Metaprogramming in Ruby

Putting the FUN in conFUsionN

Dr. Brett Giles

YYCRuby Meetup

2015-05

Metaprogramming

The Art and Science of programs that write programs:

- Compilers, Lex, YACC
- Macros in languages like C
- Templates in C++
- Interpreters that allow you to evaluate strings

In Ruby...

- `define_method`, `define_singleton_method`
- `method_missing`
- Reflection methods
  (`methods`, `responds_to?`, `send`, `__send__`, `public_send`)
- The eval/exec methods
  (`class_eval`, `instance_eval`, `eval`) and
  (`class_exec`, `instance_exec`, `exec`)
- Hook methods
  (`included`, `prepended`, `extended`, `inherited`)
- Reopening Classes (Monkeypatching!)

Why or why not metaprogramming?

Uses and goodies:

- Domain Specific Languages
- DRY code
- Nice fit for the Adaptor pattern with external libraries

Not so fun things:

- Code obfuscation
- Clever code - Too many levels of indirection
- Naming!

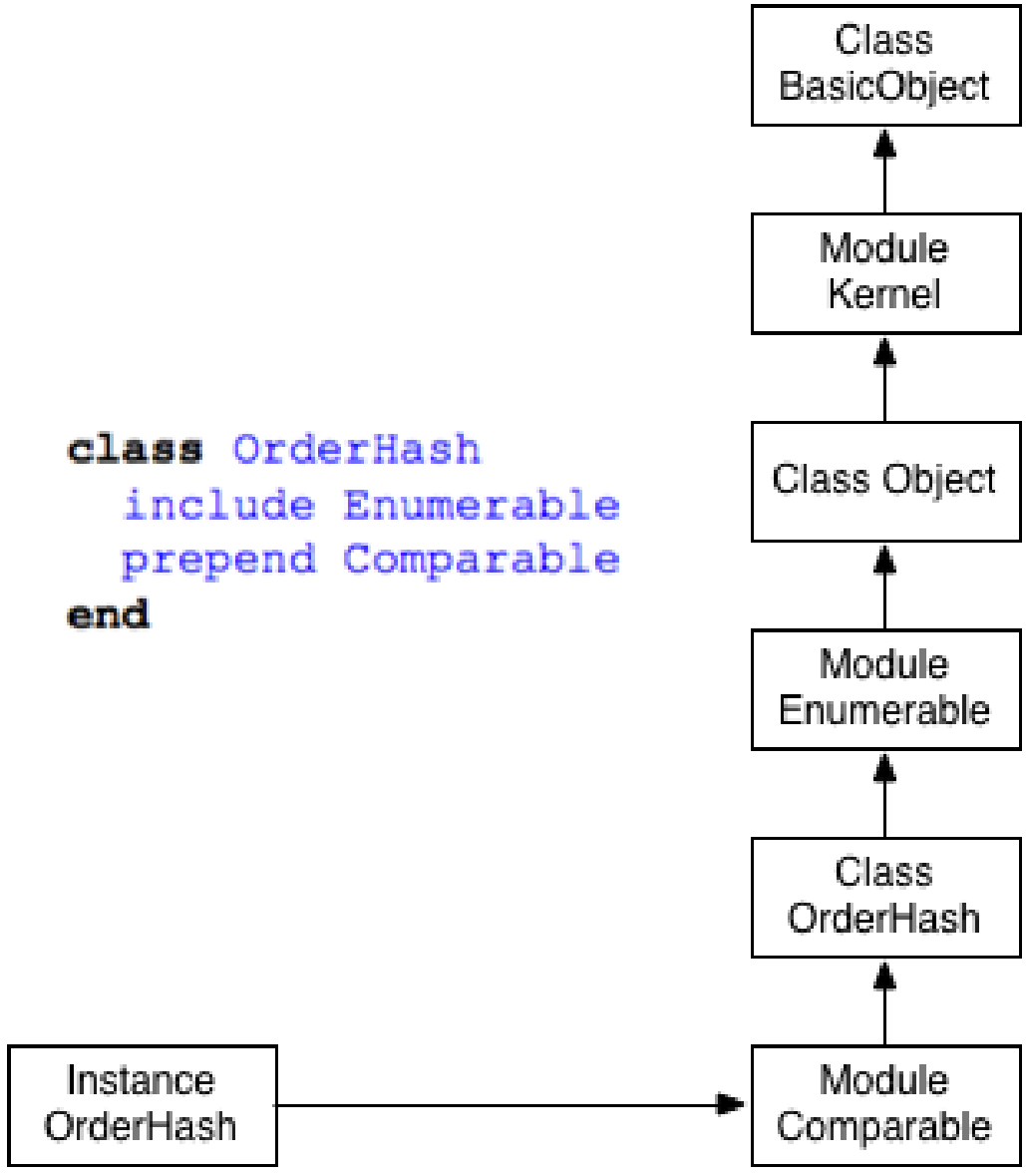
Where do we start?

Common metaprogramming tasks:

- Create a class macro
- Handle a variety of similarly named methods doing the “same” thing
- Evaluate some code in the context of an object

The object hierarchy in Ruby

```ruby
class OrderHash
  include Enumerable
  prepend Comparable
end
```



Create a class macro

```
Class Attr
  def self.my_attr(attribute)
    define_method(attribute) { instance_eval("@#{attribute}")
    define_method("#{attribute}=") do |value|
      instance_eval("@#{attribute} = #{value}")
    end
  end
end
```

Similar methods - part 1

```
class IamAnAdaptor
  def initialize
    @adaptee = SomeClass.new
    @adaptee.methods.select { |m| m =~ /work.*/}.each do |m|
      define_method "sc_#{m}" do
        puts 'Work it!'
        @adaptee.send(m)
      end
    end
  end
end
```

Similar methods - part 2

```
class IamAnAdaptor
  def initialize
    @adaptee = SomeClass.new
  end

  def method_missing(m, *args)
    md = m.to_s.match(/sc_(work.*)/)
    if md && @adaptee.methods.include?(md[1].to_sym)
      puts 'Work it!'
      @adaptee.send(md[1])
    else
      super
    end
  end
end
```

Similar methods - part 3

```
class IamAnAdaptor
  def initialize
    @adaptee = SomeClass.new
  end

  def method_missing(m, *args)
    md = m.to_s.match(/sc_(work.*)/)
    if md && @adaptee.methods.include?(md[1].to_sym)
      self.class.class_eval do
        define_method m do
          puts 'Work it!'
          @adaptee.send(md[1])
        end
      end
      self.send(m)
    else
      super
    end
  end
end
```

Real world

Active Record 4.2.1, associations/builder/association.rb

```
def self.define_readers(mixin, name)
  mixin.class_eval <<-CODE, __FILE__, __LINE__ + 1
    def #{name}(*args)
      association(:#{name}).reader(*args)
    end
  CODE
end

def self.define_writers(mixin, name)
  mixin.class_eval <<-CODE, __FILE__, __LINE__ + 1
    def #{name}=(value)
      association(:#{name}).writer(value)
    end
  CODE
end
```

Real world

rspec core 3.2, memoized_helpers.rb

```
def let(name, &block)
  # We have to pass the block directly to `define_method` to
  # allow it to use method constructs like `super` and `return`.
  raise "#let or #subject called without a block" if block.nil?
  MemoizedHelpers.module_for(self)
    .__send__(:define_method, name, &block)

  # Apply the memoization. The method has been defined in an ancestor
  # module so we can use `super` here to get the value.
  if block.arity == 1
    define_method(name) { __memoized.fetch(name) { |k|
      __memoized[k] = super(RSpec.current_example, &nil) } }
  else
    define_method(name) { __memoized.fetch(name) { |k|
      __memoized[k] = super(&nil) } }
  end
end
```

Recent blogs, more details

Books:

- Programming Ruby, Chapter 24
- MetaProgramming Ruby 2

Blogs / online:

- Ruby learning's metaprogramming
- Sitepoint: - Hook methods (included, ...)
- CodeSchool - 7 deadly sins of metaprogramming

Seven sins?

- Using `method_missing` as your first option
- Not overriding `respond_to_missing?`
- Not handling all cases!
- Using `define_method` when it is not needed(Hmmm...)
- Changing the semantics when opening classes. (e.g., redefining `:+` to add 5 to the result)
- Depending on who is using you (Depend down, not up)
- Deep nesting (e.g., RSpec tests)

Code exercise

https://github.com/drogar/meta-yycruby-code