relevance

Sleight of Hand for the Ruby Man

Aaron Bedra
Relevance, Inc.
http://thinkrelevance.com

Metaprogramming is made possible by one simple principal

CODE IS DATA

DATA IS CODE

YAML configuration file

```
server: thin
host: localhost
port: 31337

database:
  adapter: postgresql
  database: wicked_awesome_app_development
```

Turing the config into accessor methods

```
  class Configuration
  def self.load(config_file)
    config = Configuration.new
    config.config_hash = parse_yaml(config_file)
    config.parse_config(config_hash)
    config
  end

  def self.parse_yaml(config_file)
    YAML::load_file(config_file)
  end

  def self.parse_config(config_hash)
    mod = Module.new { config.each_pair { |k,v| define_method(k) { v } } }
    self.extend mod
  end
end
```

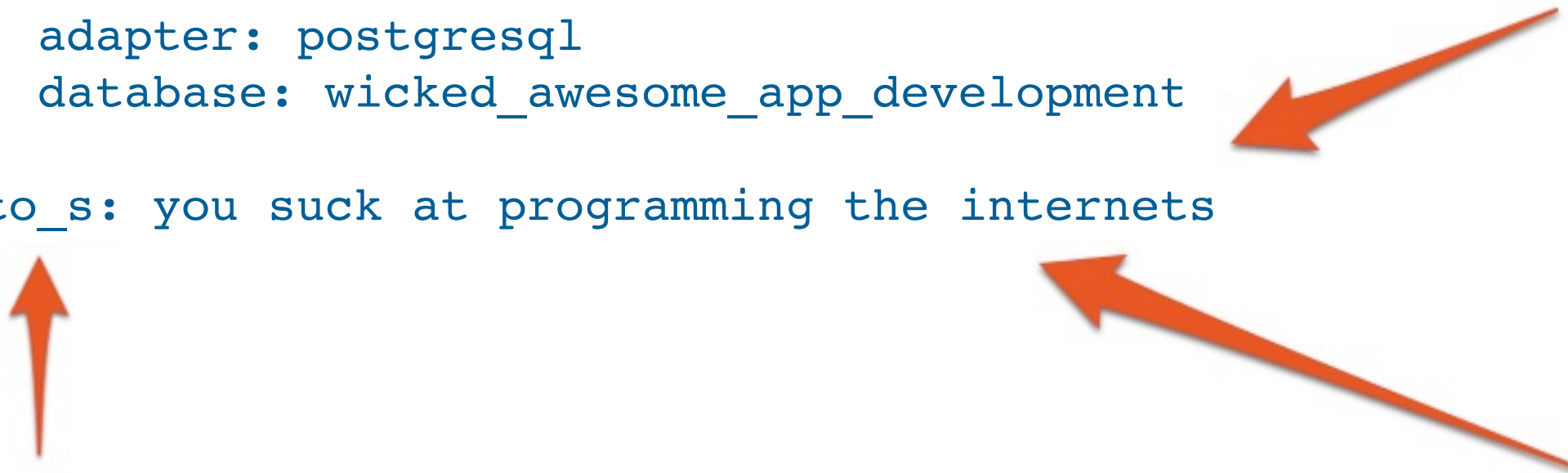
define_method()

- add new methods on the fly
- keep configuration flexible

This trick can fail!

___\|/___/ Failboat example #1 ___\|/___/

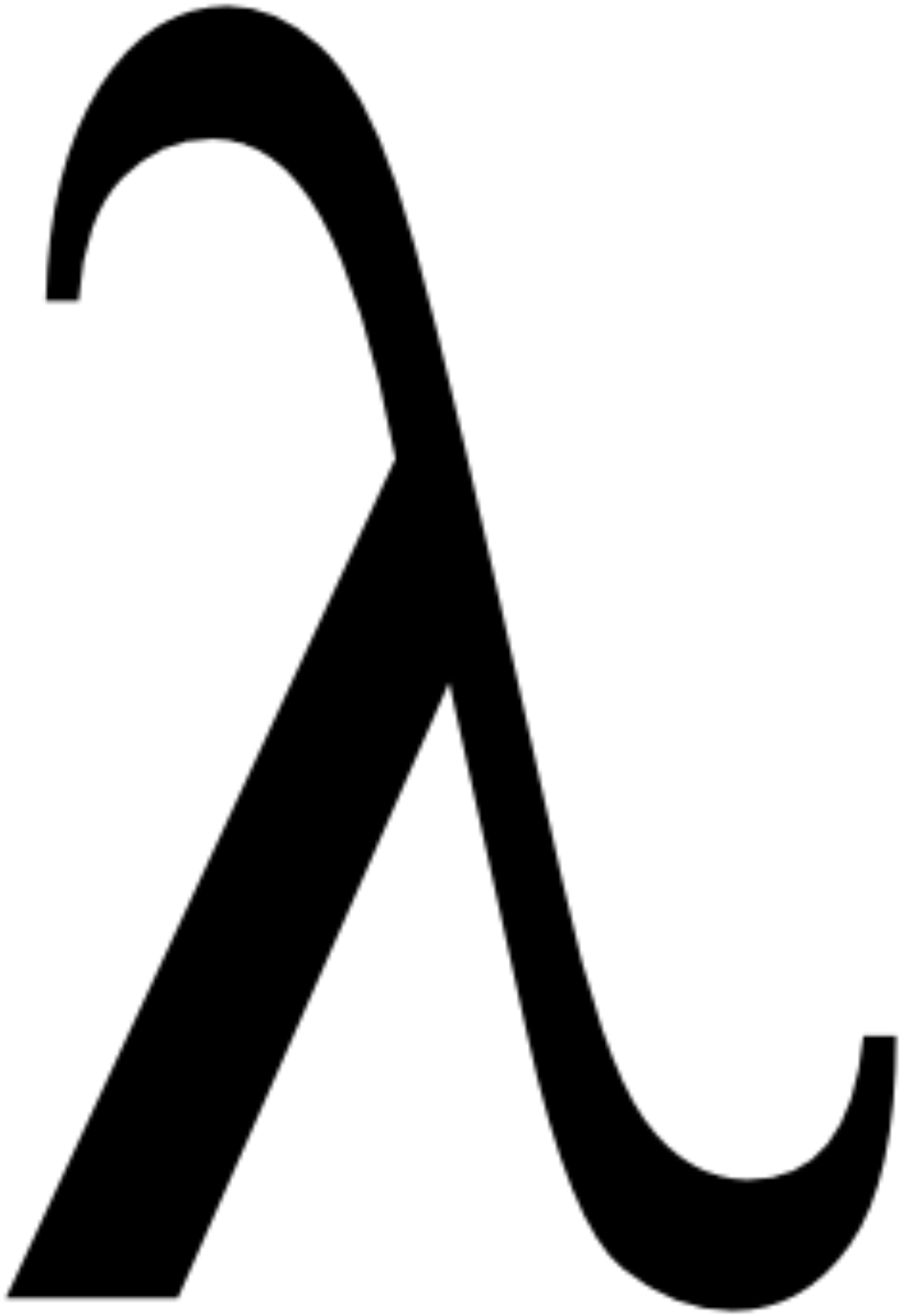
```
server: thin
host: localhost
port: 31337

database:
  adapter: postgresql
  database: wicked_awesome_app_development

to_s: you suck at programming the internets
```




no, its not that bad yet
so far its just raining cats
ICANHASCHEEZBURGER.COM

Let's add a check to the parse_config method

```
 def self.parse_config(config_hash)
  mod = Module.new do
    config.each_pair do |k,v|
      unless self.methods.include?(k)
        define_method(k) { v }
      end
    end
  end
  self.extend mod
end
```



Let's say...

- You are using a lighter weight Ruby framework such as Sinatra or Camping where the "controller" actions are blocks.
- You can't legitimately unit test these actions

We can refactor it, we have the technology

Letting a presenter do the dirty work

```
module Presenters
  class Index
    ... # delegation, ivar accessors, etc.
    def present_it
      @data = lambda {
        controller.erb :login, :locals => {
                                  :presenter => self
                                }
      }
    end
  end
end
```

@data is code

```
 get '/index/' do
  @presenter = Presenters::Index.new(self)
  @presenter.present_it
  @presenter.data.call
end
```

What we've accomplished

- We have successfully pulled all our code into testable places and delivered nice clean code for our "controller".
- Through the use of lambda we can return anything back to the "controller" we want.

Let the inspector find out whodunnit

- The inspector gives you hooks that enable you to find out exactly where a method is defined.
- This is extremely useful when you are overriding default methods in rails plugins.

Imagine if...

- You are using streamlined to develop a quick admin interface for your application.
- You update your code one morning to find that a co-worker has checked in some code that seems to be causing strange errors.
- That co-worker left last Friday for a 3 week vacation in the mountains and can't be reached.
- Your test suite doesn't expose any problems.

The stacktrace says that the offending method is Streamlined.ui_for

```
yourapp> ./script/console
>> require ‘inspector’
=> [“Inspector”]
>> Inspector.where_is_this_defined {Streamlined.ui_for
(:foo)}
=> "Streamlined received message 'ui_for', Line #6 of
yourapp/lib/extensions/streamlined/streamlined.rb"
```

Go from this

To this!

Links

- github.com/spicycode/the-inspector/tree/master
- mwrc2008.confreaks.com/03bowkett.html
- www.pragprog.com/screencasts/v-dtrubyom/the-ruby-object-model-and-metaprogramming
- aaronbedra.com
- workingwithrails.com/person/5499-aaron-bedra

relevance

Questions?