Playing it Safe

Tips for Safe Programming Practices for Ruby

Jim Weirich
Chief Scientist
EdgeCase LLC

EdgeCase
software artisans

Contact:

jim.weirich@gmail.com
http://onestepback.org
@jimweirich

Gino Tesei Asks ...

Are Ruby's Open Classes a Poor Fit for Large Projects?


```
class Folder
  # ...
  def list
    puts @name+":"
    for i in 0...@files.length
        puts "--"+@files[i]
    end
  end
  # ...
end
```

```
#base semantics
aFolder = Folder.new("reports")
aFolder.list
puts "here we're ..."
```

```
reports:
here we're ...
```



```
class Folder
  # ...
  def list
    puts @name+":"
    if @files.length == 0
      raise "empty folder!"
    end
    for i in 0...@files.length
        puts "--"+@files[i]
    end
  end
  # ...
end
```

```
#base semantics
aFolder = Folder.new("reports")
aFolder.list
puts "here we're ..."
```

```
reports:
folder2.rb:14:in `list': empty folder!
(RuntimeError)
  from folder2.rb:25
```


Zed Shaw Rants ...

The Chainsaw Infanticide Logger Maneuver

"One of the things that's really great about agile languages is they give you the power to do anything. ..."

"... One of the most horrible things about agile languages is they give every other idiot the same power to stab you in the back with a rusty pitchfork."

"Not only does this code re-open a class to do something that could **just as easily be done with subclassing**, ..."

"...but the author also TURNS OFF DOCUMENTATION with :nodoc: so that nobody knows about it. ..."

"...This is more like Principle Of Most Heinous Arsenic Injection."

"I then trolled through a few more projects and found that this is common practice."

```
class Logger

  SIMPLE_FORMAT = "%5s: %s\n"
  @@global_logger = Logger.new(STDERR)

  def self.set(logger)
    @@global_logger = logger
  end

  def self.get
    @@global_logger
  end

  def self.debug(str)
    @@global_logger.debug(str)
  end
  #...
end
```



```
Logger.warn "Munged Logger"
```

```
Log = Logger.new(STDOUT)
...
Log.warn "Standard Logger"
```



```
#--
# THINK: some people think,
# this extension is dangerous,
# investigate.
#++
```

Does it really matter?

Application
vs
Library

A Metaphor

“At least I have chicken!”

Tips for Playing it Safe

How to Avoid Being a Leeroy Jenkins

Guard against Murphy,
not Machiavelli

-- Damian Conway

Applications
vs
Libraries

Tip

Use NameSpaces

How Many?

```
class Node
   ...
end
```

IRB::Node
REXML::Light::Node
Tk::BLT::Tree::Node
Tk::BLT::Treeview::Node
Tk::BWidget::Tree::Node
NQXML::Node
YAML::Syck::Node
HTML::Node
DispatcherTest::Node
Node

IRB::Node
REXML::Light::Node
Tk::BLT::Tree::Node
Tk::BLT::Treeview::Node
Tk::BWidget::Tree::Node
NQXML::Node
YAML::Syck::Node
HTML::Node
DispatcherTest::Node
Node

```
IRB::Node
REXML::Light::Node
Tk::BLT::Tree::Node
Tk::BLT::Treeview::Node
Tk::BWidget::Tree::Node
NQXML::Node
YAML::Syck::Node
HTML::Node
DispatcherTest::Node
Node
```


Corollary

Choose Project Name Carefully

- RubyForge
- gems
- github

Tip

Avoid Top Level Functions and Constants

Rake


```
class Task ... end
class RakeApp ... end
class FileList ... end


def task ... end
```

```
module Rake
  class Task ... end
  class RakeApp ... end
  class FileList ... end
end

def task ... end
```



```
module Rake
  class Task ... end
  class RakeApp ... end
  class FileList ... end
end

def task ... end
```

```
module Rake
  class Task ... end
  class RakeApp ... end
  class FileList ... end
end
FileList = Rake::FileList
def task ... end
```


Tip

Avoid Modifying Existing Classes

Tip

Prefer Adding over Modifying


```
module Kernel
  def rake_dup()
    dup
  end
end

[
  NilClass, FalseClass, TrueClass,
  Fixnum, Symbol
].each do |clazz|
  clazz.class_eval {
    def rake_dup() self end
  }
end
```

```
module Kernel
  def rake_dup()
    dup
  end
end

[
  NilClass, FalseClass, TrueClass,
  Fixnum, Symbol
].each do |clazz|
  clazz.class_eval {
    def rake_dup() self end
  }
end
```



```
module Kernel
  def rake_dup()
    dup
  end
end

[
  NilClass, FalseClass, TrueClass,
  Fixnum, Symbol
].each do |clazz|
  clazz.class_eval {
    def rake_dup() self end
  }
end
```

```
module Kernel
  def rake_dup()
    dup
  end
end

[
  NilClass, FalseClass, TrueClass,
  Fixnum, Symbol
].each do |clazz|
  clazz.class_eval {
    def rake_dup() self end
  }
end
```


Tip

When Adding, use project scoped names.

```
fl = FileList[
  "a.pl", "b.pl"
]
fl.ext("rb")
   => ["a.rb", "b.rb"]
```



```
fn = "a.pl"
fn.ext("rb")
   => "a.rb"
```

```
class String
  unless instance_methods.include? "ext"
    def ext(newext='')
      ...
    end
  end
end
```



```
class String
  unless instance_methods.include? "ext"
    def ext(newext='')
      ...
    end
  end
end
```

```
class String
  unless instance_methods.include? "ext"
    def ext(newext='')
      ...
    end
  end
end
```


Tip

When Adding *Public* Methods, ask first.

Selector Namespaces

Replacing Behavior

```
module Rake
  class Task ... end
  class RakeApp ... end
  class FileList ... end
end
FileList = Rake::FileList
def task ... end
```



```
module Rake
  class Task ... end
  class RakeApp ... end
  class FileList ... end
end
FileList = Rake::FileList
def task ... end
if classic_namespace?
  Task = Rake::Task
  RakeApp = Rake::RakeApp
end
```

A Cool Trick


```
class Module
  alias :rake_original_const_missing :const_missing
  def const_missing(const_name)
    case const_name
    when :Task
      Rake.application.const_warning(const_name)
      Rake::Task
    when :FileTask
      Rake.application.const_warning(const_name)
      Rake::FileTask
...
    else
      rake_original_const_missing(const_name)
    end
  end
end
```

```
class Module
  alias :rake_original_const_missing :const_missing
  def const_missing(const_name)
    case const_name
    when :Task
      Rake.application.const_warning(const_name)
      Rake::Task
    when :FileTask
      Rake.application.const_warning(const_name)
      Rake::FileTask
...
    else
      rake_original_const_missing(const_name)
    end
  end
end
```



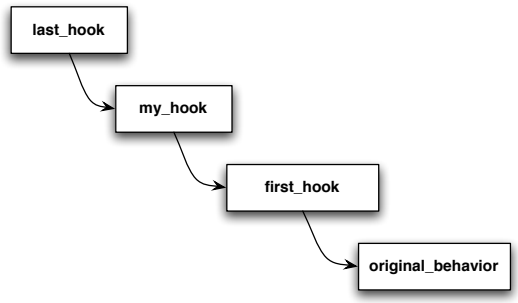
```
class Module
  alias :rake_original_const_missing :const_missing
  def const_missing(const_name)
    case const_name
    when :Task
      Rake.application.const_warning(const_name)
      Rake::Task
    when :FileTask
      Rake.application.const_warning(const_name)
      Rake::FileTask
...
    else
      rake_original_const_missing(const_name)
    end
  end
end
```

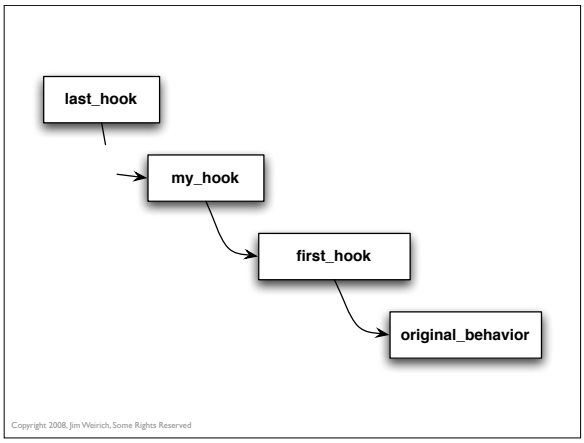
Hook Tip

Chain to the Next Hook


```
class Module
  alias :rake_original_const_missing :const_missing
  def const_missing(const_name)
    case const_name
    when :Task
      Rake.application.const_warning(const_name)
      Rake::Task
    when :FileTask
      Rake.application.const_warning(const_name)
      Rake::FileTask
...
    else
      rake_original_const_missing(const_name)
    end
  end
end
```

```
class Module
  alias :rake_original_const_missing :const_missing
  def const_missing(const_name)
    case const_name
    when :Task
      Rake.application.const_warning(const_name)
      Rake::Task
    when :FileTask
      Rake.application.const_warning(const_name)
      Rake::FileTask
...
    else
      rake_original_const_missing(const_name)
    end
  end
end
```



Hook Tip

Only handle **Your** special cases

```
class Module
  alias :rake_original_const_missing :const_missing
  def const_missing(const_name)
    case const_name
    when :Task
      Rake.application.const_warning(const_name)
      Rake::Task
    when :FileTask
      Rake.application.const_warning(const_name)
      Rake::FileTask
...
    else
      rake_original_const_missing(const_name)
    end
  end
end
```


Tip

Limit the scope of your hook

```
thing = db.fetch_data(key)
thing.do_something unless thing.nil?
```



```
thing = db.fetch_data(key)
thing.do_something unless thing.nil?
```

```
thing = db.fetch_data(key)
thing.do_something
```



```
class NilClass
  def method_missing(sym, *args, &block)
    nil
  end
end
```

```
module Kirby
  class Null
    def method_missing(sym, *args, &block)
      nil
    end
  end
end
```


Whiny Nils

```
class NilClass
  def method_missing(method, *args, &block)
    raise_nil_warning_for(...)
  end
end
```



```
class NilClass
  def raise_nil_warning_for(
      klass = nil,
      selector = nil,
      with_caller = nil)
    message =  "...nil object when you didn't expect it!"
    message << "\n... expected ... #{klass}." if klass
    message << "\n...while ... nil.#{selector}" if selector

    raise NoMethodError, message, with_caller || caller
  end
end
```

Why are Whiny Nils OK?

Require Hooks

- RubyGems
- MockFS
- ActiveSupport (Rails)
- urirequire

```
module Kernel
  alias gem_original_require require
  def require(path)
    gem_original_require path
  rescue LoadError => load_error
    begin
      @gempath_searcher ||= Gem::GemPathSearcher.new
      if spec = @gempath_searcher.find(path)
        Gem.activate(spec.name, ...)
        gem_original_require path
      else
        raise load_error
      end
    end
  end
end
```



```
module Kernel
  alias gem_original_require require
  def require(path)
    gem_original_require path
  rescue LoadError => load_error
    begin
      @gempath_searcher ||= Gem::GemPathSearcher.new
      if spec = @gempath_searcher.find(path)
        Gem.activate(spec.name, ...)
        gem_original_require path
      else
        raise load_error
      end
    end
  end
end
```

```
module Kernel
  alias gem_original_require require
  def require(path)
    gem_original_require path
  rescue LoadError => load_error
    begin
      @gempath_searcher ||= Gem::GemPathSearcher.new
      if spec = @gempath_searcher.find(path)
        Gem.activate(spec.name, ...)
        gem_original_require path
      else
        raise load_error
      end
    end
  end
end
```



```
module Kernel
  alias gem_original_require require
  def require(path)
    gem_original_require path
  rescue LoadError => load_error
    begin
      @gempath_searcher ||= Gem::GemPathSearcher.new
      if spec = @gempath_searcher.find(path)
        Gem.activate(spec.name, ...)
        gem_original_require path
      else
        raise load_error
      end
    end
  end
end
```

```
module Kernel
  alias gem_original_require require
  def require(path)
    gem_original_require path
  rescue LoadError => load_error
    begin
      @gempath_searcher ||= Gem::GemPathSearcher.new
      if spec = @gempath_searcher.find(path)
        Gem.activate(spec.name, ...)
        gem_original_require path
      else
        raise load_error
      end
    end
  end
end
```


But ...

```
$ irb
irb(main):001:0> require 'mylib'
=> true
irb(main):002:0> require 'mylib'
=> false
```



```
$ irb
irb(main):001:0> require 'mylib'
=> false
```

```
module Kernel
  alias gem_original_require require
  def require(path)
    gem_original_require path
  rescue LoadError => load_error
    begin
      @gempath_searcher ||= Gem::GemPathSearcher.new
      if spec = @gempath_searcher.find(path)
        Gem.activate(spec.name, ...)
        gem_original_require path
      else
        raise load_error
      end
    end
  end
end
```


Tip

Preserve the Original Behavior

Tip

Understand and Obey Contracts

Two Parts

What must be true **before** a method is called

What must be true **before** a method is called

precondition

What must be true **after** a method is called

What must be true **after** a method is called

postcondition

Math.sqrt(n)

Precondition

Postcondition

Math.sqrt(n)

Precondition

- `n >= 0`

Postcondition

`Math.sqrt(n)`

Precondition

- `n >= 0`

Postcondition

- `(Result*Result - n).abs < 0.001`

New Behavior Must ...

New Behavior Must ...

- Have the Same or *Weaker* Precondition

`Math.sqrt(n)`

Precondition: **`n >= 0`**

New Behavior Must ...

- Have the Same or *Weaker* Precondition

`Complex.sqrt(n)`

Precondition: **`true`**

New Behavior Must ...

New Behavior Must ...

- Have the Same or *Stronger* Postcondition

`Math.sqrt(n)`

Postcondition: **(result*result - n).abs < 0.001**

New Behavior Must ...

- Have the Same or *Stronger* Postcondition

`ReallyAccurate.sqrt(n)`

Postcondition: **(result*result - n).abs < 0.000001**

Require no more. Promise no less.

require filename

Precondition

file_in_loadpath?(filename)

Postcondition

previously_loaded? => Result == false

! previously_loaded? => Result == true

require filename

Precondition

file_in_loadpath?(filename)

Postcondition

previously_loaded? => Result == false

! previously_loaded? => Result == true

```
class Folder
  # ...
  def list
    puts @name+":"
    for i in 0...@files.length
        puts "--"+@files[i]
    end
  end
  # ...
end
```


Folder#list

(original)

Precondition

true

Postcondition

file_list_is_displayed

Folder#list

(munged)

Precondition

! @list.empty?

Postcondition

file_list_is_displayed

Folder#list

(munged)

Precondition

! @list.empty?

Postcondition

file_list_is_displayed

Stronger Precondition!

Copyright 2008, Jim Weirich, Some Rights Reserved

Wednesday, August 18, 2010 105

`Folder#list`

(munged)

Precondition

! @list.empty?

Postcond

file_list_is_displayed

Stronger Precondition!

Replaced Behavior Must be "DUCK TYPE" Compatible

Contracts
Define
"DUCK TYPE"
Compatible

Summary

"What if someone changes the logic of + for math expressions. ..."

-- Dave Thomas

"... you take them out in the parking lot and beat them with a rubber hose! The language shouldn't prohibit us from doing powerful things."

-- Dave Thomas

- Use Namespaces
- Choose Project Name Carefully
- Avoid Top Level Functions and Constants
- Avoid Modifying Existing Classes
- Prefer Adding over Modifying
- Use Project Scoped Names
- Ask Before Adding Public Methods
- Chain to the Next Hook
- Only Handle Your Special Cases
- Limit the Scope of your Hook
- Understand and Respect Contracts

Summary

Summary

Be Polite with Global Resources

Summary

Be Polite with Global Resources

Respect and Obey Contracts

License