Ruby Metaprogramming

Zhang Yuning

March 5, 2016

Hello, world!

```ruby
# myapp.rb
require 'sinatra'

get '/' do
  'Hello world!'
end
```

Ruby's Feature

- Completely object-oriented
- Blocks
- Metaprograming
- Dynamic typing
- Garbage collection
- Rails Girls

Ruby Basics

In ruby there is only expressions:

```
winston = if 2 + 2 == 5
            'Ignorance is strength'
          else
            'Freedom'
          end # => 'Freedom'
winston # => 'Freedom'
```

Ruby Basics

Symbols, like in Lisp:

```
:symbol
```

Used mainly in metaprogramming methods

```
attr_reader :length
```

Ruby Basics - Naming convention

predicates' name should end with ?

```
block_given?
empty?
```

'impure' methods' name should end with !

```
reverse!
reverse
```

Ruby Basics

Blocks are a important feature of Ruby:

```ruby
[1, 2, 3].each do |x|
  puts x
end
```

We will meet it often.

Ruby Basics

every method can be given a block and call it with yield

```ruby
def call_with_42
  if block_given?
    yield 42
  end
end
call_with_42 {|x| puts x}
```

Everything is object

Fixnum is object:

```
3.times do
  puts 'quark'
end
# even
3.days.ago
```

Everything is object

nil is also object:

```ruby
# Note that built-in Classes can be modified
class Object
  def try
    if block_given?
      yield self
    end
  end
end

class NilClass
  def try
    nil
  end
end
```

Everything is object

Usage:

```
nil.try{|x| x + 1} # => nil
2.try{|x| x + 1} # => 3
```

```
Nothing >>= (\x -> Just (x + 1)) # => Nothing
Just 2 >>= (\x -> Just (x + 1)) # => Just 3
```

Everything is object

Classes are objects, too!

```ruby
class A
end
A.class # => Class
Class.class # => Class
```

Everything is object

From a linguist's point of view, Blocks are not objects. But there are three ways to convert them to objects. (Note: there are subtle differences between each other)

```
Proc.new {|x| x + 1}
lambda {|x| x + 1}
-> x { x + 1 } # introduce in ruby 1.9
```

```
p = Proc.new {|x| x + 1}
p.(1)
p[1]
p.call 1
```

Note that object is first-class. So...

Duck Typing

But there's an old American phrase about if it walks like a duck and quacks like a duck and so forth, it's a duck.

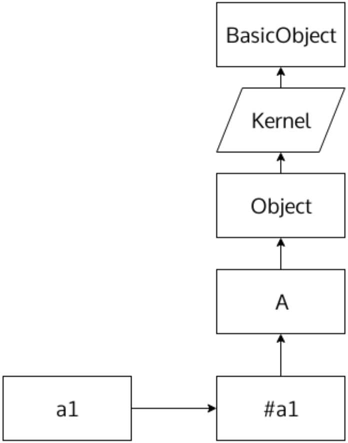
— *Mike Wallace*

```ruby
# taken from Sinatra (modified)
if boom.respond_to? :http_status
  status(boom.http_status)
elsif boom.respond_to?:code and boom.code.between? 400,599
  status(boom.code)
else
  status(500)
end
```

Singleton Method

```ruby
class A
  def hello
    'hello'
  end
end
a1 = A.new
a2 = A.new
def a1.bye
  'bye'
end
a1.hello # => 'hello'
a1.bye # => 'bye'
a2.hello # => 'hello'
a2.respond_to? :bye # => false
```

Method Lookup



Class definition

Class definition in Ruby just changes the environment. You can invoke method in them.

```
class A
  puts 'hello, world'
end
```

define_method & method_missing

The html example.

Hooks

```ruby
at_exit { Application.run! if $!.nil? && Application.run? }

class Base
  def inherited(subclass)
    subclass.reset!
    subclass.set :app_file, caller_files.first unless subclass.app_file?
  end
end
```

Wrap up - The logger decorator

```ruby
class A
  include Logger

  def f
    puts 'hello, world!'
  end

  add_logger :f
end
# output:
# f started
# hello, world!
# f finished
```