Ruby Application Basics

from script ☞ to service

Tim Pease

- https://github.com/TwP
- @pea53

Enigma Project

enigma.rb

```ruby
#!/usr/bin/env ruby

require 'base64'

input = ARGV[0] ? File.open(ARGV[0],'r') : STDIN
decoded = Base64.decode64(input.read)
STDOUT.write decoded
```

Enigma Service

- Run as a daemon
- Read messages from a queue
- Log the message processing events
- Configurable
- Command line options

Separation of Concerns

separate the application from the command line

Servolux

- Toolkit for creating servers and managing daemons
- https://github.com/TwP/servolux
- Servolux::Server
- Servolux::Daemon

file tree

```
.
├── bin
│   └── enigma
├── config
└── lib
    ├── enigma
    │   └── app.rb
    └── enigma.rb
```

lib/enigma/app.rb

```ruby
module Enigma
  class App < Servolux::Server

    def initialize( )
      super('enigma',
          :logger   => Logger.new('log/enigma.log'),
          :pid_file => 'log/enigma.pid'
      )
      @job = nil
      @beanstalk = nil
      self.continue_on_error = true
    end

    def before_starting ...
    def before_stopping ...
    def run ...
  end
end
```

lib/enigma/app.rb

```ruby
def before_starting
  @beanstalk = Beanstalk::Pool.new(['localhost:11300'])
end

def before_stopping
  return unless @beanstalk
  @beanstalk, beanstalk = nil, @beanstalk

  beanstalk.close if @job.nil?
  Thread.pass  # allow the server thread to wind down
end
```

lib/enigma/app.rb

```ruby
def run
  return unless @beanstalk
  @job = @beanstalk.reserve 30 rescue nil
  if @job
    logger.info "Processing '#{@job.id}'"
    decoded = Base64.decode64(@job.body)
    File.open("decoded/#{@job.id}.txt",'w') { |fd|
      fd.write decoded
    }
  end

rescue Beanstalk::TimedOut
rescue StandardError => err
  logger.info "Error while processing job '#{@job.id}'"
  logger.error err
ensure
  @job.delete rescue nil if @job
  @job = nil
end
```

bin/enigma

```
#!/usr/bin/env ruby

root = File.expand_path('../..', __FILE__)
require File.join(root, %w[lib enigma])

server = Enigma::App.new
daemon = Servolux::Daemon.new(
  :server => server,
  :timeout => 60,
  :nochdir => true
)
deamon.startup
```

Accomplishments

- Enigma::App that runs our decryption service
- Enigma command line tool that daemonizes the app
- Basic logging of messages

Configuration

authoritative source for system parameters

Loquacious

- Verbose configuration
- https://github.com/TwP/loquacious
- Provides namespaced configuration parameters
- Loquacious.configuration_for
- Loquacious.help_for

Configuration Goals

- provide configuration with sensible defaults so the application works out of the box
- support environment based configuration
  - think of environments as named configuration sets

file tree

```
.
├── bin
│   └── enigma
├── config
│   └── environments
│       ├── development.rb
│       ├── production.rb
│       └── test.rb
└── lib
    ├── enigma
    │   ├── app.rb
    │   ├── config.rb
    │   └── initializer.rb
    └── enigma.rb
```

lib/enigma/config.rb

```
config = Enigma.config {}

Enigma.defaults {
  app_name  $0.dup, :desc => <<-__
    Name of the running program (used for log file naming).
  __

  desc <<-__
    The name and location of the PID file. This file is used
    to output the process ID of an Enigma application when
    started as a daemon.
  __
  pid_file(Proc.new {
    File.join(config.log_path,
              "#{config.app_name}.#{config.environment}.pid")
  })
}
```

config/environments/*

```
config.beanstalk {
  host  'localhost'
  port  11300
}

config.log_level = :debug
```

```
config.beanstalk {
  host  'example.com'
  port  3001
}

config.log_level = :warn
```

Initialization Goals

- Load default configuration
- Load environment specific configuration
- Allow for ad-hoc overriding of configuration
  - command line options

file tree

```
.
├── bin
│   └── enigma
├── config
│   └── environments
│       ├── development.rb
│       ├── production.rb
│       └── test.rb
└── lib
    ├── enigma
    │   ├── app.rb
    │   ├── config.rb
    │   └── initializer.rb
    └── enigma.rb
```

lib/enigma/initializer.rb

```ruby
module Enigma
  def self.setup( &block )
    Enigma::Initializer.run(&block)
  end

  class Initializer
    def self.run( *args, &block )
      new.process(*args, &block)
    end

    def initialize
      @config = Enigma.config
    end

    def process ...
    def load_environment ...
  end
end
```

lib/enigma/initializer.rb

```ruby
def process( *args, &block )
  load_environment
  block.call(@config) unless block.nil?
  @config.initializers.each { |init|
    self.send "initialize_#{init}"
  }
  self
end

def load_environment
  fn = Enigma.config_path('environments',
                          "#{@config.environment}.rb")
  return self unless test(?f, fn)

  config = @config
  eval(IO.read(fn), binding, fn)
  self
end
```

bin/enigma

```ruby
#!/usr/bin/env ruby

root = File.expand_path('../..', __FILE__)
require File.join(root, %w[lib enigma])

Enigma.setup do |config|
  config.app_name = 'enigma'
end

server = Enigma::App.new
daemon = Servolux::Daemon.new(
  :server  => server,
  :timeout => 60,
  :nochdir => true
)
deamon.startup
```

Accomplishments

- Provided a single source configuration system
- That supports environments (configuration sets)
- And can be modified at runtime

Logging

when things go awry

Logging

- Multiple destinations for log events
- https://github.com/TwP/logging
- Logging to stdout, files, email, syslog, stringio, growl, ...
- Per-class log level settings

Logging basics

```ruby
require 'logging'

include Logging.globally
logger.info "We now have a 'logger' method in every Object"

Logging.logger[Enigma]
Logging.logger[self]
Logging.logger['Enigma::App']

Logging.logger.root.level = :debug
Logging.logger.root.appenders = 'stdout'
```

file tree

```
.
├── bin
│   └── enigma
├── config
│   ├── environments
│   │   ├── development.rb
│   │   ├── production.rb
│   │   └── test.rb
│   └── logging.rb
└── lib
    ├── enigma
    │   ├── app.rb
    │   ├── config.rb
    │   └── initializer.rb
    └── enigma.rb
```

lib/enigma/initializer.rb

```ruby
def initialize_logging
  fn = Enigma.config_path('logging.rb')
  return self unless test(?f, fn)

  if @config.log_path and !test(?e, @config.log_path)
    FileUtils.mkdir @config.log_path
  end

  config = @config
  eval(IO.read(fn), binding, fn)

  Logging.show_configuration if Logging.logger[Enigma].debug?
  self
end
```

config/logging.rb

```ruby
Logging.format_as :inspect
layout = Logging.layouts.pattern(
             :pattern => '[%d] %-5l %c : %m\n'
         )

Logging.appenders.stdout(
  'stdout',
  :auto_flushing => true,
  :layout => layout
) if config.log_to.include? 'stdout'

Logging.logger.root.level = config.log_level

unless config.log_to.empty?
  Logging.logger.root.appenders = config.log_to
end
```

config/environments/*

```ruby
config.beanstalk {
  host  'localhost'
  port  11300
}

config.log_level = :debug
config.log_to = %w[stdout]
```

```ruby
config.beanstalk {
  host  'example.com'
  port  11300
}

config.log_level = :warn
config.log_to =
  %w[logfile email]

Logging.
  logger['Enigma::App'].
  level = :info
```

Accomplishments

- Include a flexible logging framework
- That supports multiple log event destinations
- And per-class log levels
- All integrated with our configuration system

Command Line

give me help / offer me a choice

Main

- A class factory and DSL for generating command line programs real quick
- https://github.com/ahoward/main
- Uniform command line parsing and options
- Support for mode / sub-commands

file tree

```
.
├── bin
│   └── enigma
├── config
│   ├── environments
│   │   ├── development.rb
│   │   ├── production.rb
│   │   └── test.rb
│   └── logging.rb
└── lib
    ├── enigma
    │   ├── app.rb
    │   ├── config.rb
    │   └── initializer.rb
    └── enigma.rb
```

bin/enigma

```ruby
#!/usr/bin/env ruby

require 'main'

Main do
  argument 'environment' do
    default 'development'
    cast :symbol
    attribute
  end

  option '--debug' do
    attribute
  end

  def run ...
end
```

bin/enigma

```ruby
Main do
  ...
  def run
    Enigma.config.environment = environment
    Enigma.setup do |config|
      config.app_name = 'enigma'
      config.log_level = :debug if debug?
    end

    server = Enigma::App.new
    daemon = Servolux::Daemon.new(
      :server  => server,
      :timeout => 60,
      :nochdir => true
    )
    deamon.startup
  end
end
```

Accomplishments

- Flexible command line parser
- That injects settings into our configuration system
- And spins up our Enigma server

Summary

- The “Initializer” pulls Enigma’s parts together
- Loquacious gives us a central configuration authority
    - All the parts interact through configuration
- Servolux gives us server & daemon tools
- Logging gives us flexibility to see what is going on
- Main lets us tweak settings at runtime

?

Presentation Access

https://github.com/TwP/enigma