Zaawansowane języki programowania

Refactoring kodu w języku Ruby przy pomocy narzędzia Reek

Adrian Podlawski

Jako cel mojej Refactorigu obrałem projekt Open Source Hanami. Jest to nowoczesny Framework webowy pisany w języku Ruby. Przedstawia się on jako lekki Framework, z dużymi możliwościami. Cechuje go szybki czas odpowiedzi na zapytania oraz dobre mechanizmy Security.

Pracę rozpocząłem od pobrania repozytorium oraz zidentyfikowania ilości plików oraz Code-smells w nich występujących.

Reek wykazał 266 Code-Smells w 231 plikach.

```
root@adrianbuntu:/home/adrian/ruby/hanami# find lib -name '*.rb' | xargs reek -f json | jq .[].docu
entation_link | sort | uniq -c | sort -n
      1 "https://github.com/troessner/reek/blob/v5.2.0/docs/Attribute.md"
      1 "https://github.com/troessner/reek/blob/v5.2.0/docs/Data-Clump.md"
      1 "https://github.com/troessner/reek/blob/v5.2.0/docs/Long-Parameter-List.md"
      1 "https://github.com/troessner/reek/blob/v5.2.0/docs/Too-Many-Instance-Variables.md"
      1 "https://github.com/troessner/reek/blob/v5.2.0/docs/Unused-Parameters.md"
      2 "https://github.com/troessner/reek/blob/v5.2.0/docs/Nested-Iterators.md"
      2 "https://github.com/troessner/reek/blob/v5.2.0/docs/Uncommunicative-Parameter-Name.md"
      4 "https://github.com/troessner/reek/blob/v5.2.0/docs/Repeated-Conditional.md"
      5 "https://github.com/troessner/reek/blob/v5.2.0/docs/Irresponsible-Module.md"
      5 "https://github.com/troessner/reek/blob/v5.2.0/docs/Manual-Dispatch.md"
      5 "https://github.com/troessner/reek/blob/v5.2.0/docs/Too-Many-Constants.md"
      5 "https://github.com/troessner/reek/blob/v5.2.0/docs/Too-Many-Methods.md"
      6 "https://github.com/troessner/reek/blob/v5.2.0/docs/Control-Parameter.md"
     10 "https://github.com/troessner/reek/blob/v5.2.0/docs/Feature-Envy.md"
     16 "https://github.com/troessner/reek/blob/v5.2.0/docs/Nil-Check.md"
     20 "https://github.com/troessner/reek/blob/v5.2.0/docs/Uncommunicative-Variable-Name.md"
     22 "https://github.com/troessner/reek/blob/v5.2.0/docs/Missing-Safe-Method.md"
     23 "https://github.com/troessner/reek/blob/v5.2.0/docs/Instance-Variable-Assumption.md"
     33 "https://github.com/troessner/reek/blob/v5.2.0/docs/Too-Many-Statements.md"
     40 "https://github.com/troessner/reek/blob/v5.2.0/docs/Duplicate-Method-Call.md"
     51 "https://github.com/troessner/reek/blob/v5.2.0/docs/Utility-Function.md"
root@adrianbuntu:/home/adrian/ruby/hanami#
```

Na załączonym obrazku widać ilość poszczególnych Code-Smells. Najczęściej występującym okazał się Utility-Function, czyli metoda, która nie posiada żadnych zależności do stanu instancji.

Skupiłem się na pliku Server.rb i poniżej przedstawię kroki podjęte podczas jego Refactoringu. Reek po przeanalizowaniu pliku Server.rb pokazał następujący wynik:

```
lib/hanami/server.rb -- 7 warnings:
  [34, 36]:DuplicateMethodCall: Hanami::Server#middleware calls 'mw["development"]' 2 times
  [34, 36]:FeatureEnvy: Hanami::Server#middleware refers to 'mw' more than self (maybe move it to a
other class?)
  [32]:TooManyStatements: Hanami::Server#middleware has approx 6 statements
  [33]:UncommunicativeVariableName: Hanami::Server#middleware has the variable name 'e'
  [33]:UncommunicativeVariableName: Hanami::Server#middleware has the variable name 'm'
  [61]:UtilityFunction: Hanami::Server#code_reloading? doesn't depend on instance state (maybe move
it to another class?)
  [55]:UtilityFunction: Hanami::Server#environment doesn't depend on instance state (maybe move it
o another class?)
```

Wykazał 7 zapachów. Za pierwszy mój cel obrałem Utility Function z lini 61. Metoda wywoływała tylko metodę z klasy Hanami i nie robiła żadnej dodatkowej czynności na obiekcie. Przeanalizowałem klasę Hanami i skorzystałem bezpośrednio z jej metody. Efekt Refactoringu znajduje się na zrzucie ekranu po prawej stronie.

```ruby
48    # @api private
49    def setup
50      return unless code_reloading?
51      @app = Shotgun::Loader.new(rackup)
52    end
53
54    # @api private
55    def environment
56      Components['environment']
57    end
58
59    # @since 0.8.0
60    # @api private
61    def code_reloading?
62      Hanami.code_reloading?
63    end
64
65    # @api private
66    def rackup
67      environment.rackup.to_s
68    end
69
70    # @api private
71    def preload
72      if code_reloading?
73        Shotgun.enable_copy_on_write
74        Shotgun.preload
75      else
76        Hanami.boot
77      end
78    end
```

```ruby
47
48    # @api private
49    def setup
50      return unless Hanami.code_reloading?
51      @app = Shotgun::Loader.new(rackup)
52    end
53
54    # @api private
55    def environment
56      Components['environment']
57    end
58
59    # @api private
60    def rackup
61      environment.rackup.to_s
62    end
63
64    # @api private
65    def preload
66      if Hanami.code_reloading?
67        Shotgun.enable_copy_on_write
68        Shotgun.preload
69      else
70        Hanami.boot
71      end
72    end
73
```

Idąc tropem Utility Functions zająłem się także linią 55. Możemy ją również zobaczyć na powyższych zrzutach ekranu. W tym przypadku przeniosłem metodę do klasy Hanami, ponieważ zawierała ona dużo podobnych metod. Wszystkie były utworzone w identyczny sposób.

```
68
69      # Options for Rack::Server superclass
70      #
71      # @since 0.8.0
72      # @api private
73      def _extract_options
74        Hanami.environment_component.to_options.merge(
75          config:      rackup,
76          Host:        Hanami.environment_component.host,
77          Port:        Hanami.environment_component.port,
78          AccessLog:   []
79        )
80      end
81    end
82  end
83
```

Po wykonaniu powyższych operacji ponownie zapytałem Reeka co o tym sądzi.

```
server.rb -- 6 warnings:
  [64, 66, 67]:DuplicateMethodCall: Hanami::Server#_extract_options calls 'Hanami.environment' 3 tim
es
  [34, 36]:DuplicateMethodCall: Hanami::Server#middleware calls 'mw["development"]' 2 times
  [34, 36]:FeatureEnvy: Hanami::Server#middleware refers to 'mw' more than self (maybe move it to an
other class?)
  [32]:TooManyStatements: Hanami::Server#middleware has approx 6 statements
  [33]:UncommunicativeVariableName: Hanami::Server#middleware has the variable name 'e'
  [33]:UncommunicativeVariableName: Hanami::Server#middleware has the variable name 'm'
adrian@adrianbuntu:~/ruby/hanami/lib/hanami$ _
```

Potwierdził, że pozbyłem się pierwszego Code-Smella. Jednak w drugim przypadku nasza Utility Function zamieniłą się w zapach Duplicate Method Call.

Następnym krokiem było uporanie się z metodą Middleware. Dany fragment zawierał aż cztery różne Code Smelle: Duplicate Method Call, Feature Envy, Too Many statements, Uncommunicative Variable Name.

Kod przed Refactorem wyglądał następująco:

```
32      def middleware
33        mw = Hash.new { |e, m| e[m] = [] }
34        mw["development"].concat([::Rack::ShowExceptions, ::Rack::Lint])
35        require 'hanami/assets/static'
36        mw["development"].push(::Hanami::Assets::Static)
37        mw
38      end
```

Wyodrębniłem metodę, która definiowała nam parametr development oraz tworzyła Hash mapę. Następnie dokonałem nazwy zmiennych e, oraz m tak, aby były zrozumiałe dla wszystkich programistów czytających kod.

```
    def middleware
      middleware_development.concat([::Rack::ShowExceptions, ::Rack::Lint])
      require 'hanami/assets/static'
      middleware_development.push(::Hanami::Assets::Static)
      mw
    end

    def middleware_development
      mw = Hash.new { |environment, middleware| environment[middleware] = [] }
      mw["development"]
    end
```

Ponownie poprosiłem Reeka o ocenę:

```
server.rb -- 1 warning:
  [52, 54, 55]:DuplicateMethodCall: Hanami::Server#_extract_options calls 'Hanami.environment' 3 tim
es
adrian@adrianbuntu:~/ruby/hanami/lib/hanami$ _
```

Pozostał tylko 1 Code-Smell: Duplicate Method Call, który powstał po uporaniu się z zapachem Utility Function.

Niestety w tym przypadku z racji niewystarczającej znajomości bibliotek oraz mechanizmów języka Ruby, nie udało mi się całkiem wyeliminować ostatniego zapachu, ponieważ za każdym razem ewoluował on w inny.

Postanowiłem więc jeszcze raz prześledzić i przetestować działanie kodu, który miałem przyjemność Refactorować. Zobaczyłem, że popełniłem jeden zasadniczy błąd. Zmieniłem działanie programu, co nie powinno mieć miejsca przy Refactoringu. Przy wyodrębnianiu metody middleware_development wyodrębniłem również tworzenie Hasha, co skutkuje niekoniecznie problemem podczas testów, jednak zmieniłem tym działanie programu. Obiekt był tworzony dwukrotnie. W tym wypadku ponownie zabrałem się za Refactoring. Ostatecznie wyodrębniłem obie metody do nowej klasy ServerMiddleware, a do metody przekazywałem utworzony Hash jako parametr.

```ruby

module Hanami

    class ServerMiddleware
        def middleware
            mw = Hash.new { |environment, middleware| environment[middleware] = [] }
            middleware_env(mw).concat([::Rack::ShowExceptions, ::Rack::Lint])
            require 'hanami/assets/static'
            middleware_env(mw).push(::Hanami::Assets::Static)
            mw
        end

        def middleware_env(middleware)
            middleware["development"]
        end
    end
end

```

Po tej operacji Reek dalej pokazywał jeden warning, jednak oryginalne działanie programu zostało zachowane.

Ostatecznie udało mi się uporać z 6 zapachami. Poprawiło to czytelność kodu, oraz zredukowało jego długość. Refactoring nie należy jednak do łatwych zadań, często trafić się nam może uparty Smell. Kiedy wydaje nam się, że został zwalczony, wraca pod inną postacią.