Introduction to Ruby

박상길
다음커뮤니케이션

Ruby
A Programmer's Best Friend

What is Ruby?

- 1st release on December 1995
- Yukihiro Matsumoto("Matz"), Japan

What is Ruby?

- Ruby == Perl++
- 100% OO Language
- Dynamically typed language
- Open Source(GPL) with implementations on Java VM(JRuby), Windows, several Linux distributions ...
- Concise, Simple, Fun!

What is Ruby?

- A Scripting language
- An object-oriented language
- A dynamic language
- A Very High Level Language(VHLL)
- A human-oriented language
- An open-source project

Philosophy

- Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something."
  They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines.
  We are the masters. They are the slaves.
  - Yukihiro Matsumoto

Basic features

- Classes, Inheritance
- Threads
- Cross Platform
- Iterators and Closures
- Garbage Collection
- Exception Handling
- Object Oriented

Basic features

- Regular Expressions
- Variables are not typed
- Powerful string operations
- Operator Overloading
- Introspection, Reflection, Meta programming

to Ruby from Java

- Similarities
  - Garbage Collector
  - Objects are strongly typed
  - public, private and protected methods
  - Embedded doc tools

to Ruby from Java

- Differences
    - Don’ t need to compile code, just run it
    - No static type checking
    - No casting

to Ruby from PHP

- Similarities
  - Dynamically typed
  - eval
  - Fairly large standard library
  - Arrays and hashes work like expected

to Ruby from PHP

- Differences
  - Strong typing
  - Everything is an object
  - No abstract classes or interfaces
  - Almost everything is a method call

Influences

- Lisp
- Smalltalk
- Perl

Influences

- Lisp - everything is an expression
- Smalltalk - everything is an object
- Perl - everything should be possible

Hello, World

```
public class HelloWorld {
   public static void main(String[] args) {
      for(int i = 0; i< 10; i++) {
         System.out.println(i + “ times ...“);
      }
   }
}
```

Hello, World

```
10.times do |i|
  puts “#{i} times ...”
end
```

Some basic syntax

- # comment
- # multiple assignment:
  a, b = 1, 2
- # Interpolating values in strings:
  puts “The sum is #{a+b}”

Conventions

- CamelCaseClassNames
- @instance_variables
- @@class_variables
- $globals
- CONSTANTS
- :symbols

Loops

-
  ```
  for index in 1..9 do
      puts "Iteration #{index}"
  end
  ```

-
  ```
  1.upto(9) {|index| puts index}
  ```

-
  ```
  index = 1
  while index < 9 do
      puts index
      index += 1
  end
  ```

-
  ```
  loop { puts "Infinite loop." }
  ```

Control Expressions

- if a == true then puts “yes” end
- if a == true { puts “yes” }
- puts “yes” if a == true

-
  ```
  if a == true
      puts “ok, you got the point...”
  end
  ```

And more control expressions

- if elsif else end
- unless(same as "if not")
- while
- loop
- for
- case

Boolean operations

- !, ||, && - same as Java. Evaluation from left to right.
- not, and, or - evaluation from right to left

Implicit typing

- a = “some string”
  a.class
  =>” String”
- 1.class
  => “Fixnum”

Wacky Syntax

- You don't need ; to end lines, but you can.
- Parenthesis are optional. But not always
- There's no begin without an end. But things can end without a begin.
- {} or begin/end. Whatever.

No Interfaces!

- "Duck Typing" philosophy:

  If it walks like a duck,
  And talks like a duck,
  Then we can treat it like a duck.
  (who cares what it really is)

OOP in ruby

- Everything is an object - no wrappers as in Java
- Standalone functions are really methods of Object
- Code can be stored as objects
- Singletons are permitted
- Metaclasses
- Data hiding: public, private, protected

Inheritance

```
class Student < Person
    def initialize(name, number, id, major)
        @name, @phone = name, number
        @id, @major = id, major
    end
    def inspect
        super +   “ID=#@id Major=#@major”
    end
end
```

Garbage Collection

- No need for destructors
- No memory deallocation, etc.
- Currently "mark and sweep" technique
- Plans for generational GC

The Bignum class

- A Fixnum will transparently "roll over" into a Bignum - an arbitrary-precision integer

And no NullPointers, too

- Nil is an object
  puts nil.class
  => NilClass
- a = nil
  a.nil?
  => True

Classes, Structs and Modules

- A Class is an extensible definition or something
- A Module is the definition of a class. It cannot be instantiated, nor extended. But classes can include it, or extend it.
- A Struct is a temporary class made of something.

Attributes and Constructors

```
class Moto
    attr_accessor :owner

    def initialize(owner_name, purchase_date = Time::now)
        @owner = owner_name
        @purchase_date = purchase_date
    end
end

y = Moto.new( “John Doe” )
puts y.owner

puts y.instance_variables
puts y.purchase_date
```

Getters and Setters, be gone!

- Every attribute is automatically encapsulated, but overriding is easy

-
  ```
  class Person
      attr_reader :name # read only
      attr_accessor :phone # getter/setter
  end
  ```

/regexp/

- Same syntax as Perl
- require 'net/http'

```
conn = Net::HTTP.new('www.cheju.ac.kr', 8080)
resp, data = conn.get('/', nil)

if resp.message == "OK "
    data.scan(/href\s*=\s*\"*[^\">]*/i) { |x| puts x }
end
```

Catch that!

- begin
  ...
  rescue [error_type [=> variable]]
  ensure
  end
- Java
  try ... catch ... finally

Useful classes, strange syntaxes

- Hashes

```
hash = Hash.new
hash.store “key” , “value”
puts hash.get( “key” )
```

or

```
hash = {}
hash[ “key” ] = “value”
puts hash[ “key” ]
```

Useful classes, strange syntaxes

- Arrays

```
a = Array.new
a[4] = "a"
=> [nil, nil, nil, "a" ]

or

a = []
a << "a"
```

Useful classes, strange syntaxes

- Strings

  String.new or “”

  a = “some string”
  puts a[0..3]
  => “some”

:symbols

- A symbol is something that is not yet anything, but it is already there. Like a string, without the string part.
- Constants, keys on maps, string representations.

```
john =
    {:name =>  “John Doe” ,
    :car=>  “BMW” }
puts john[:name]
=>” Jone Doe”
```

Ah... Blocks

```
something = 10, something_else = 20
my_price = {
    if something == 20
        1
    elsif something_else == 10
        2
    else
        3
    end
}
puts my_price
=> 3
```

Did I say block assignment?

- nome, endereco, telefone = “John Doe” , “Gotham City” , “+1 123 456 789”
  puts nome
  => “John Doe”
- car, parts = “Peugeot” , [:engine, :wheels, :seats]
  puts parts.size
  => 3
  puts parts.class
  => array

Extending Ruby in C

- Every Ruby object is accessed as a VALUE(either an immediate value or a pointer)
- The only header file needed is ruby.h
- Various rb_* functions crrespond to Ruby operations(rb_ary_push, rb_define_var, and so on)
- C datatype wrapping is accomplished with Data_Wrap_Struct, Data_Make_Struct, and Data_Make_Struct

Libraries and Utilities

- HTTP, CGI, XML, and related libraries
- Network and distributed app libraries
- Ruby On Rails!
- TextMate!

Who’ s into Ruby

- Dave Thomas, Andy Hunt: authors of The Pragmatic Programmer
- Ron Jeffries, Chet Hendrickson: XP gurus and co-authors of XPI

Ruby’s Weaknesses

- Some external add-ons(libraries,tools, utilities) of the language are immature, incomplete, or missing
- Many things are still documented only in Japanese
- There are some “issues” with Windows platforms
- The Ruby Application Archive(RAA) is not nearly so comprehensive as the CPAN as yet
- User base is limited and expertise is rare
- Industry acceptance is limited as yet

Conclusion

- A very flexible and natural language
- Based on few, but powerful concepts: closures, blocks, messages
- Not a tyrannical language: bad programmers can produce really nasty code with little or no restrictions