
Metaprogramming the Ruby way

Joannah Nanjey
RubyConf Kenya, 8th, June, 2017

About Me

- Joannah Nanjekye (@Captain_Joannah)
- Software Engineer
- Aeronautical Engineering Student
- Organizer Rails girls Kampala
- RGSoC Alumnae (2016)
- Outreachy Intern with Ceph (May - August 2017)
- FOSS contributor & Authoring something tech.



Outline

What is metaprogramming

Metaprogramming techniques

What is Metaprogramming

Writing code that writes code for food.

A look at languages

C ++ - language constructs are no more during runtime.

Java - c# - language constructs survive enough to be introspected.

Ruby - Runtime is very busy with constructs much alive.

Ruby's busy runtime gives us power

To do introspection at runtime

```
class Hello
```

```
  def initialize(word)
    @word = word
  end
```

```
  def shout
    Puts word
  end
```

```
end
```

```
obj = Hello.new("Hello world")
```

```
puts obj.class.name           #Hello
```

```
puts obj.instance_of?(Hello)  #true
```

```
puts obj.methods.inspect
```

```
puts obj.instance_variables.inspect  #@word
```

Ruby's busy runtime gives us power

To modify constructs at runtime

```
class Student < ActiveRecord::Base
end

student = Student.create
student.name = "Ruby Smith"
movie.name
```

Therefore...

Writing code that manipulates language constructs at runtime.

Motivation

Ability for the Language to grow.
Make the language dance to you.

Avoid Duplication.

Manage Scopes.

Add methods to objects.

Problem1 : Ruby not dancing to our tune

```
def strip_alphanumeric(s)
  s.gsub /[^\\w\\s]/, "
end
```

Open Classes to the rescue

```
class String
  def to_alphanumeric
    gsub /[^\\w\\s]/, ""
  end
end
```

Note on Open Class (Monkey Patch)

- Do you really have to open the class?
 - May make sense for generic methods to the class.
 - Alternatively Just add a new method to the class.
-

Problem2 : Duplication

```
class RGSoC

  def initialize(individual_id, db)
    @individual_id = individual_id
    @db = db
  end

  def intern
    info = @db.get_intern_info(@individual_id)
    return "#{info}"
  end

  def mentor
    info = @db.get_mentor_info(@individual_id)
    return "#{info}"
  end
end
```

Dynamic Dispatch and Methods to the rescue

Dynamic Dispatch

```
class Number
  def add (x, y)
    return x + y
  end
end

obj = Number.new()
obj.add(2, 3)

obj.send(add, 2, 3)
```

Dynamic Methods

```
class Number
  define_method :add do |x, y|
    return x + y
  end
end

obj = Number.new()
obj.add(2, 3)
```

Reviewing the duplicated code

```
|
class RGSoc

  def initialize(name, db)
    @name = name
    @db = db
  end

  def self.define_individual(@name)
    define_method("#{@name}") do |@name|
      info = @db.send (get("#{@name}")_info, @name)
      return "#{info}"
    end
  end
end
end
```

Note : Dynamic Dispatch

- Using send breaks Encapsulation unwillingly.

Problem 3 : Work Around scopes

```
class Scope

  age = 2
  x = 0

  def scope_method(y)

    x = 4
    age = 5

  end
  x

end
|
```

Rescue 1 : Nested Lexical scopes

```
var1 = 1
```

```
Scope = Class.new do
```

```
  puts "#{var1} in the class definition!"
```

```
  define_method :my_method do
```

```
    puts "#{var1} in the method!"
```

```
  end
```

```
end
```

Rescue 2 : Shared Scopes

```
def define_methods
  shared = 0
  define_method :counter do
    shared
  end
  define_method :inc do |x|
    shared += x
  end
end
```

Note : Breaking Encapsulation

Do if you must.

Problem 4 : Adding methods to object

```
class MyClass
```

```
  def add(x,y)
```

```
    x+y
```

```
  end
```

```
end
```

```
obj1 = MyClass.new
```

```
obj2 = MyClass.new
```

Singleton Methods to the rescue

```
class MyClass
  def add(x,y)
    x+y
  end
end
```

```
obj1 = MyClass.new
obj2 = MyClass.new
```

```
class << obj2 #this places you in the scope of eigen class
  def singleton_method
    "Hello"
  end
end
```

```
obj2.singleton_method # => "Hello"
```



