

# 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*



# 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

```
require 'fcgi'
require 'dispatcher'
require 'showconfig'

class RailsFcgiHandler < Dispatcher
  attr_accessor :log_file_path
  attr_accessor :request_timeout
  attr_accessor :request_period

  # Initialize and run the FastCGI instance, passing arguments through to new.
  def self.procexec(argv, options)
    new(argv, options).start
  end

  # Initialize the FastCGI instance with the path to a crash log.
  # detailing unhandled exceptions (default RAILS_ROOT/log/fastcgi.crash.log)
  # and a request timeout (default 10 seconds).
  # (default nil for normal GC behavior.) Optionally, pass a block which
  # surrounds the main loop. This is useful for testing.
  def initialize(log_file_path = nil, gc_request_period = nil)
    super()
    @log_file_path = log_file_path || RAILS_ROOT + 'log/fastcgi.crash.log'
    self.gc_request_period = gc_request_period
  end

  # Yield for additional configuration.
  yield self if block_given?

  # Safely install signal handlers.
  install_signal_handlers

  # Start the FastCGI instance.
  start
end

def process(listener = RAILS_ROOT + 'log/fastcgi_crash.log')
  # Safely release this instance.
  self.release
end
```

**188 lines of code**

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

# Start the WEBrick server with the given options, mounting the
# DispatchServlet at /dispatch/v1/tbs
def self.dispatch(options={})
  self.new.dispatch(options)
end

Socket.do_not_reverse_lookup = true # patch for OS X

params = { :port => options[:port].to_i,
           :server_type => options[:server_type],
           :bind_address => options[:ip] }

params[:mime_types] = options[:mime_types] if options[:mime_types]

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

trap("INT") { server.shutdown }

require File.join(Rails.root, "...", "config", "environment") unless defined?(RAILS_ROOT)
require "dispatcher"

```

# rails/lib/webrick\_server.rb

```
def initialize(server, options) #nodoc:
```

## **rails/lib/webrick\_server.rb**

**Almost 400 lines of code just to  
use 2 different handlers!**

## ...after Rack

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

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

## run Rails.application

170 lines of code

```
when create
  class Connection(cgi)
    restart
    new
    close_connection(cgi)
    break
  end

  gc_countdown
end

GC::enable
dispatcher_log.info, "(terminated gracefully)"

rescue SystemExit => #exit_error
  dispatcher_log.info, "terminated by explicit exit"

rescue Object => cgi_error
  # retry on errors that would otherwise have terminated the CGI process,
  # so they can occur after the connection has opened
  if ($@isinstance($!, CGIError)) {
    $@.log_error("CGIError in Thread-#{$@.thread_id} on line >@<")
    $@.error_on = true
    new
    dispatcher_error($@.error, "about killed by this error")
    retry
  }
  else
    dispatcher_error($@.error, "killed by this error")
  end
end

private
def logger
  @logger ||= Logger.new(log_file_path)
end

def dispatcher_error(path, msg)
  log_file = File.open(path, "a+")
  log_file.write("-----\n")
  log_file.write("----- Failed to create: (#{$@.class})\n")
  log_file.write("----- #{$@.backtrace.join("\n")}\n")
  log_file.close
  dispatcher_log.error, msg
end

def install_signal_handlers
  Signal::on(SIGALRM) do |signal, handler_name|
    install_signal_handler(signal, method("#{handler_name}_handle"), :to_proc)
  end
end

def install_signal_handler(signal, handler)
  trap(signal, handler)
  rescue ArgumentError
    dispatcher_log.warn, "Ignoring unsupported signal #{$signal}"
  end

  def exit_new_handler(sign)
    dispatcher_log.info, "asked to terminate immediately"
    exit
  end

  def exit_handler(sign)
    dispatcher_log.info, "asked to terminate ASAP"
    $been_ready = exit
  end

  def reload_handler(sign)
    dispatcher_log.info, "asked to reload ASAP"
    $been_ready = reload
  end

  def restart_handler(sign)
    dispatcher_log.info, "asked to restart"
    $been_ready = restart
  end

  def process_request(sign)
    Dispatcher.dispatch(cgi)
    rescue Errno::EMFILE
      raise if $@.class == Connection
      dispatcher_error($@)
    end
  end

  def restart
    config = ::Config::CONFIG
    ruby = File::join(config[:bindir], config[:ruby_install_name]) + config[:PREFIX]
    command_line = ["nohup", "bin/ruby", "-e", "require 'script'; $0=$0"]
    dispatcher_log.info, "restarting"
    exec(*command_line)
  end

  def reload
    run_get if $@.request_permit
    rescue Errno::EMFILE
      $been_ready = nil
      dispatcher_log.info, "reloaded"
    end
  end

  def mask!
    $features = '$:close'
    end

  def restart?
    $features & Features::RESTART_APP
    Dispatcher.reset_application
    ActionController::Base.reload
    end

  def run_get?
    $@.request_permit && $@.request_permit <= GC.request_timeout
    GC.start; GC.disable
    end

  def gc_countdown
    if $@.request_permit
      $@.request_countdown -= 1
      run_get? if $@.request_countdown <= 0
    end
  end

  def close_connection(cgi)
    cgi.instance_variable_get("Drequest").finish
  end

  def handle_file(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 !~ /\.(?!(\?|&|&=|&=))/i
      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
    rescue WEBrick::HTTPStatus::PartialContent, WEBrick::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
    return false
  end

  private
  def create_env_table(req, origin)
    env = Hash::new
    env["REDIRECT_VARS"] = clone
    env["REMOTE_USER"] = 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(/\r\n\r\n/, 2)
    header = WEBrick::HTTPUtils::parse_header(raw_header)

    return header, body
  end

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

  def assign_status(res, header)
    if !("200" <= header["status"])[0]
      res.status = $http_status[header["status"]]
      header.delete("status")
    end
  end

  def assign_status(res, header)
    if !("200" <= header["status"])[0]
      res.status = $http_status[header["status"]]
      header.delete("status")
    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                                # rack/handler/apache.rb
class Rack::Handler::Tomcat                         class Rack::Handler::Apache
  def self.run(app, options = {})
    # talk to Tomcat
  end
end                                              end

# rack/handler/nginix.rb
class Rack::Handler::Nginix
  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

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

Why does this work?

# 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>