Ruby's past, present, and future

Nov 8th, 2011

Shugo Maeda

Self introduction

- Shugo Maeda
- Twitter ID: @shugomaeda
- A Ruby committer
- The co-chairman of the Ruby Association
- A director of Network Applied Communication Laboratory Ltd.

Topics

- Ruby's past,
- present,
- and future

Ruby's past

The birth of Ruby

- Born on February 24th 1993
  - Just the name
- Father: Yukihiro Matsumoto, a.k.a. Matz
- Godfather: Keiju Ishitsuka

How was the name Ruby chosen?

```
keiju> Come to think of it, did you come up with the name
of the language?
matz> Hmm, if it's similar enough to shells, Tish.
matz> But I need a more cool name
...

keiju> ruby
keiju> Of course, it should be a jewel name
matz> Is it taken from ruby annotation?
matz> Why should it be a jewel name?
matz> Affected by Mitsubishi?
keiju> perl
matz> I see
```

Ruby 0.49

- The oldest version available
  - available at ftp.ruby-lang.org
- Released privately in July 1994

Hello world in Ruby 0.49

```
print("Hello world\n")
```

- Is it the same as Ruby 1.9?
  - Yes, and no
  - Parentheses are necessary in Ruby 0.49

```
# Not valid in Ruby 0.49
print "Hello world\n"
```

Hello world in Python

```
$ python2.6 -c 'print "hello"'
hello
$ python3.1 -c 'print "hello"'
  File "<string>", line 1
    print "hello"
                ^
SyntaxError: invalid syntax
$ python3.1 -c 'print("hello")'
hello
```

CoC

- ~~Convention over Configuration~~
- Convenience over Consistency
- In Python, the print statement was removed for consistency
- In Ruby, parenthesis omission was supported for convenience

Example code in Ruby 0.49

```
class greeting
  def !=(array)          # != can be overridden
    do array.each using x  # old syntax for blocks
      @say_hello(x)
    end
    fail("error")        # raises an exception
  end def                # optional keyword after end

  def @say_hello(x)      # private method
    print("Hello, ", x, "!\n")
  end def
end class
```

Example code in Ruby 0.49 (cont'd)

```
protect                          # begin
  g = greeting.new()
  g != ("Matz" :: "Larry")  # a cons cell
  print("success\n")
resque
  print("failed\n")
end protect
```

- **resque** is a typo by Matz
  - Not by me!

Does it look like Ruby?

- No!
- But it already had essential features:
  - Interpreter
  - Pure object oriented
  - Dynamically typed
  - Garbage collection
  - Blocks
  - ...

What was Ruby created for?

- For UNIX
- For easy object-oriented programming
- For fun

Ruby for UNIX

- Ruby was created for UNIX
- Developed on SONY NEWS-OS
- Easy scripting like Perl
- APIs tied closely to UNIX/C
  - Not self-contained in contrast to Smalltalk
  - Better UNIX
  - Emulation for non-Unix OS

APIs came from UNIX/C

- open
- read
- gets
- write
- printf
- puts
- fcntl
- ioctl
- stat
- select
- getuid
- setuid
- fork

Better UNIX

- Some API's origins are in UNIX/C
- However, their behavior is improved
- Examples
  - IO#eof? returns true before read fails while feof(3) does't.
  - IO.select can be used for buffered IO while select(2) can't.

Ruby for object-oriented programming

- Problem
  - Object-oriented programming is excellent
  - But too heavy for daily scripting
- Solution
  - Non-OO style syntax
  - Pure object-oriented semantics

Non-OO style syntax

```
def fib(n)
  if n < 2
    return n
  else
    return fib(n - 2) + fib(n - 1)
  end
end
puts fib(20)
```

- fib is just a function, isn't it?

Pure object-oriented semantics

- Receivers can be omitted
  - fib(20) is a short form of self.fib(20)
- At top level:
  - self is an instance of Object
  - def defines a method in Object
- All data including integers are objects
- Most operators such as - are methods

Ruby for fun

- For Matz's fun
  - Creating a new language was his dream
- For your fun
  - Who are YOU?
  - YOU = programmers

My first contact with Ruby

- In 1997
- I was a Java programmer
    - Please don't throw a stone at me!
- Posted my regexp library in a Java ML
- Someone said "Ruby's Regexp is better"
- Threw away Java, and fell in love with Ruby
- Got involved with Ruby development

How I learned to stop worrying and love Ruby

- My worries were:
  - Ruby is unpoplular, isn't it?
  - Ruby is slow, isn't it?
  - Dynamic typing is unsafe, isn't it?

Ruby is unpopular, isn't it?

- Yes, it **was**.
  - No books
  - No real world applications
  - No recruitment for Ruby programmers
- But all the more, Ruby was worth learning
  - Ruby was my "Secret Weapon"
  - Read Paul Graham's "Beating the Averages"

Ruby is slow, isn't it?

- Yes, it's slow because it's:
  - Dynamically typed
    - Can't use type information for optimization
  - Pure object oriented
    - No primitive types like int in Java
  - Extremely dynamic
    - Method redefinition etc...
- But, the slowness is acceptable
  - At least for I/O bound applications

Dynamic typing is unsafe, isn't it?

- Yes, so write test code
- Ruby programming is like riding a motorcycle
  - You can go anywhere anytime you want
  - But you may sometimes slip down
- It's fun for me

Ruby 1.0 - 1.6

- 1997 - 2002
- Many changes

Ruby 1.8

- The first release was on Aug 4th 2003
- Stable
- The most successful version of Ruby

Ruby 1.8 is past

- 1.8.8
  - Never released
- 1.8.7
  - Only bug fixes until June 2012
  - Only security fixes until June 2013

Some of my past works

- puts
- callcc
- protected

puts

- writeln and println were rejected
  - They came from Pascal and Java
  - Matz likes neither Pascal nor Java
- I proposed the name puts [ruby-dev:771]
  - It came from C
  - Matz likes C
  - The behavior is a bit weird
    - puts [1,2,3]

callcc

- Introduced callcc into Ruby [ruby-dev:4206]
- callcc = call with current continuation
- It provides first class continuations
- It came from Scheme
- Removed from built-in libraries in Ruby 1.9
- It may be completely removed in Ruby 2.0

Example of callcc

```
01: require "continuation"
02:
03: cont = nil
04: x = callcc { |c|
05:    cont = c
06:    "first"
07: }
08: p x
09: if x == "first"
10:    # go to line 04 with value "second"
11:    cont.call("second")
12: end
```

Non-deterministic problems

- Baker, Cooper, Fletcher, Miller, and Smith live on different floors of an apartment house that contains only five floors.
- Baker does not live on the top floor.
- Cooper does not live on the bottom floor.
- Fletcher does not live on either the top or the bottom floor.
- Miller lives on a higher floor than does Cooper.
- Smith does not live on a floor adjacent to Fletcher's.
- Fletcher does not live on a floor adjacent to Cooper's.
- Where does everyone live?

Solution

```
require "amb"

a = Amb.new

baker = a.choose(1, 2, 3, 4, 5)
cooper = a.choose(1, 2, 3, 4, 5)
fletcher = a.choose(1, 2, 3, 4, 5)
miller = a.choose(1, 2, 3, 4, 5)
smith = a.choose(1, 2, 3, 4, 5)

a.assert([baker, cooper, fletcher, miller, smith].uniq.length == 5)
a.assert(baker != 5)
a.assert(cooper != 1)
a.assert(fletcher != 1 && fletcher != 5)
a.assert(miller > cooper)
a.assert((smith - fletcher).abs != 1)
a.assert((fletcher - cooper).abs != 1)

p [baker, cooper, fletcher, miller, smith]
```

Implementation of Amb

```
class Amb
  ...
  def choose(*choices)
    choices.each { |choice|
      callcc { |fk|
        @back << fk
        return choice
      }
    }
    failure
  end

  def failure
    @back.pop.call
  end

  def assert(cond)
    failure unless cond
  end
```

Why callcc is evil

- Objects are mutable in Ruby
  - callcc doesn't restore the state of objects
- C calls and Ruby calls are nested in Ruby
  - C calls are also restored by callcc
  - Most C code doesn't take care of it

protected

- Method visibility
- Equivalent to Java's protected
- Useful to implement binary operators

```
def ==(other)
  @repr == other.repr # error if repr is private
end

protected

def repr; @repr; end
```

abnormal use of protected

- Often seen in Rails applications

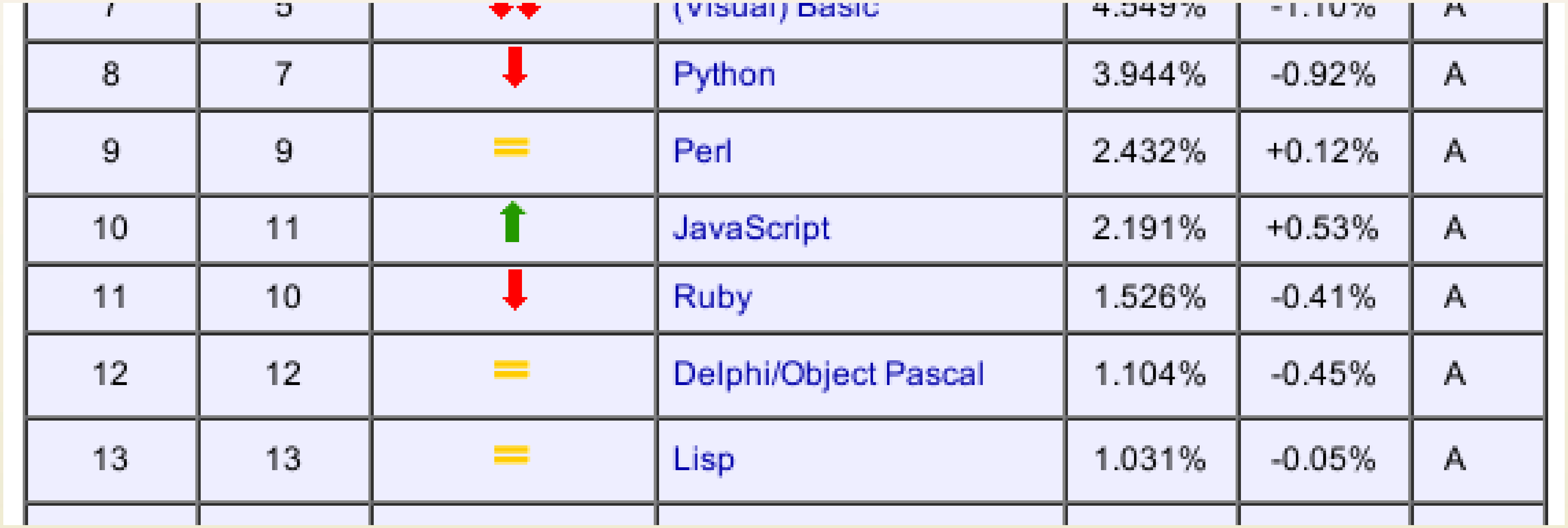
```
class ApplicationController < ActionController::Base
  before_filter :login_required

  protected  # Why not private?

  def login_required
    ...
```

- Use private instead if possible
  - private methods can be called from subclasses

Ruby's present

Mainstream Language

- TIOBE Index (October 2011)



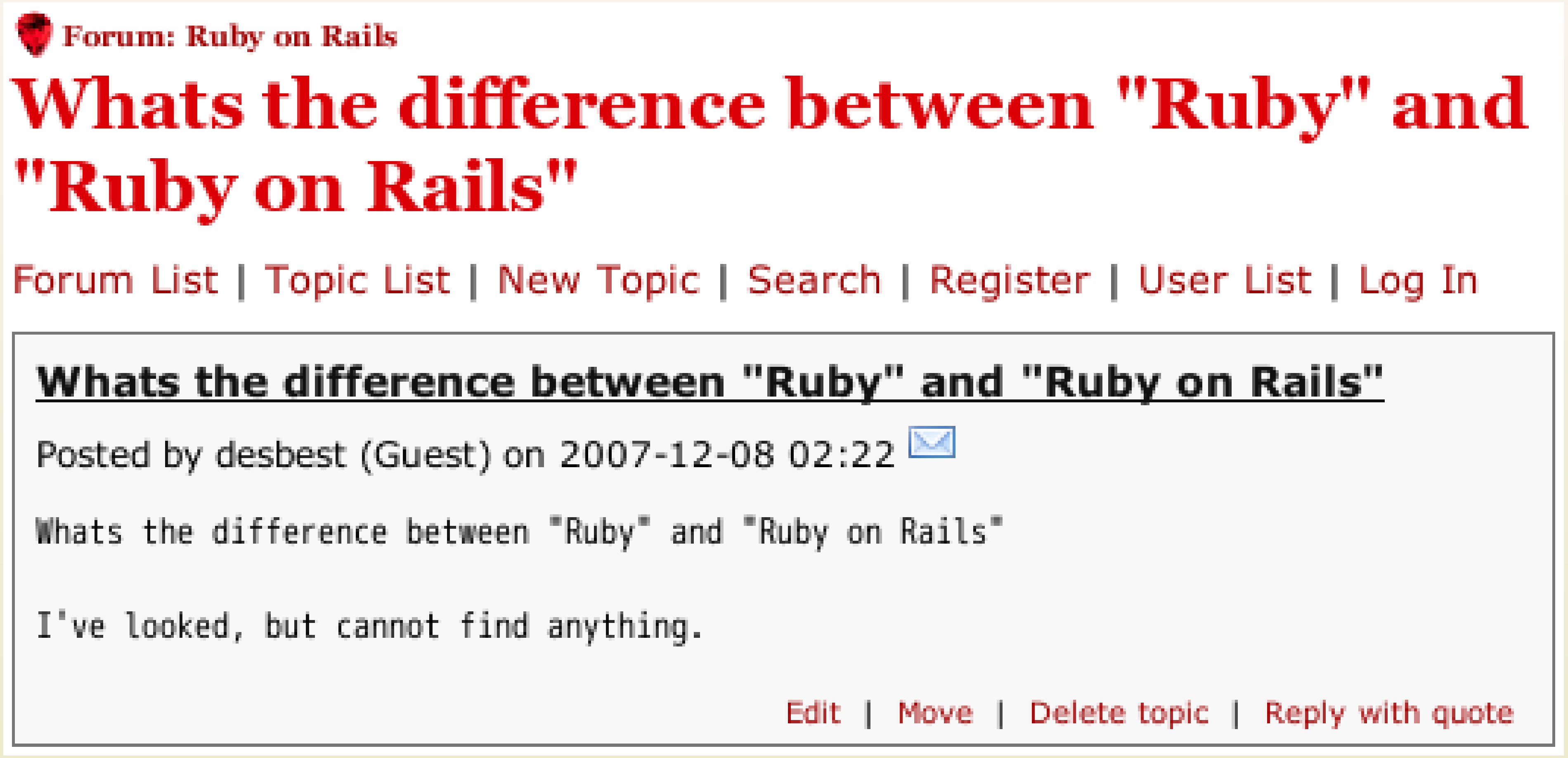
7	5	⇊	(Visual) Basic	4.549%	-1.10%	A
8	7	↓	Python	3.944%	-0.92%	A
9	9	=	Perl	2.432%	+0.12%	A
10	11	↑	JavaScript	2.191%	+0.53%	A
11	10	↓	Ruby	1.526%	-0.41%	A
12	12	=	Delphi/Object Pascal	1.104%	-0.45%	A
13	13	=	Lisp	1.031%	-0.05%	A

- "A" means mainstream

Ruby on Rails

- Rails made Ruby more popular
- Rails may be more popular than Ruby

Difference between Ruby and Rails?



Forum: Ruby on Rails

Whats the difference between "Ruby" and "Ruby on Rails"

Forum List | Topic List | New Topic | Search | Register | User List | Log In

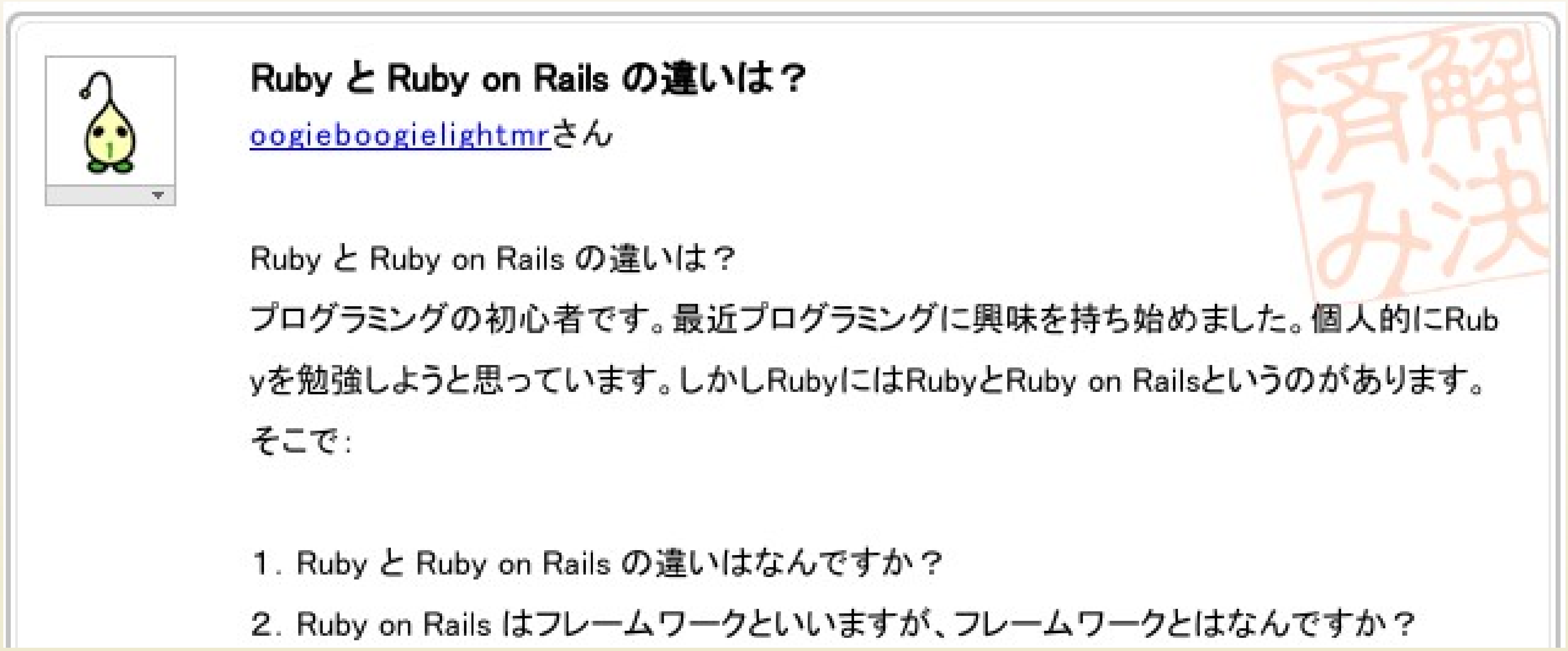
Whats the difference between "Ruby" and "Ruby on Rails"

Posted by desbest (Guest) on 2007-12-08 02:22

```
Whats the difference between "Ruby" and "Ruby on Rails"

I've looked, but cannot find anything.
```

Edit | Move | Delete topic | Reply with quote

Same question in Japan



Ruby と Ruby on Rails の違いは？

oogieboogielightmrさん

Ruby と Ruby on Rails の違いは？

プログラミングの初心者です。最近プログラミングに興味を持ち始めました。個人的にRubyを勉強しようと思っています。しかしRubyにはRubyとRuby on Railsというのがあります。

そこで：

1．Ruby と Ruby on Rails の違いはなんですか？

2．Ruby on Rails はフレームワークといいますが、フレームワークとはなんですか？

Ruby standard

- JIS X 3017
  - published on March 22nd 2011
  - JIS = Japanese Industrial Standards
- ISO/IEC JTC1 Fast-Track procedure
  - "Voting closed 6 September; it received a 100% approval. Only Japan made comments."
    - http://grouper.ieee.org/groups/plv/DocLog/300-399/360-thru-379/22-WG23-N-0364/n0364.pdf

Why Ruby standard?

- Business reasons
  - Tranquilizer for enterprise users
  - Necessary for government procurement
- Technical reason
  - Written specification may help development
    - I have found some bugs in CRuby:)
    - The standard may also have bugs:(

How Ruby has been standardized

- Codified the existing (implicit) specification
  - No new invention by the standardization WG
  - Ruby development is kept free
- Asked public comments from the community
  - Over 100 comments

Ruby 1.9

- New implementation
- New syntax
- Other new features

New implementation

- YARV = Yet Another Ruby VM
- It's now **the** Ruby VM
- Word-code interpreter
  - The size of opcode and operands is the size of pointers
- Faster than Ruby 1.8

New syntax

- New hash syntax
- New syntax for blocks and lambda
- Extended splat

## New hash syntax

```
# New hash syntax
h = {a: 1, b: 2, c: 3}
# equivalent to h = {:a=>1, :b=>2, :c=>3}

# Use new hash syntax for optional arguments
foo opt1: 123, opts2: 456
```

- Hash is now ordered

New syntax for blocks and lambda

```
# extended block parameters
foreach = lambda { |list, method = :each, &block|
  list.send(method, &block)
}
foreach.call([1,2,3]) do |i|
  p i
end
# New lambda syntax
add = ->(x, y) { x + y }
p add.(1, 2)  # same as add.call(1, 2) or add[1, 2]
```

Extended splat

```
def foo(a, b, *c, d)
end
a, *b, c = [1,2,3,4,5]
x = [1,2,3]; y = [4,5,6]
ary = *x, *y
foo *x, *y
```

Other new features

- M17N
- Fiber
- Enumerator

M17N

- M17N = multilingualization
- Code Set Independent (CSI)
  - Not UCS Normalization
  - Unicode is just one of supported character sets
- Strings are character strings
  - In Ruby 1.8, strings are byte strings

Fiber

- Semi-coroutines
- Coroutines are similar to subroutines
  - But have multiple entry points
- Semi-coroutines are restricted coroutines
  - Parent/child relationship

Example of Fiber

```ruby
fib = Fiber.new {
  i, j = 0, 1
  Fiber.yield(i)
  Fiber.yield(j)
  loop do
    i, j = j, i + j
    Fiber.yield(j)
  end
}

10.times do
  puts fib.resume
end
```

Enumerator

- Some methods return Enumerator
  - String#lines, String#chars etc.
- Enumerator is a lazy list
- Enumerator is Enumerable
- Enumerator is an external iterator

Example of Enumerator

```
s = <<EOF
ruby
perl
python
EOF
puts s.lines.each_with_index.select { |line, i|
  /p/ =~ line
}.map { |line, i|
  format("%3d: %s", i + 1, line)
}
```

Enumerator as an external iterator

```
lines = ARGF.lines
10.times do
  puts lines.next
end
```

Ruby 1.9 is present

- Everyone should use it now
- Migration from Ruby 1.8 to 1.9 is easier than migration from Rails 2 to Rails 3

Recent trends in Ruby

- Functional programming
- Monkey patching

Functional programming

- "a programming paradigm that treats computation as the evaluation of mathematical functions and **avoids state and mutable data**" from Wikipedia

Concepts in functional programming

- Pure functional functions
  - No side effects
  - Expressions over statements
- Higher-order functions
  - Take functions as arguments
  - Return functions

Functional programming in Ruby

- Advantages
  - Blocks and lambda
  - Methods like higher-order functions
    - Enumerable#map etc...
  - Almost everything is an expression
- Disadvantages
  - No real function
  - Almost everything is mutable

New features for functional programming

- Proc#curry
- Enumerable#flat_map

Proc#curry

- Useful for partial application

```
add = lambda { |x, y|
  x + y
}
curried_add = add.curry
# => lambda {|x| lambda {|y| x + y}}
add1 = curried_add.call(1)
p add1.call(2) #=> 3
```

Enumerable#flat_map

```
def queens(n, k = n)
  if k == 0
    [[]]
  else
    queens(n, k - 1).flat_map {|qs|
      (1..n).map {|col| [k, col]}.select {|q|
        qs.all? {|q2|
          q[1] != q2[1] &&
            (q[0] - q2[0]).abs != (q[1] - q2[1]).abs
        }
      }.map {|q| [q, *qs]}
    }
  end
end
```

Monkey patching

- Classes and modules are also mutable
- Runtime extension of classes and modules

Use cases of monkey patching

- Workaround for a bug of a library
- Plugin system like Rails
- Extensions of built-in classes
  - Often used for internal DSLs

```
@empty_array.size.should == 0
```

alias_method_chain

```
ActionView::Helpers::RenderingHelper.module_eval do
  def render_with_update(options = {}, locals = {},
                                            &block)
    if options == :update
      update_page(&block)
    else
      render_without_update(options, locals, &block)
    end
  end

  alias_method_chain :render, :update
end
```

Ruby's future

Ruby 2.0

- Had been a vaporware for a long time
  - Matz mentioned Ruby 2.0 at RubyConf **2001**
- Something like Web 2.0?

Ruby 2.0 is real

```
$ ruby-trunk -v
ruby 2.0.0dev (2011-10-31 trunk 33588) [i686-linux]
$  fgrep -B2 '2.0' ChangeLog
Wed Oct 19 17:06:54 2011  Yukihiro Matsumoto  <matz@
ruby-lang.org>

   * version.h (RUBY_VERSION): finally declare start
 of 2.0 work!
$
```

Ruby 2.0 is future

- It's near future
- Current schedule

– Aug 24th 2012	big-feature freeze
– Oct 24th 2012	feature freeze
– Feb 2nd 2013	2.0 release

New features in Ruby 2.0

- Accepted features
  - Keyword arguments
  - Module#prepend
- Not accepted features
  - Enumerable#lazy
  - Refinements

Keyword arguments

- support for formal arguments

```
def foo(x, y, *a, opt1: "foo", opt2: 0, **h)
  p [x, y, a, opt1, opt2, h]
end
foo(1, 2, 3, 4, opt1: "bar", opt2: 2, opt3: 2)
#=> [1, 2, [3, 4], "bar", 2, {:opt3=>3}]
```

- opt1: "foo" defines a keyword argument opt1 whose default value is "foo"
- **h receives the rest keyword arguments

Module#prepend

- Replacement of alias_method_chain

```ruby
module RenderUpdate
  def render(options = {}, locals = {}, &block)
    if options == :update
      update_page(&block)
    else
      # call the original RenderingHelper#render
      super(options, locals, &block)
    end
  end
end
ActionView::Helpers::RenderingHelper.module_eval do
  prepend RenderUpdate
end
```

Enumerable#lazy

- Proposed by @yhara

```
def pythagorean_triples
  (1..Float::INFINITY).lazy.flat_map {|z|
    (1..z).lazy.flat_map {|x|
      (x..z).lazy.select {|y|
        x**2 + y**2 == z**2
      }.map {|y|
        [x, y, z]
      }
    }
  }
end
p pythagorean_triples.take(10)
```

Considerations for Enumerable#lazy

- Is lazy a good name?
  - Is view, delay, or defer better?
- Is lazy necessary?
  - Why not Enumerator#map returns an Enumerator instead of an Array?

Refinements

- Scoped monkey patching

```
module MathN
  refine Fixnum do
    def /(other)
      quo(other)
    end
  end
end
module Foo
  using MathN
  p 1 / 2 #=> (1/2)
end
p 1 / 2 #=> 0
```

Considerations for Refinements

- Performance issue
  - It's slow even if refinements are not used
- Scope of refinements
  - Lexical scoping is safe, but not flexible
  - Do we need refinement propagation?

Who creates Ruby's future?

- You!

2011 Call for grant proposals

- Grants for development projects
- Anyone can submit proposals
- Grant size: 500,000 yen (JPY)
- Selection criteria
  - Impact on the productivity and performance of Ruby and its environment
  - Originality and creativity of the project
  - Feasibility of the project

How to submit a proposal

- Please send an email
- See our web site for details
  - http://www.ruby-assn.org/en/releases/20111025_grant.htm

Conclusion

Ruby's past

- Ruby was created for fun
- Ruby 1.8 is past

Ruby's present

- Ruby is now mainstream
- Ruby 1.9 is present

Ruby's future

- Ruby 2.0 is future
- You can change it!

Thank you!

Any questions?