Meta Programming Ruby

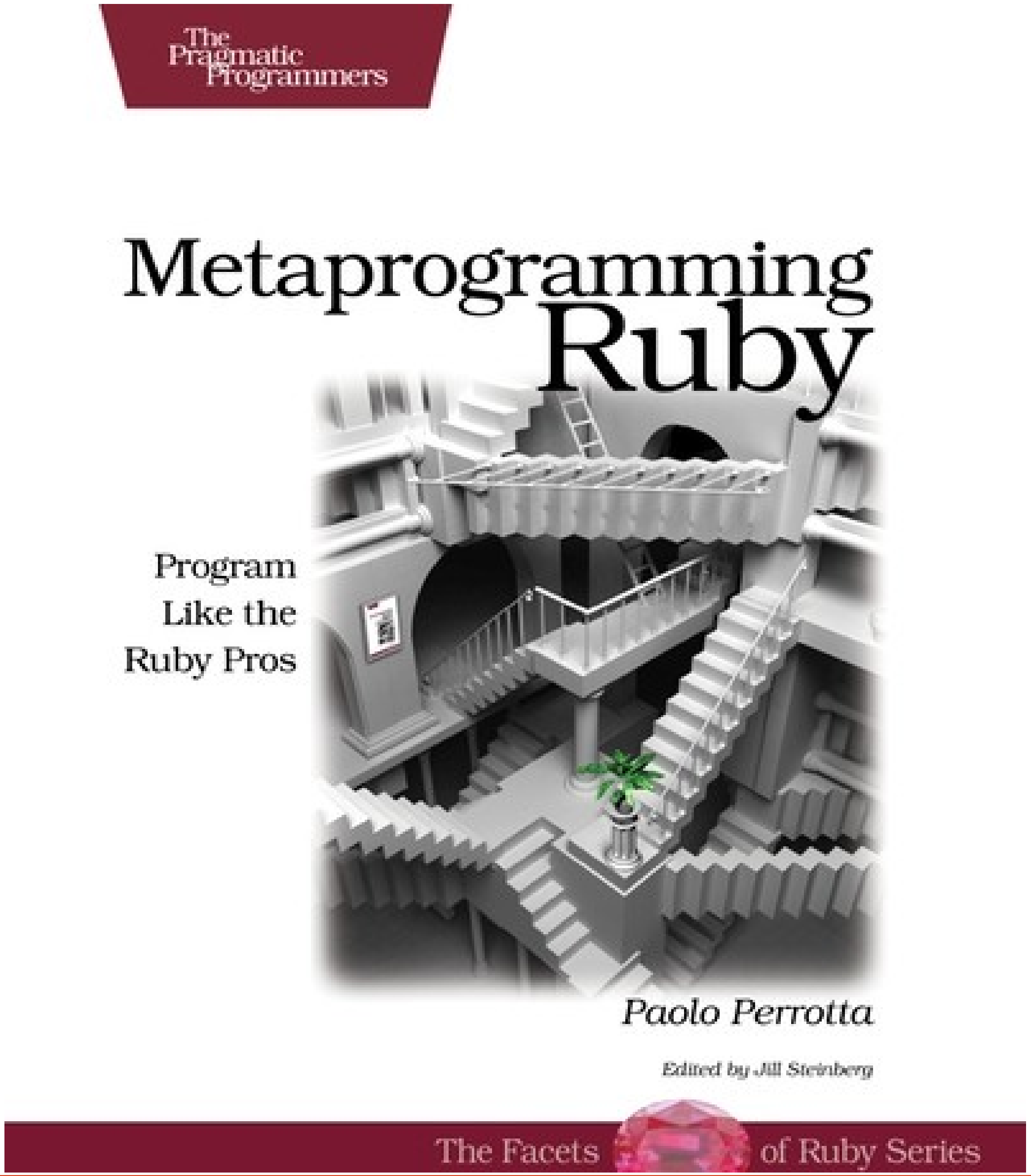
概念： code that writes code.

双刃剑：
- 优点：可以把代码写的很优雅 / 漂亮 / 易于理解。
- 缺点：也可以把代码写的一团糟，难于测试。

测试：
- 把它看成普通代码。该怎么测试就怎么测试。 (test/unit, rspec)

我们一定要掌握的理由：
- 深入理解 RUBY 的必经之路。
- 看懂开源项目代码的必要条件。

The book



The Pragmatic Programmers

Metaprogramming Ruby

Program Like the Ruby Pros

Paolo Perrotta

Edited by Jill Steinberg

The Facets of Ruby Series

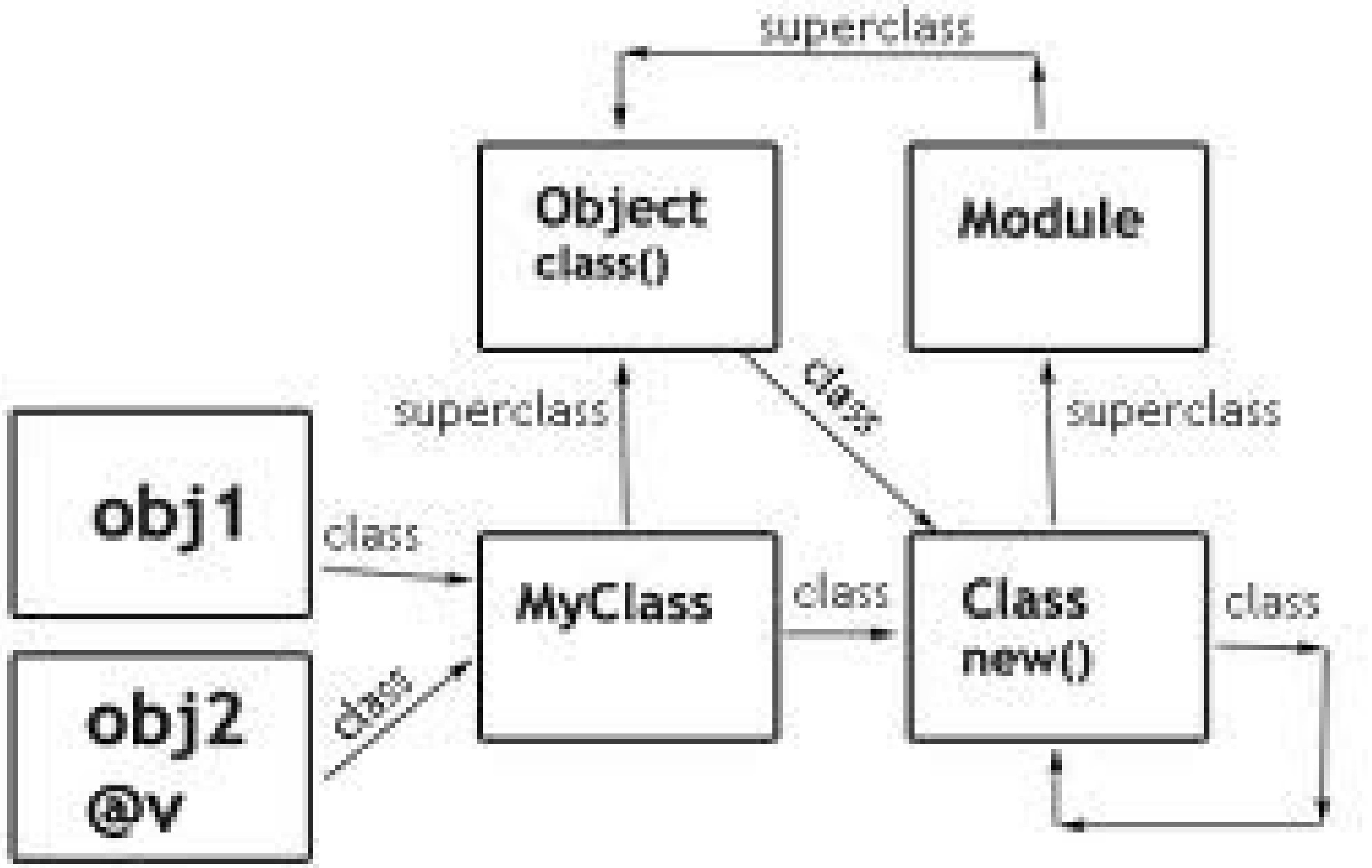
ruby structures

```
class Apple
end

apple = Apple.new
apple.class # => Apple
Apple.class # => Class
Apple.ancestors # => [Apple, Object, Kernel]
Apple.superclass # => Object
Class.superclass # => Module
Module.superclass # => Object
Object.class # => Class
```

简言之：　所有的东西都是 Object，它们的类，都叫 class.

ruby structure diagram



method receiver

receiver: an object that you call a method on.

```
receiver = "some string"
receiver.reverse()    # here the 'receiver' is the receiver.
```

```
irb(main):001:0> receiver = "some string"
=> "some string"
irb(main):002:0> receiver.reverse()
=> "gnirts emos"
```

what is self in Ruby

the receiver object, the current receiver

```
puts self  # => main, an instance of Object

class Foo
  puts self   # => Foo
end

puts self  # => main .
```

core methods

To learn about the following methods read The Book of Ruby, Chapter 20: Dynamic Programming.

1. eval, instance_eval , class_eval (aka: module_eval)
2. class_variable_set, class_variable_get, class_variables (Try it out: instance_variables), instance_variable_set (Try it out: instance_variable_get)
3. define_method, send (Try it out: method), remove_method, undef_method, method_missing
4. const_set, const_get (Try it out: constants)
5. Class.new (Try it out: Struct.new)
6. binding (Try it out: lambda)

refactoring:

```
def eat_apple
  puts “apple is great!”
end

def eat_banana
  puts “banana is great!”
end

[:apple, :banana].each do |fruit|
  define_method “eat_#{fruit}” do
    puts “#{fruit} is great!”
  end
end
```

refactoring: 双刃剑

1. 一定要为动态方法加上注释，便于 IDE 找到。

```
# define methods:
# eat_apple
# eat_banana
[:apple, :banana].each do |fruit|
  define_method “eat_#{fruit}” do
    puts “#{fruit} is great!”
  end
End
```

2. 不要搞得太复杂，例如 2， 3 层嵌套。：

```
[:apple, :banana].each do |fruit|
  People.where(:love_fruits => true).all.each do |person|
    (1..5).each { |i|
      define_method do “#{person}_love_#{fruit}”; puts “balbala” ; end
    }
  end
end
```

Basics: eval

eval mean: evaluate

```
string = <<-CODE
  (1..5).each { puts " I love Apple! "}
CODE
eval(string)
```

output : =>

```
 I love Apple!
 I love Apple!
 I love Apple!
 I love Apple!
 I love Apple!
```

Basics: class_eval

```
class Apple
end

Apple.class_eval(%Q{ def say_hi; puts 'hi'; end })
Apple.new.say_hi    # => hi

Apple.class_eval(%Q{ def say_error; raise 'error'; end }, "apple.rb", 123)

Apple.new.say_error    # =>
apple.rb:123:in `say_error': error (RuntimeError)
    from ...
```

Basics: instance_eval

```
apple_string = "apple"
apple_string.instance_eval %Q{ def say_hi; puts 'hi, from apple'; end }

apple_string.say_hi  # => 'hi, from apple'
```

Basic: class_variables

```
class Fruit
  @@name = 'fruit'
end
class Apple < Fruit
  @@color = 'red'
  @@taste = 'good'
end

Fruit.class_variables.inspect  # => ["@@name"]
Apple.class_variables.inspect # => ["@@taste", "@@color", "@@name"]
Apple.class_variable_defined? :@@color # => true
Apple.send(:class_variable_get,:@@color) # => 'red'
Apple.send :class_variable_set,:@@color, 'dark red'
Apple.send(:class_variable_get,:@@color) # => 'dark red'
```

Substitute Class

```
class Apple
  def taste;    puts "good";  end
end

class MockApple
  def taste;    puts "this is a mock apple... tastes also good!";   end
end

Apple.new.taste  # => “good”

Object.send(:remove_const, "Apple")
Object.send(:const_set, "Apple", MockApple)

Apple.new.taste # => "this is a mock apple... tastes also good!"
```

Dynamic define methods

```
class Apple
end

a = Apple.new
a.instance_eval{       def say_hi ;         puts 'hi';       end     }
a.say_hi  # => 'hi'

b = Apple.new
b.say_hi  # NoMethodError: undefined method `say_hi' ….

Apple.class_eval {
    define_method :say_goodbye do;  puts "goodbye";    end
}
Apple.new.say_goodbye # => "goodbye"
```

Scope 被 class/module/def 定义

```
var = "meta"
class Apple
  puts var # undefined local variable or method `var' for Apple...
end

module Banana
  puts var # undefined local variable or method `var' for Banana...
end

class Apple
  color = 'red'
  def show_color;    puts color;  end
end
Apple.new.show_color # undefined local variable or method `color' ...
```

Flat Scope

1. 名字固定的 class.

```
class Apple
  taste = "sweet"
  define_method "taste" do ;     puts "very #{taste}" ;   end
end

Apple.new.taste # =>  “very sweet”
```

2. 动态名字的 class name:

```
operation = "say"
target = "hi"
dynamic_class_name = Class.new do
  define_method operation do;    puts "#{operation} #{target}";  end
end
Kernel.const_set("DynamicClassName", dynamic_class_name)

DynamicClassName.new.say #=> “say hi”
```

Alias method

```
def say_goodbye
  puts 'goodbye'
end

say_goodbye #=> 'goodbye'

alias :original_goodbye :say_goodbye
def say_goodbye
  puts 'my love,'
  original_goodbye
  puts 'and farewell'
End

say_goodbye # => 'my love, goodbye, and farewell'
```

Mixin

就是 include 多个 module

```
module A
  def say_hi;  puts “hi” ; end
end
module B
  def say_goodbye;  puts “goodbye” ; end
end

class C
  include A
  include B
end

c = C.new
c.say_hi  # => “hi”
c.say_goodbye # => “goodbye”
```

Hooks: include/extend

两个关键的钩子方法

```
module M
  def self.extended(another)
    puts "#{self} is extended by #{another}"
  end
  def self.included(another)
    puts "#{self} is included by #{another}"
  end
end

class C; include M end  #=> M is extended by C
class D; extend M end  #=> M is extended by D
```

（inherited 也一样，略）

Extension Mixin 的例子

使用一个 include，就能同时实现 include/extend 的功能（增加 instance / class methods)

```
module M
  def self.included(base)
    base.extend(ClassMethods)
  end
  module ClassMethods
    def say_hi; "hi" end
  end
  def say_goodbye; "goodbye" end
end

class C; include M end
C.say_hi   # => “hi”
C.new.say_goodbye # => “goodbye”
```

Self Yield

```
class Book
  attr_accessor :title, :published_at
  def initialize &block
    yield self
  end
end

book = Book.new do |b|
  b.title = "diablo"
  b.published_at = "2012-12-12"
end

puts book.inspect  # => #<Book:0xb78d0934 @published_at="2012-12-12",
    @title="diablo">
```

Class Method

三种形式定义 class method.   (回香豆的回字四种写法。。。 >_<)

```
class Apple
  def self.color ;    "red";  end    # 第一种
  class << self
    def size;      "big";    end    #  第二种
  end
end
def Apple.taste;  'sweet' ; end  #  第三种

Apple.color  # => “red”
Apple.size # => “big”
Apple.taste  # => 'sweet'
```

Eigen class

eigen /'ei dgen/ , 德语，   one's own

Eigenclass, the class of a class. An object's own class.

跟 singleton class, 类似， 也叫 meta class .

跟 singleton 模式无关。

Eigen class

```
class Object
  def eigenclass
    class << self
      self
    end
  end
end
class Fruit;
  def sef.taste
      'good'
  end
end

class Apple < Fruit; end
Apple.taste # => 'good'
Apple.eigenclass # => #<Apple>
Fruit.eigenclass # => #<Fruit>
Apple.eigenclass.superclass # => 1.9 #<Fruit>      , 1.8: #<Class>
Fruit.eigenclass.superclass # =>  1.9 #<Object>    , 1.8: #<Class>
```

Eigen class Diagram

↑ super class

↑ eigen class

```
class Fruit; end

class Apple<Fruit
end
```

$$\begin{tikzcd}
\text{Object} \arrow[r] & \text{\#Object} \\
\text{Fruit} \arrow[u] \arrow[r] & \text{\#Fruit} \arrow[u] \\
\text{Apple} \arrow[u] \arrow[r] & \text{\#Apple} \arrow[u]
\end{tikzcd}$$

Symbol to Proc（&:method)

只适用于　receiver == 集合中 element 的场合。

```
class String
   def contains_a_or_c?
     match(/a|c/)
   end
end

array = ["aaa", "bbb", "ccc", "ddd"]

result1 = array.collect(&:contains_a_or_c?)
result2 = array.collect{ |element| element.contains_a_or_c? }

result: [#<MatchData "a">, nil, #<MatchData "c">, nil]
```

不适合：　array.collect{ |element| String.contains_a_or_c?(element) }

rspec 中的一个例子

Rpsec mock 的一个典型用法：

```
class Apple;
  def color;  'red;  end
end

red_apple = Apple.new
red_apple.color  # => 'red'

green_apple = Apple.new
green_apple.stub(:color) { “green” }
green_apple.color # => “green”
```

rspec 中的一个例子（代码）

lib/rspec/mocks/method_double.rb

```
        #core method:
          stash_original_method
          define_proxy_method

      #  rename 'color', to  “obfuscated_by_rspec_mocks__color”
      def stash_original_method;  end

      def define_proxy_method
        method_name = @method_name
        visibility_for_method = "#{visibility} :#{method_name}"
        object_singleton_class.class_eval(<<-EOF, __FILE__, __LINE__)
          def #{method_name}(*args, &block)
            __mock_proxy.message_received :#{method_name},*args,&block
          end
        #{visibility_for_method}
        EOF
      end
```

Method missing (1)

lib/active_record/attribute_methods.rb, rails 3.2.7

```ruby
# If we haven't generated any methods yet, generate them, then
# see if we've created the method we're looking for.
def method_missing(method, *args, &block)
  unless self.class.attribute_methods_generated?
    self.class.define_attribute_methods

    if respond_to_without_attributes?(method)
      send(method, *args, &block)
    else
      super
    end
  else
    super
  end
end
```

Method missing (2)

```
module ClassMethods
  # Generates all the attribute related methods for columns in the da
  # accessors, mutators and query methods.
  def define_attribute_methods
    unless defined?(@attribute_methods_mutex)
      msg = "It looks like something (probably a gem/plugin) is overr
    end
    # Use a mutex; we don't want two thread simaltaneously trying to
    # attribute methods.
    @attribute_methods_mutex.synchronize do
      return if attribute_methods_generated?
      superclass.define_attribute_methods unless self == base_class
      super(column_names)
      column_names.each { |name| define_external_attribute_method(..
      @attribute_methods_generated = true
    end
  end
```

Method missing (rails code)

lib/active_record/attribute_methods/read.rb

```
def define_external_attribute_method(attr_name)
  generated_external_attribute_methods.module_eval <<-STR...
    def __temp__(v, attributes, attributes_cache, attr_name)
      #{external_attribute_access_code(attr_name, attribute_cas
    end
    alias_method '#{attr_name}', :__temp__
    undef_method :__temp__
  STR
end
```

欢迎提问

结束，欢迎提问。

THANKS!