

Designing by Principle

A Case Study: Rack

Principles

Everything should be made as simple as possible, but not simpler.

Principles

Everything should be made as simple as possible, but not simpler.

Definition

“a basic truth, law, or assumption”

Principles

Everything should be made as simple as possible, but not simpler.

Definition

“a basic truth, law, or assumption”

Rule

“don’t use Comic Sans”

Principles

Everything should be made as simple as possible, but not simpler.

Definition

“a basic truth, law, or assumption”

Rule

“don’t use Comic Sans”

Principle

“use a font that works well for your design”

Principles

Everything should be made as simple as possible, but not simpler.

Definition

“a basic truth, law, or assumption”

Rule

“don’t use Comic Sans”

Principle

“use a font that works well for your design”

Where rules are specific principles are general, and therefore have tremendous utility.

Learning to apply them is hard.

Principles

Everything should be made as simple as possible, but not simpler.

Definition

“a basic truth, law, or assumption”

Scientific Method

Keeping clear in mind the outcome we're trying to achieve,
and then considering what principles will help us get there.

Tradeoffs

(everything has a cost)

System design is about balancing *simplicity* with
expressive power

Simplicity	Expressive Power
Reliability, maintainability	Satisfying requirements, flexibility

Rack

Dockerfile
Gemfile
Gemfile.lock
README.md
Rakefile
app/
bin/
config/
config.ru
db/
docker-stack.yml
docs/
extra/
lib/
log/
node_modules/
package.json
public/
spec/
surveyor.iml
tmp/
vendor/
yarn.lock

Rack

```
# This file is used by Rack-based servers to start the application.
```

```
require_relative 'config/environment'
```

```
run Rails.application
```

Rack...Why?

Web Servers / Application Containers



Thin Puma Mongrel
WEBrick Unicorn

Server-side Web Programming Interfaces

CGI FastCGI
SCGI

Web Frameworks



Your web application...

Rack...Why?

Rails 1.0.0...before Rack

```
rails/lib/cgi_handler.rb
```

[illegible]

```
def process(provider: PGID)
  # ... can safely reload this instance.
  nil
end
```

188 lines of code

```
require "dispatcher"
```

rails/lib/webrick_server.rb

```

class DispatchServlet < WEBrick::HTTPServlet::AbstractServlet
  REQUEST_MUTEX = Mutex.new

  # Start the WEBrick server with the given options, mounting the
  # DispatchServlet at <tt>/</tt>.
  def self.dispatch(options = {})
    Socket.do_not_reverse_lookup = true # patch for OS X

    params = { :Port      => options[:port] || 3000,
               :server_type => options[:server_type],
               :bindAddress => options[:ip] }
    params[:MimeTypes] => options[:mime_types] if options[:mime_types]

    server = WEBrick::HTTPServer.new(params)
    server.mount '/' , DispatchServlet, options

    trap("INT") { server.shutdown }

    require File.join(@server_options[:server_root], "...", "config", "environment") unless defined?(RAILS_ROOT)
    require "dispatcher"

    def initialize(server, options) #nodoc:
      @server_options = options
      @file_handler = WEBrick::HTTPServlet::FileHandler.new(server, options[:server_root])
      Dir.chdir(@server_options[:server_root])
      super
    end

    def service(req, res) #nodoc:
      begin
        unless handle_file(req, res)
          REQUEST_MUTEX.lock unless ActionController::Base.allow_concurrency
          unless handle_dispatch(req, res)
            raise WEBrick::HTTPStatus::NotFound, "'#{req.path}' not found."
          end
        end
      ensure
        unless ActionController::Base.allow_concurrency
          REQUEST_MUTEX.unlock if REQUEST_MUTEX.locked?
        end
      end
    end

    def handle_file(req, res) #nodoc:
      begin
        req = req.dup
        path = req.path.dup

        # Add .html if the last path piece has no . in it
        path <= '.html' if path =~ /^\/?(%([^\s\/\?&])+\?)+$/
        path.gsub!('.', '/') # Unescape + since FileHandler doesn't do so.

        req.instance_variable_set(:@path_info, path) # Set the modified path...

        @file_handler.send(:service, req, res)

        return true
      rescue HTTPStatus::PartialContent, HTTPStatus::NotModified => err
        res.set_error(err)
        return true
      rescue => err
        return false
      end
    end

    def handle_dispatch(req, res, origin = nil) #nodoc:
      data = StringIO.new
      Dispatcher.dispatch(
        CGI.new("query", create_env_table(req, origin), StringIO.new(req.body || "")),
        ActionController::CGIRequest::DEFAULT_SESSION_OPTIONS,
        data
      )

      header, body = extract_header_and_body(data)

      set_charset(header)
      assign_status(res, header)
      res.cookies.concat(header.delete('set-cookie') || [])
      header.each { |key, val| res[key] = val.join(", ") }

      res.body = body
      return true
    rescue => err
      p err, err.backtrace
      return false
    end

    private

    def create_env_table(req, origin)
      env = req.meta_vars.clone
      env.delete "SCRIPT_NAME"
      env["QUERY_STRING"] = req.request_uri_query
      env["REQUEST_URI"] = origin if origin
    end

    def extract_header_and_body(data)
      data.rewind
      data = data.read

      raw_header, body = *data.split(/^[\x\d\x\s]+/m, 2)
      header = WEBrick::HTTPUtils::parse_header(raw_header)

      return header, body
    end

    def set_charset(header)
      ct = header['content-type']
      if ct.any? { |x| x =~ /^text\/\// } && ! ct.any? { |x| x =~ /\charset=/ }
        ch = @server_options[:charset] || "UTF-8"
        ct.find { |x| x =~ /^text\/\// } << ("; charset=" + ch)
      end
    end

    def assign_status(res, header)
      if /\A\d+\z/ =~ header['status'] || []
        res.status = $1.to_i
        header.delete('status')
      end
    end
  end
end

```

```
@server_options = options
@file_handler = M::Rich::HTTPServlet::FileHandler.new(server, options)
Dir.chdir(ABSOLUTE_RAISRON)
super
end
```

...after Rack

```
# This file is used by Rack-based servers to start the application.
```

```
require_relative 'config/environment'
```

run Rails.application

```
def handle_dispatch(req, res, origin = nil) $moudoc:
  data = StringIO.new
  Dispatcher.dispatch(
    CGI.new("query").create_env_table(req, origin), StringIO.new(req.body || ""),
    ActionController::CGIRequest::DEFAULT_SESSION_OPTIONS,
    data
  )
end

header, body = extract_header_and_body(data)

set_charset(header)
assign_status(res, header)
res.cookies.concat(header.delete("set-cookie") || [])
header.each { |key, val| res[key] = val.join(", ") }

res.body = body
return true
rescue => err
  p err, err.backtrace
  return false
end

private
def create_env_table(req, origin)
  env = req.meta_vars.clone
  env.delete("SCRIPT_NAME")
  env["QUERY_STRING"] = req.request_uri.query
  env["REQUEST_URI"] = origin if origin
  return env
end

def extract_header_and_body(data)
  data.rewind
  data = data.read

  raw_header, body = data.split(/^(\s*\n)/m, 2)
  header = WEBrick::HTTPUtils::parse_header(raw_header)

  return header, body
end

def set_charset(header)
  ct = header['Content-Type']
  if ct.match? /\[.*?;.*?text\/.*?/ && ! ct.any? { |x| x =~ /charset/ }
    ch = %w(server_options[:charset] || "UTF-8")
    ct.find { |x| x =~ /^text\/.*?/ } << ["; charset=" + ch]
  end
end

def assign_status(res, header)
  if /^(\d+)/ =~ header['status'][:0]
    res.status = $1.to_i
    header.delete('status')
  end
end
end
```


Rack...Why?



Principles

- Well-defined interfaces
(good fences make good neighbors)
- Extensibility *(don't try to predict the future)*
- Composition *(we're better together)*
- Immutability *(respecting boundaries, keeping promises)*

Well-Defined Interface

(good fences make good neighbors)

```
# rack/handler/tomcat.rb
```

```
class Rack::Handler::Tomcat
  def self.run(app, options = {})
    # talk to Tomcat
  end
end
```

```
# rack/handler/apache.rb
```

```
class Rack::Handler::Apache
  def self.run(app, options = {})
    # talk to Apache
  end
end
```

```
# rack/handler/nginx.rb
```

```
class Rack::Handler::Nginx
  def self.run(app, options = {})
    # talk to NGINIX
  end
end
```

```
# config.ru
```

```
run Proc.new { |env| ['200', {'Content-Type' => 'text/html'}, ['Hello neighbor!']] }
```

Extensibility

(don't try to predict the future)

```
# config.ru
```

```
use Rack::CommonLogger  
use Rack::Session::Cookie  
run App
```


Extensibility

(don't try to predict the future)

```
# config.ru
```

```
use Rack::CommonLogger  
use Rack::Session::Cookie  
run App
```

We want to make systems that can easily be extended without modifying its source code.

- **Web API**
- **Service Objects**
- **Adapter Pattern**
- **Blocks / Procs / Closures**

Composition

(we're better together)

```
# config.ru
```

```
use Rack::CommonLogger
use Rack::Session::Cookie
run App
```

```
# config.ru
```

```
App = Rack::CommonLogger.new(
  Rack::Session::Cookie.new(MyApp.new))

run App
```

Composition

(we're better together)

```
# config.ru
```

```
use Rack::CommonLogger
use Rack::Session::Cookie
run App
```

```
# config.ru
```

```
App = Rack::CommonLogger.new(
  Rack::Session::Cookie.new(MyApp.new))
```

```
run App
```

Composable systems are extensible systems

Immutability

(respecting boundaries, keeping promises)

```
class MyApp
  def call(env)
    ['200', {'Content-type' => 'text/html'}, ["This is true"]]
  end
end
```

What's missing?

Immutability

(respecting boundaries, keeping promises)

```
class MyApp
  def call(env)
    ['200', {'Content-type' => 'text/html'}, ["This is true"]]
  end
end
```

```
class Logger
  def initialize(app)
    @app = app
  end
```

Why does this work?

```
    def call(env)
      log(env)
      app.call(env)
    end
  end
end
```

References

- Inventing on Principle - Bret Victor
<https://vimeo.com/36579366>
- Simplicity Matters - Rich Hickey
<https://www.youtube.com/watch?v=rI8tNMsozo0>
- The Mess We're In - Joe Armstrong
<https://www.youtube.com/watch?v=IKXe3HUG2I4>