Metaprogramming

Outline

1. Reflection
2. Metaprogramming basic (#send, #method_missing, #define_method)
3. Metaprogramming advance (#eval, #instance_variable_set, #instance_variable_get)

1. Reflection

...a program can examine its state and its structure

```
#print out all of the objects in our system
ObjectSpace.each_object(Class) {|c| puts c}

#Get all the methods on an object
"Some String".methods

#see if an object responds to a certain method
obj.respond_to?(:length)

#see if an object is a type
obj.kind_of?(Numeric)
obj.instance_of?(FixNum)
```

2. Metaprogramming basic

“**Metaprogramming** is a programming technique in which computer programs have the ability to treat programs as their data” https://en.wikipedia.org/wiki/Metaprogramming

# Example code without metaprogramming

2. Metaprogramming basic (cont.)

#send( ) is an instance method of the ***Object class***

```
class Rubyist
  def welcome(*args)
    "Welcome " + args.join(" ")
  end
end
obj = Rubyist.new
puts(obj.send(:welcome, "famous", "Rubyists")) # => Welcome famous Rubyists
```

2. Metaprogramming basic (cont.)

```ruby
class Rubyist
end

rubyist = Rubyist.new

if rubyist.respond_to?(:also_railist)
  puts rubyist.send(:also_railist)
else
  puts "No such information available"
end
```

```ruby
class Rubyist
  private

  def say_hello name
    "#{name} rocks!!"
  end
end

obj = Rubyist.new
puts obj.send(:say_hello, "Matz")
```

2. Metaprogramming basic (cont.)

The **Module#define_method( )** is a private instance method of the class ***Module***

```
class A
  define_method(:wilma) {puts "Touch me!!!"}
end

class B < A
  define_method(:barney) {puts "Call me!!!"}
end

b = B.new
b.barney => "Call me!!!"
b.wilma => "Touch me!!!"
```

2. Metaprogramming basic (cont.)

Kernel#method_missing( ) responds by raising a ***NoMethodError***

```
class Caller
  def method_missing(m, *args, &block)
    puts "Called #{m} with #{args.inspect} and #{block}"
  end
end

Caller.new.anything
# => Called anything with [ ] and

Caller.new.anything(3, 4) {something}
# => Called anything with [3, 4] and #<Proc:0x02efd664@tmp2.rb:7>
```

3. Metaprogramming advance

The module **Kernel** has the **eval()** method and is used to execute code in a string

```
str = "Hello"
puts eval("str + ' Rubyist'") # => "Hello Rubyist"
```

3. Metaprogramming advance (cont.)

The **eval()** method can evaluate strings spanning many lines, making it possible to execute an entire program embedded in a string

=> Slow

=> Dangerous (difficult to manage external data)

=> Considered a method of last resort

Read more about **#instance_eval**, **#module_eval**, **#class_eval**

3. Metaprogramming advance (cont.)

```ruby
class Person
  def initialize(p1, p2)
    @geek, @country = p1, p2
  end
end

obj = Person.new("Matz", "USA")
puts obj.instance_variable_get(:@geek) # => Matz
puts obj.instance_variable_get(:@country) # => USA
```

3. Metaprogramming advance (cont.)

```ruby
class Person
  def initialize(p1, p2)
    @geek, @country = p1, p2
  end
end

obj = Person.new("Matz", "USA")
obj.instance_variable_set(:@country, "Japan")
puts obj.inspect # => #<Rubyist:0x2ef8038 @country="Japan", @geek="Matz">
```

Read more about: **#class_variable_get**, **#class_variable_set**, **#const_get**, **#const_set**, **#class_variables**

Thank you for listening!