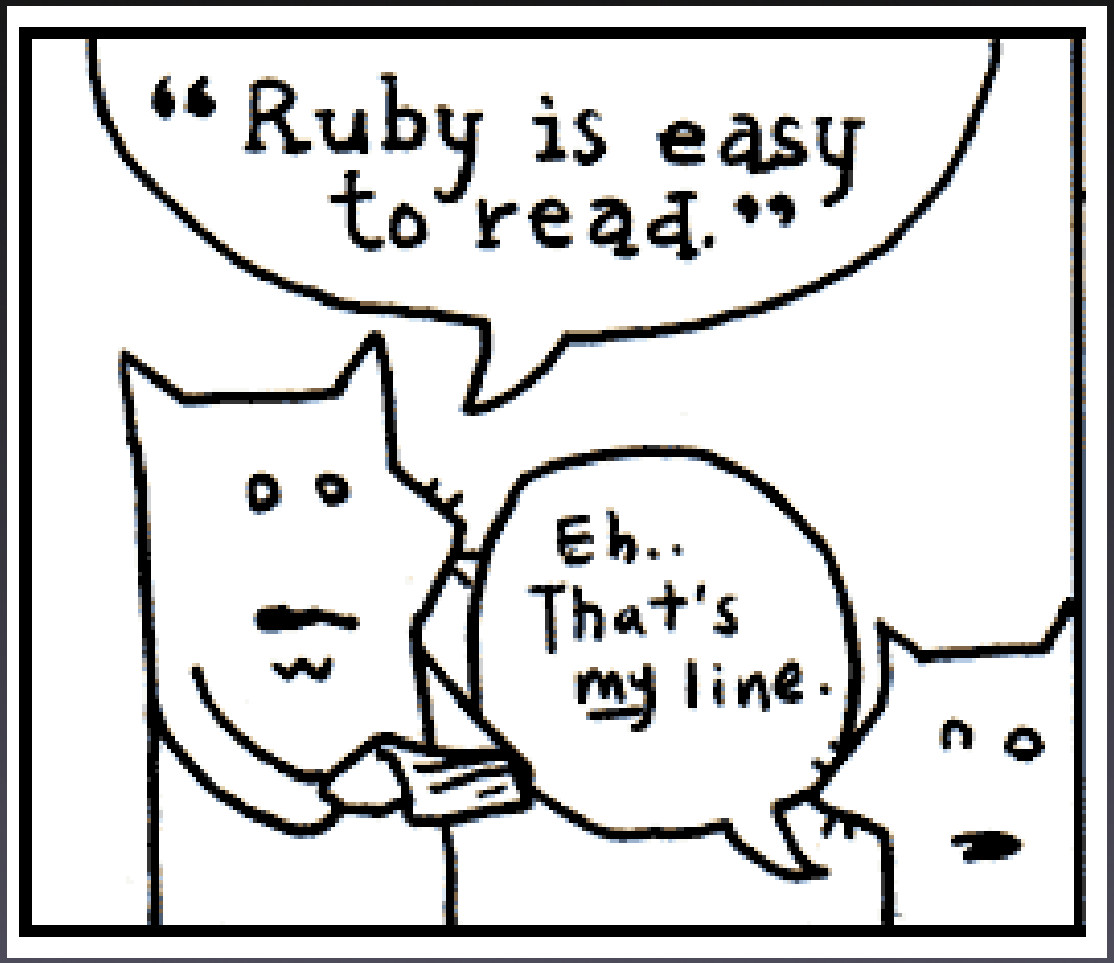
Obecnie (luty 2008) 11 miejsce w rankingu TIOBE

W czym jest dobry?

- jednolinijkowce (ruby –e ”puts ‘hello world’”)
- skrypty „quick and dirty” (napisz, uruchom, skasuj)
- skrypty „administratorskie“
- przetwarzanie tekstu
- prototypowanie
- aplikacje webowe
- aplikacje bazodanowe
- aplikacje sieciowe i rozproszone
- DSL (Domain Specific Language)

Myśli przewodnie

- Principle of Least Surprise
- język jest dla ludzi, nie dla komputerów
- naturalność, swoboda
- proste rzeczy powinny być proste, skomplikowane powinny być możliwe



Przegląd składni

Składnia

```ruby
puts "Hello World!"
x = 2 ** 100
puts "x = #{x}" #=> x = 1267650600228229401496703205376

a = [1, 2, 3, 4, 5]
if a.include?(3)
  puts "ok, 3"
else
  puts "co jest grane?"
end #=> "ok, 3"

if m = "--123--".match(/(\d+)/)
  puts m[0].to_i * 2 #=> 246
end

reversed_ip = "127.0.0.1".split(".").reverse.join(".")
puts reversed_ip #=> 1.0.0.127
```

Składnia

```ruby
i = 0
while i < 10
  i += 1
end

begin
  puts "przed wyjątkiem"
  raise "coś poszło źle"
  puts "za późno"
rescue => e
  puts "wyjątek: #{e.message}"
ensure
  puts "koniecznie!"
end
```

Składnia

```ruby
# przetwarzanie pliku
file = File.open(__FILE__)
file.each_line do |line|
  next if line =~ /^#/
  puts line
end
file.close

def fib(n)
  if n == 0 || n == 1
    return 1
  else
    return n * fib(n - 1)
  end
end

1.upto(10) {|i| puts fib(i) }
```

Wszystko jest obiektem

Wszystko!*

* czyli to co możesz przypisać do zmiennej

Typy „proste“ także są obiektami

```ruby
0.zero?             # => true
1.zero?             # => false
1.abs               # => 1
-1.abs              # => 1
1.methods           # => lista metod dla obiektu 1
2.+(3)              # => 5 (to samo co 2+3)
10.class            # => Fixnum
(10**100).class     # => Bignum
true.class          # => TrueClass
false.class         # => FalseClass
nil.class           # => NilClass
(1..10).class       # => Range
/foo/.class         # => Regexp
```

Klasy

```ruby
require "open-uri"

class WordFinder
  attr_accessor :url

  def initialize(url)
    @url = url
    @content = open(url).read
  end

  def how_many?(word)
    return @content.scan(word).size
  end
end

wf = WordFinder.new("http://google.pl")
puts("google: %s" % wf.how_many?("google")) # => 14
puts("polski: %s" % wf.how_many?("polski")) # => 1
```

Otwarte klasy

```ruby
class String
  def encrypt
    tr "a-z", "b-za"
  end
end

puts "cat"
puts "cat".encrypt
# cat
# dbu
```

Bloki, domknięcia

- do wywołania metody można dołączyć stowarzyszony z nią blok
- blok to kawałek kodu, ale to wywołana metoda decyduje kiedy i jak wywołać blok
- bloki to podstawowym element Rubiego i stosowane są do:
  - iteracji
  - callbacków
  - pozyskiwania zasobów
  - wątków
  - tranzakcji

Przykłady bloków

```ruby
a = [1, 2, 3]
a.each do |n|
  puts n
end

10.times { puts "Ruby!" }
2.upto(5) {|i| puts i }

a = [5, 6, 13, 4, 5, 2, 2, 1, 5]
a.uniq.sort.map! {|e| e * 2}

["ccc", "a", "bb"].sort_by {|s| s.length }
```

Przykłady bloków

```ruby
button.on_click do
  puts "button pressed..."
end

File.open("filename.txt", "w") do |f|
  f << "message"
end # plik został zamknięty

Thread.new do
  puts "nowy wątek wystartował"
end

ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

Domain Specific Language (DSL)

Domain Specific Language – język programowania zaprojektowany, w przeciwieństwie do języków takich jak C lub Java, do ściśle określonego zadania.

Przykładem takich języków są wyrażenia regularne, SQL, make. Język dziedzinowy skupia się na robieniu jednego rodzaju zadań dobrze. Np. narzędzie make służy do definiowania celów oraz zależności między nimi.

Ruby i DSL

- elastyczność składni Rubiego ułatwia tworzenie takich języków
- możesz tworzyć mini języki i wykorzystywać bezpośrednio w kodzie
- przykłady:
  - rake – Ruby make
  - Ruby on Rails
  - RSpec
- dobrze napisany kod Rubiego z natury jest bliski DSL

Przykład DSL - Rake

```
1  file "main.o" => "main.c" do
2    sh "cc -c -o main.o main.c"
3  end
4
5  desc "build hello executable"
6  file "hello" => "main.o" do
7    sh "cc -o hello main.o"
8  end
9
10 desc "cleans everything"
11 task :clean do
12   rm_f ["main.o", "hello"]
13 end
14
15 desc "runs hello"
16 task :run => :hello do
17   sh "./hello"
18 end
```

Przykład DSL - Rake

```
$ rake -T
rake clean  # cleans everything
rake hello  # build hello executable
rake run    # runs hello
$ rake hello
cc -c -o main.o main.c
cc -o hello main.o
$ rake run
./hello
Hello!
```

Przykład DSL - ActiveRecord

```ruby
class User < ActiveRecord::Base
  has_many    :projects
  belongs_to  :user_group

  validates_presence_of     :login, :name, :password, :email, :age
  validates_uniquenes_of    :login

  before_create   :notify_admin
  after_destroy   :say_goodbye

private
  def notify_admin
  end

  def say_goodbye
  end
end
```

Metaprogramowanie

- technika która umożliwia programom tworzenie lub modyfikację kodu innych programów (lub ich samych)
- odpowiednio wykorzystana daje ogromne możliwości
- „zaprogramuj Twój program“
- w Rubym to bułka z masłem!

Metaprogramowanie - przykład

```ruby
module Deprecation
  def deprecated(name)
    self.class_eval do
      alias_method "old_#{name}", name

      define_method(name) do
        puts "(warn) method #{name} is now deprecated"
        self.send("old_#{name}")
      end
    end
  end
end

class MyLib
  extend Deprecation
  def say_hi
    puts "Hi!"
  end

  deprecated "say_hi"
end

MyLib.new.say_hi
# (warn) method say_hi is now deprecated
# Hi!
```

Rspec - Behaviour Driven Development framework

```ruby
class Stack
  def initialize
    @s = []
  end

  def empty?
    return @s.size == 0
  end

  def size
    return @s.size
  end

  def push(e)
    @s.push(e)
  end

  def pop
    if empty?
      raise "empty stack"
    else
      return @s.pop
    end
  end
end
```

Rspec - Behaviour Driven Development framework

```ruby
require "stack"

context Stack do
  it "should be empty after create" do
    @stack = Stack.new
    @stack.should be_empty
  end

  it "should increase size after push" do
    @stack = Stack.new
    s = @stack.size
    @stack.push(1)
    @stack.size.should == s + 1
  end

  it "should raise error when pop on empty stack" do
    @stack = Stack.new
    lambda { @stack.pop }.should raise_error
  end

  it "should return top item when pop" do
    @stack = Stack.new
    @stack.push(1)
    @stack.pop.should == 1
  end
end
```

Rspec - Behaviour Driven Development framework

```
$ spec stack_spec.rb -f specdoc

Stack
- should be empty after create
- should increase size after push
- should raise error when pop on empty stack
- should return top item when pop

Finished in 0.021827 seconds

4 examples, 0 failures
```

Narzędzia

- irb (interactive ruby) – interaktywna konsola
- rake – ruby make
- rubygems – instalacja, dystrybucja bibliotek
- rdoc, ri – dokumentacja kodu źródłowego

IRB

```
>> 1 + 1
=> 2
>> 3 ** 30
=> 205891132094649
>> class Hello
>>   def initialize(name)
>>     @name = name
>>   end
>>   def say_hi
>>     puts "Hello #{@name}"
>>   end
>> end
=> nil
>> h = Hello.new("Perl Mongers")
=> #<Hello:0x2b49eca42070 @name="Perl Mongers">
>> h.say_hi
Hello Perl Mongers
=> nil
>>
```

Przyszłość Rubiego?

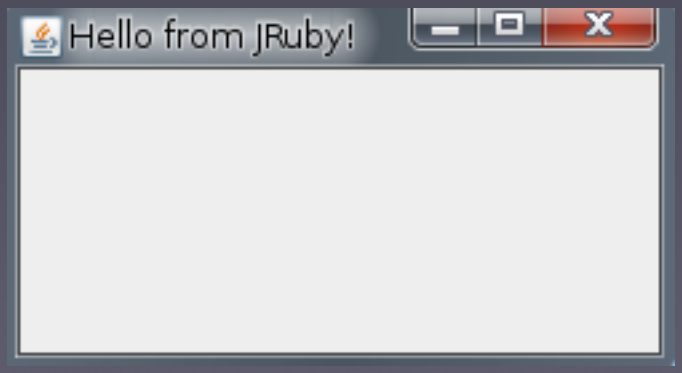
- wiele usprawnień w wersji 1.9
  - obsługa kodowań (np. utf8)
  - lepsza wydajność
- Rubinius – maszyna wirtualna wzorowana na Smalltalk
- JRuby (Sun)
- IronRuby (Microsoft)
  - Silverlight

JRuby

```ruby
require "java"
include Java

include_class "javax.swing.JFrame"

frame = JFrame.new
frame.set_size(250, 120)
frame.title = "Hello from JRuby!"
frame.set_default_close_operation(JFrame::EXIT_ON_CLOSE)
frame.visible = true
```



Dlaczego warto?

- przejrzysty, czytelny kod
- radość z programowania

„Wiesz, ja jestem maniakiem jeśli chodzi o piękno mojego kodu. Czasem robię przerwę i patrzę na niego. Bo jest super :D.“

Ruby czyni programistę szczęśliwym

Dziękuję!

Radosław Bułat
http://radarek.jogger.pl
29 luty 2008