Introduction to Meta-Programming with Ruby

Ruby宏编程介绍

阿德 – **Adeh**

Director, Technology at Kudelabs
Organizer, StartupGrind Guangzhou

酷德实验室
kude labs

Startup GRIND 广州 guangzhou

adeh@kudelabs.com
http://kudelabs.com
微博: @americanfood

Ruby 例如

“I write C code, so you don’t have to!” - Matz

```
puts "Hello World!"

self.clap_hands if self.happy?

5.times { jump_for_joy! }
```

Ruby来源

- Created by Yukihiro “matz” Matsumoto
- Released 1995
- Made popular by Ruby on Rails in 2005
- Ruby is designed for humans
- “I write C so you don’t have to!” - Matz

Ruby社区

- Online Community - RubyChina.org
- Corporate Support - ruby.taobao.org
- Local Guangzhou Community - gzruby.org
- We meet every 2 months - join us!

What is Meta-Programming
宏编程是什么?

- “Code that writes code”
- Operating on data that happens to be code.
- Making changes to a program during runtime

Everything is an object
一切都是Object

In Ruby, everything you are working with is an object itself. The root class is called Object.

```
5.class.name #=> Fixnum

5.class.class.name #=> Class

5.class.name.class #=> String

5.class.superclass.name #=> Object
```

send()

Rather than calling a method directly like this:

```
obj.method(arg)

student.add_class("Introduction to Ruby")
```

You can call methods programmatically like this:

```
obj.send('method', arg)

student.send("add_class", "Introduction to Ruby")
```

send()

You can use send() in practice like this:

```
def calc(left, operator, right)
  raise "unknown operator" unless
       ['+', '-', '/', '*'].include?(operator)
  left.send(operator,right)
end

calc 1, '+', 1    #=> 2
```

Monkey Patching

Let's add some methods to Array:

```
class Array

  def hello
    "Hello, I'm an #{self.class.name}!"
  end

end

a = [1,2,3,4,5,6,7,8,9,10]
a.hello        #=> Hello, I'm an Array!
```

Monkey Patching

Let's add a useful method to Array:

```
class Array

  def foldl(method)
    inject(0) { |result, element|
      result ? result.send(method, element) : 0
    }
  end

end

a = [1,2,3,4,5,6,7,8,9,10]
puts a.foldl('+')  #=> 55
```

Monkey Patching

Let's add an evil method to Array:

```
class Array
  def first
    self.last
  end
end

a = [1,2,3,4,5,6,7,8,9,10]
puts a.first  #=> 10
```

define_method()

This is a way to create methods in existing classes at runtime:

```
class Drinker
  if $CHINA
     define_method(‘tea’) {
        puts “I drink tea”
     }
  else
    define_method(‘coffee’) {
      puts “I drink coffee”
    }
  end
end

$CHINA = true
tom = Drinker.new
tom.tea #=> I drink tea
tom.coffeee #=> NoMethodError
```

define_method()

A more practical example:

```
class Drinker
  def self.drinks(*drinks)
    for drink in drinks
      define_method("#{drink}_amount") { #... }
      define_method("#{drink}_description") { #... }
      define_method("#{drink}_effect") { #... }
    end
  end
end
```

define_method()

A more practical example:

```
class Student < Drinker
  drinks ‘cola’, ‘tea’, ‘beer’
end

jim = Student.new

jim.cola_amount    #=> a lot!
jim.tea_effect     #=> awake!
jim.beer_bottles   #=> NoMethodError
```

method_missing()

In the root class, `method_missing` is something like this:

```
def method_missing(method_name, *args)
  raise "NoMethodError: #{method_name}
    not defined!"
end
```

Meta-Programming in real life
现实世界的宏编程

- Rails uses meta-programming heavily
- Build strong layers of abstraction
- Create intuitive APIs

attr_accessor

attr_accessor is a common pattern:

```
class Student
  attr_accessor :name, :major, :dorm
  def to_s
    "#{name}学生在学习#{major}，住在#{dorm}"
  end
end

hao       = Student.new
hao.name  = "好朋友"
hao.dorm  = "西区C座9楼304房"
hao.major = "计算机科学"
hao.to_s #=> 好朋友学生在学习计算机科学，住在西区C座9楼304房
```

attr_accessor

Let’s see if we can reproduce this behavior:

```
class Shuxingable

  def self.wo_de_shuxing(*shuxings)
    shuxings.each do |shuxing|
     define_method(shuxing) {
       instance_variable_get("@#{shuxing}")
     }
     define_method("#{shuxing}=") { |p|
       instance_variable_set("@#{shuxing}", p)
     }
    end
  end
end
```

attr_accessor

And it works in the same way:

```
class Student < Shuxingable
  wo_de_shuxing :name, :major, :dorm
  def to_s
    "#{name}学生在学习#{major}，住在#{dorm}"
  end
end


tai       = Student.new
tai.name  = "太漂亮"
tai.dorm  = "东区Q座3楼409房"
tai.major = "计算机科学"
```

ActiveRecord dynamic find

You can make calls like this:

```
student = Student.find_by_name_and_dorm(
  params[:name],
  params[:dorm]
)
teacher = Teacher.find_by_shenfenzhen(params[:sfz])
```

ActiveRecord dynamic find

Lets take a look at how this is implemented:

```
def method_missing(method_id, *arguments, &block)
  if match = DynamicFinderMatch.match(method_id)
   # verify this is a finder method
   self.class_eval <<-EOS, __FILE__, __LINE__ + 1
     def self.#{method_id}(*args)
       # define the finder method
     end
   EOS
   # then call the method we just defined!
   send(method_id, *arguments)
   # … lots more code …
end
```

from rails-2-3-14/activerecord/lib/activerecord/base.rb:1873

Meta-Programming is Fun!
宏编程就好玩

- Very powerful tools.
- Need to use them carefully
- Can be too much of a good thing
- Thanks!