Reflection and metaprogramming in Ruby

Satyadev Nandakumar

November 7, 2022

Outline

Contents

- Reflection: programs can examine their own structure and modify it
  - enables metaprogramming
- Metaprogramming: writing programs that write programs
  - extends the syntax of Ruby
  - used in writing framework
  - Example: unit testing framework
- Domain-specific languages

1. Reflection: types, classes and modules

Asking the type of an object

- `o.class`
- `o.superclass`
- `o.instance_of? c`
  - whether o.class == c (excludes superclasses)
- `o.is_a? c` or `c === o` or `o.kind_of? c`
  - whether o is an instance of c, or any subclass of c. (for example, `car.is_a? Vehicle` will be true)

Hierarchy

- `C.ancestors` if `C` is a class:
  - chain of superclasses and included modules
- `C < M`: does C include M? or is C a subclass of M?
  - returns `true` if so, `nil` otherwise
- `C.included_modules`

1. Reflection: modifying classes and instances

defining classes

```
D = Class.new { include Comparable; }
```

evaluating arbitrary code : e.g. adding methods

- `instance_eval` and `class_eval`
  - can take code strings or blocks as arguments

```
String.class_eval {def len
                     size
                   end } #instance method
String.instance_eval {def null
                        ''
                      end } #class method
```

- `instance_exec` and `class_exec`
  - can take blocks with arguments

2. Reflection: variables

Querying

```
class Point
  def initialize x, y
    @x,@y = x, y
  end
  def setx x
    @x=x
  end
end

Point.class_variables
p = Point.new 1,1
p.instance_variables # gives [:@x, :@y]
```

2. Reflection: variables

Modifying

```
p.instance_variable_set(:@x,0)
p.instance_variable_get     # gives (0,1)
```

Note: can modify (private) variables!

3. Reflection: methods : listing, defining

query methods of instances

`p.public_~methods`, `p.protected_methods`, `p.private_methods`, `p.singleton_methods` etc.

of classes

`String.instance_methods`, `String.public_instance_methods`, `String.protected_instance_methods`,

defining methods

We can define methods using `class_eval` and `instance_eval`. We can give alternative names using `alias` (important later)

```
Point.class_eval { def norm
                     Math.sqrt @x*@x+@y*@y
                   end
                   alias magnitude norm
}
```

3. Reflection: methods : invoking

- We can call methods by sending messages

invoking methods

```
p.send :norm p.send :setx, 3
```

`method_missing`

- The killer feature that allows powerful metaprogramming techniques
- if a method is missing from the instance/class, `method_missing` is called.
- `method_missing` is in the module `Kernel`
- classes in Ruby are open
- Hence we can redefine `method_missing` for any class!
  - perhaps add the required method at runtime!

Application 1: add accessor methods at runtime!

See example `./ruby_code/runtime_accessor.rb`

Application 2: unit testing

See notes

introduction to domain-specific languages