Debuggable Ruby

@ConradIrwin

July 30, 2013

"Beware of bugs in the above code; I have only proved it correct, not tried it."

— Donald Knuth

Dealing with bugs

Java Let's make interfaces simple enough that bugs don't happen.

Dealing with bugs

Java Let's make interfaces simple enough that bugs don't happen.

Haskell Let's make the type-system powerful enough to catch all bugs.

Dealing with bugs

Java	Let's make interfaces simple enough that bugs don't happen.
Haskell	Let's make the type-system powerful enough to catch all bugs.
Ruby	Bugs happen anyway, let's make it really easy to debug.

Debugging process

- Find something that's broken.
- Narrow it down until you know why it's broken.
- Fix the underlying cause.
- Check that it now works.

How people debug

- `puts foo.inspect`
- `raise "zomg wtf bbq" if foo?`
- Run the code in `rails c`
- Read the code...
- `binding.pry`

How people debug

- `p foo`
- `raise "zomg wtf bbq" if foo?`
- Run the code in `rails c`
- Read the code...
- `binding.pry`

How people debug

- `p foo`
- `raise "zomg wtf bbq" if foo?`
- Run the code in `rails c`
- Read the code...
- `binding.pry`

How people debug

- `p foo`
- `raise "zomg wtf bbq" if foo?`
- Run the code in `rails c`
- Read the code...
- `binding.pry`

Debuggable code

- Easy to inspect objects.
- Easy to run short snippets.
- Easy to locate the problem.

`Object#inspect` in Ruby

- Lists all instance variables by default.
- Almost always good enough.
- Override if the default is too noisy.
- Override if you define `#to_s`.

```
Object.instance_method(:inspect).
  bind(Authorization::Base.first).call
```

```
#<Authorization::LinkedIn:0x007f85c23c2748
  @changed_attributes={}, @previously_changed={},
  @marked_for_destruction=false, @destroyed=false,
  @attributes={"id"=>"87", "user_id"=>"38", "type"=>"Authoriza
tion::LinkedIn", "status"=>"pending", "created_at"=>"2013-06-18
23:21:07.138227", "updated_at"=>"2013-06-18 23:21:07.138227",
"account_identifier"=>nil, "properties"=>
   #<struct ActiveRecord::AttributeMethods::Serialization::Attribute
     coder=JSONProperties, value=nil, state=:serialized>,
"encrypted_properties"=>
   #<struct ActiveRecord::AttributeMethods::Serialization::Attribute
    coder=#<EncryptedJSONProperties:0x007f85c23757b8
    @encryptor=#<ActiveSupport::MessageEncryptor:0x007f85c2392200
    @cipher="aes-256-cbc",
    @verifier=#<ActiveSupport::MessageVerifier:0x007f85bbbf9a80
    @serializer=ActiveSupport::MessageEncryptor::NullSerializer,
    @secret="dummysecretdummysecretdummysecretdummysecretdummysecretdummysecretdummysecre",
    @digest="SHA1">,
    @secret="dummysecretdummysecretdummysecretdummysecretdummysecretdummysecretdummysecre",
    @serializer=JSONProperties>>, value=nil, state=:serialized>},
  @readonly=false, @new_record=false, @attributes_cache={}, @relation=nil,
  @aggregation_cache={}, @association_cache={}>
```

```
Authorization::Base.first.inspect

#<Authorization::LinkedIn
  id: 87, user_id: 38,
  type: "Authorization::LinkedIn",
  status: "pending",
  created_at: "2013-06-18 23:21:07",
  updated_at: "2013-06-18 23:21:07",
  account_identifier: nil,
  properties: {},
  encrypted_properties: {}>
```

```
Object.instance_method(:inspect).
  bind(URI.parse("http://google.com/")).call
=> "http://google.com/"
```

```
URI.parse("http://google.com/").inspect
=> "#<URI::HTTP:0x007fcaa
            URL:http://www.google.com/>"
```

Getting the default #inspect back

```ruby
# If you override to_s, inspect will use that.
# This is never what you want.
# Can restore the default with this:
def inspect
  pointer = "0x#{(object_id * 2).to_s(16)}"
  ivars = instance_variables.map do |ivar|
    "#{ivar}=#{instance_variable_get(ivar).inspect}"
  end.join(" ")
  "#<#{self.class.name} #{pointer} #{ivars}>"
end
```

Debuggable code

- Easy to inspect objects.
- Easy to run short snippets.
- Easy to locate the problem.

Debuggable code

- ~~Easy to inspect objects.~~
- Easy to run short snippets.
- Easy to locate the problem.

Example from pry

- Pry lets you edit methods:
  `[1] pry(main)> edit Module#inspect.`
- Bug in the edit command.
- `edit Module.inspect` would sometimes redefine `Module#inspect`.

Hard to debug

- Pry::Commands::Edit “method name”
- — Uses Pry::CodeObject to get a method
- — Invokes MethodPatcher(pry, method)
- — Uses pry.edit to open code in vim
- — Evals the changed source code

Hard to debug

- Pry::Commands::Edit “method name”
- — Uses Pry::CodeObject to get a method
- — Invokes MethodPatcher(pry, method)
- — Uses pry.edit to open code in vim
- — Evals the changed source code

Hard to debug

- Pry::Commands::Edit “method name”
- — Uses Pry::CodeObject to get a method
- — Invokes MethodPatcher(pry, method)
- — Uses pry.edit to open code in vim
- — Evals the changed source code

Minimal test case...

```ruby
module Foo
  def self.foo; :wrong; end
end

binding.pry

# type "edit Foo.foo" into pry...
# type "def foo; :right; end" into vim...

puts Foo.foo == :right
```

Minimal test case...

```ruby
module Foo
  def self.foo; :wrong; end
end

meth = Pry::CodeObject.lookup("Foo.foo")
pry = Pry.new
Commands::Edit::MethodPatcher.new(pry, meth)
# type "def foo; :right; end" into vim...

puts Foo.foo == :right
```

Minimal test case...

```ruby
module Foo
  def self.foo; :wrong; end
end

meth = Pry::CodeObject.lookup("Foo.foo")
Commands::Edit::MethodPatcher.new(meth,
  "def foo; :right; end")

puts Foo.foo == :right
```

Minimal test case...

```ruby
module Foo
  def self.foo; :wrong; end
end

meth = Pry::Method(Foo.method(:foo))
meth.redefine "def foo; :right; end"

puts Foo.foo == :right
```

Debuggable code

- ~~Easy to inspect objects.~~
- Easy to run short snippets.
- Easy to locate the problem.

Debuggable code

- ~~Easy to inspect objects.~~
- ~~Easy to run short snippets.~~
- Easy to locate the problem.

Don't rescue nil

```ruby
# Fuzzily find a command for a user.
# @param [String] search The user's search.
# @return [Pry::Command?]
def find_command_for_help(search)
 find_command(search) ||
 (find_command_by_listing_or_match(search) rescue nil)
end
```

Don't rescue nil

```ruby
# Fuzzily find a command for a user.
# @param [String] search The user's search.
# @return [Pry::Command?]
def find_command_for_help(search)
  find_command(search) || (begin
    find_command_by_match_or_listing(search)
  rescue ArgumentError
    nil
  end)
end
```

When in doubt, raise

```ruby
def do_delivery
  begin
    if perform_deliveries
      delivery_method.deliver!(self)
    end
  # Net::SMTP errors or sendmail pipe errors
  rescue Exception => e
    raise e if raise_delivery_errors
  end
end
```

Preserve __FILE__ and __LINE__

```
def __define_callback(kind, object)
  name = __callback_runner_name(kind)
  unless object.respond_to?(name, true)
    str = object.send("_#{kind}_callbacks").compile
    class_eval <<-RUBY_EVAL, __FILE__, __LINE__ + 1
      def #{name}() #{str} end
      protected :#{name}
    RUBY_EVAL
  end
  name
end
```

Preserve __FILE__ and __LINE__

```
# Create a new rack app from a config.ru
def new_from_string(builder_script, file="config.ru")
  eval "Rack::Builder.new {\n" +
        builder_script + "\n}.to_app",
    TOPLEVEL_BINDING, file, 0
end
```

Good code is debuggable

- Single Responsibility Principle
- Separation of Concerns.
- KISS.

Final words

- Programmers spend 50 – 85% of their time debugging.
- Costs the world $312,000,000 each year.
- A little effort goes a long way.

Thanks

@ConradIrwin