Rails Exploration

2013-03-25

James Zhan

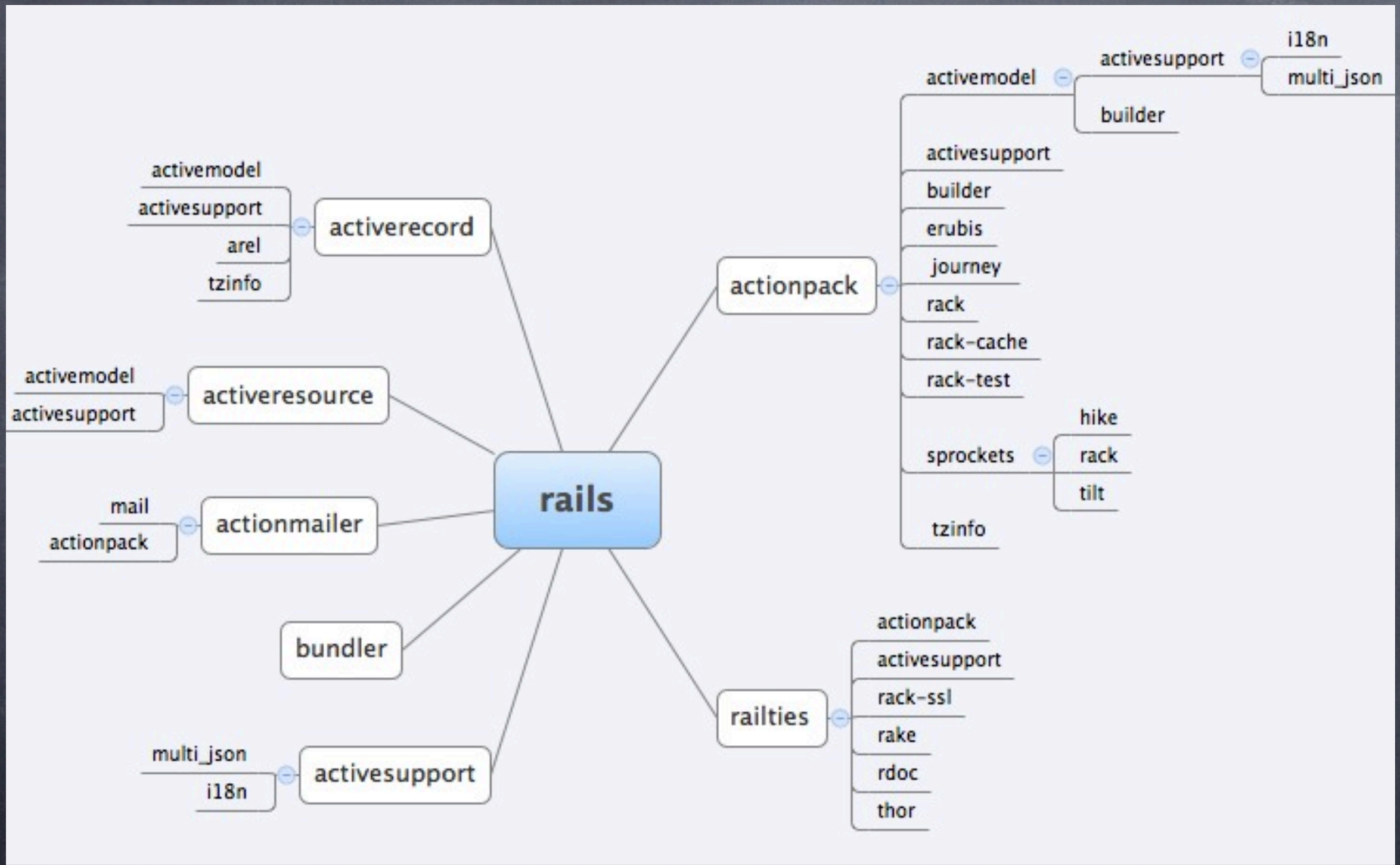
ActiveSupport

```
module ActiveSupport
  class << self
    attr_accessor :load_all_hooks
    def on_load_all(&hook) load_all_hooks << hook end
    def load_all!; load_all_hooks.each { |hook| hook.call } end
  end
  self.load_all_hooks = []

  on_load_all do
    [Dependencies, Deprecation, Gzip, MessageVerifier, Multibyte]
  end
end
```

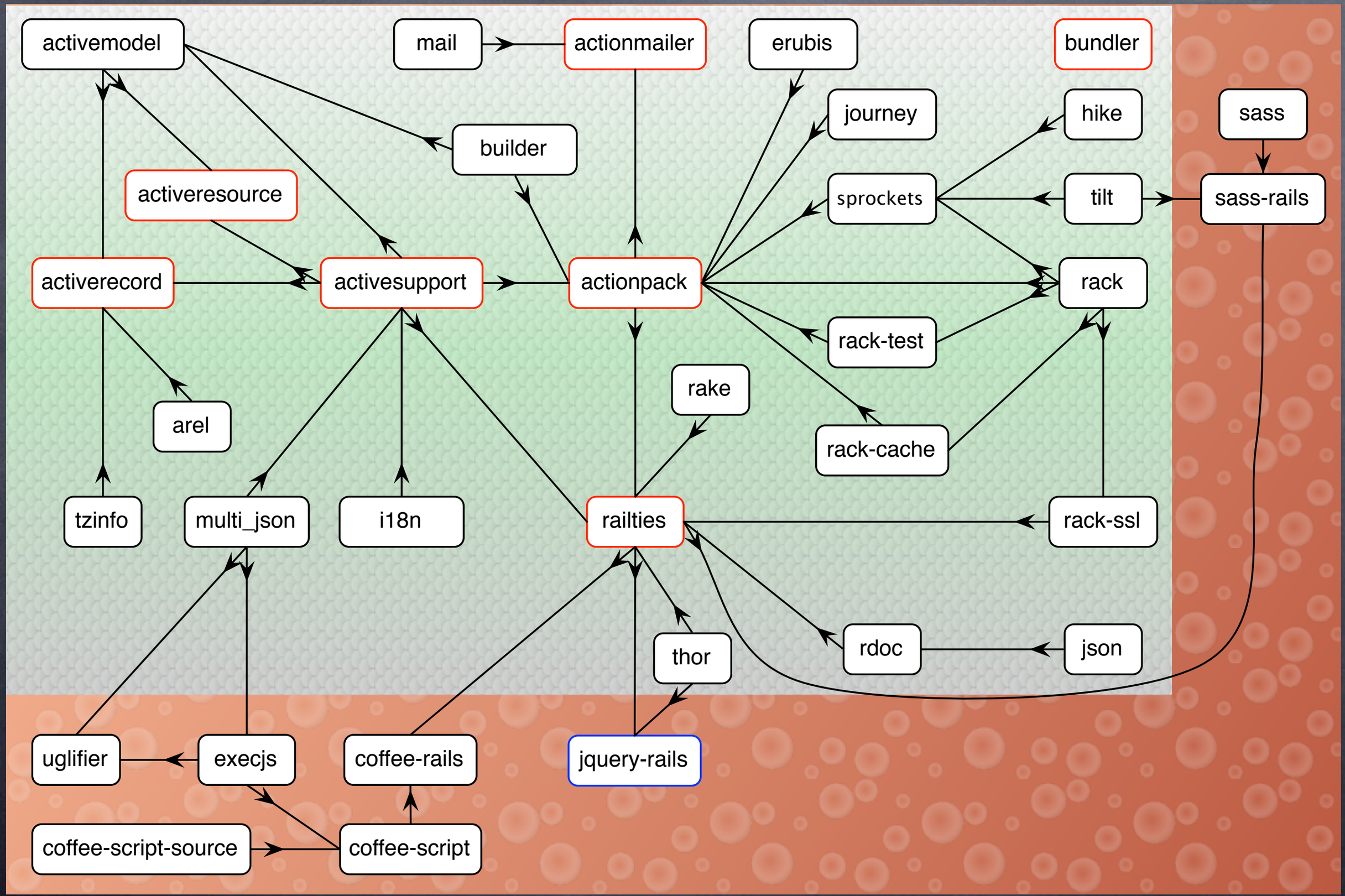
```
module ActiveSupport
  autoload :Duration, 'active_support/duration'
  autoload :TimeWithZone, 'active_support/time_with_zone'
  autoload :TimeZone, 'active_support/values/time_zone'

  on_load_all do
    [Duration, TimeWithZone, TimeZone]
  end
end
```

```
module ActiveSupport
  @load_hooks = Hash.new { |h,k| h[k] = [] }
  @loaded = Hash.new { |h,k| h[k] = [] }

  def self.on_load(name, options = {}, &block)
    @loaded[name].each do |base|
      execute_hook(base, options, block)
    end
    @load_hooks[name] << [block, options]
  end

  def self.execute_hook(base, options, block)
    if options[:yield]
      block.call(base)
    else
      base.instance_eval(&block)
    end
  end

  def self.run_load_hooks(name, base = Object)
    @loaded[name] << base
    @load_hooks[name].each do |hook, options|
      execute_hook(base, options, hook)
    end
  end
end
```

Rails Dependencies



Rails Dependencies



Railties -- Gluing the Engine to the Rails

Railties is responsible for gluing all frameworks together, Overall, it:

- handles the bootstrapping process for a Rails application;
- manages the +rails+ command line interface;
- and provides the Rails generators core.

Railtie is the core of the Rails framework and provides several hooks to extend Rails and/or modify the initialization process.

Every major component of Rails (Action Mailer, Action Controller, Action View, Active Record and Active Resource) is a Railtie. Each of them is responsible for their own initialization. This makes Rails itself absent of any component hooks, allowing other components to be used in place of any of the Rails defaults.

Developing a Rails extension does not require any implementation of Railtie, but if you need to interact with the Rails framework during or after boot, then Railtie is needed.

For example, an extension doing any of the following would require Railtie:

1. creating initializers
2. configuring a Rails framework for the application, like setting a generator
3. adding config.* keys to the environment
4. setting up a subscriber with ActiveSupport::Notifications
5. adding rake tasks

Rails::Engine

Rails::Engine allows you to wrap a specific Rails application or subset of functionality and share it with other applications. Since Rails 3.0, every Rails::Application is just an engine, which allows for simple feature and application sharing.

Active Support -- Utility classes and Ruby extensions from Rails

Active Support is a collection of utility classes and standard library extensions that were found useful for the Rails framework. These additions reside in this package so they can be loaded as needed in Ruby projects outside of Rails.

Action Pack -- From request to response

Action Pack is a framework for handling and responding to web requests. It provides mechanisms for *routing* (mapping request URLs to actions), defining *controllers* that implement actions, and generating responses by rendering *views*, which are templates of various formats. In short, Action Pack provides the view and controller layers in the MVC paradigm.
It consists of several modules:

- Action Dispatch, which parses information about the web request, handles routing as defined by the user, and does advanced processing related to HTTP such as MIME-type negotiation, decoding parameters in POST/PUT bodies, handling HTTP caching logic, cookies and sessions.
- Action Controller, which provides a base controller class that can be subclassed to implement filters and actions to handle requests. The result of an action is typically content generated from views.
- Action View, which handles view template lookup and rendering, and provides view helpers that assist when building HTML forms, Atom feeds and more.
  Template formats that Action View handles are ERB (embedded Ruby, typically used to inline short Ruby snippets inside HTML), and XML Builder.

With the Ruby on Rails framework, users only directly interface with the Action Controller module. Necessary Action Dispatch functionality is activated by default and Action View rendering is implicitly triggered by Action Controller. However, these modules are designed to function on their own and can be used outside of Rails.

Active Model -- model interfaces for Rails

Active Model provides a known set of interfaces for usage in model classes. They allow for Action Pack helpers to interact with non-ActiveRecord models,for example. Active Model also helps building custom ORMs for use outside of the Rails framework.

Prior to Rails 3.0, if a plugin or gem developer wanted to have an object interact with Action Pack helpers, it was required to either copy chunks of code from Rails, or monkey patch entire helpers to make them handle objects that did not exactly conform to the Active Record interface. This would result in code duplication and fragile applications that broke on upgrades.

Active Model solves this. You can include functionality from the following modules:

- ★ Add attribute magic to objects
- ★ Callbacks for certain operations
- ★ Tracking value changes
- ★ Adding +errors+ interface to objects
- ★ Model name introspection
- ★ Observer support
- ★ Making objects serializable
- ★ Internationalization (i18n) support
- ★ Validation support
- ★ Custom validators

Active Resource

Active Resource (ARes) connects business objects and Representational State Transfer (REST) web services. It implements object-relational mapping for REST web services to provide transparent proxying capabilities between a client (ActiveResource) and a RESTful service (which is provided by Simply RESTful routing in ActionController::Resources).

Active Resource attempts to provide a coherent wrapper object-relational mapping for REST web services. It follows the same philosophy as Active Record, in that one of its prime aims is to reduce the amount of code needed to map to these resources. This is made possible by relying on a number of code- and protocol-based conventions that make it easy for Active Resource to infer complex relations and structures. These conventions are outlined in detail in the documentation for ActiveResource::Base.

Model classes are mapped to remote REST resources by Active Resource much the same way Active Record maps model classes to database tables. When a request is made to a remote resource, a REST XML request is generated, transmitted, and the result received and serialized into a usable Ruby object.

Active Resource is built on a standard XML format for requesting and submitting resources over HTTP. It mirrors the RESTful routing built into Action Controller but will also work with any other REST service that properly implements the protocol. REST uses HTTP, but unlike "typical" web applications, it makes use of all the verbs available in the HTTP specification:

* GET requests are used for finding and retrieving resources.
* POST requests are used to create new resources.
* PUT requests are used to update existing resources.
* DELETE requests are used to delete resources.

Project: Builder

Provide a simple way to create XML markup and data structures.

- Builder::XmlMarkup:: Generate XML markup notiation
- Builder::XmlEvents:: Generate XML events (i.e. SAX-like)
- Examples:

```
builder = Builder::XmlMarkup.new(:target=>STDOUT, :indent=>2)
builder.person { |b| b.name("Jim"); b.phone("555-1234") }
```

OUTPUT:

```
# <person>
#   <name>Jim</name>
#   <phone>555-1234</phone>
# </person>
```

Action Mailer -- Easy email delivery and testing

Action Mailer is a framework for designing email-service layers. These layers are used to consolidate code for sending out forgotten passwords, welcome wishes on signup, invoices for billing, and any other use case that requires a written notification to either a person or another system.

Action Mailer is in essence a wrapper around Action Controller and the Mail gem. It provides a way to make emails using templates in the same way that Action Controller renders views using templates.

Additionally, an Action Mailer class can be used to process incoming email, such as allowing a blog to accept new posts from an email (which could even have been sent from a phone).

Active Record -- Object-relational mapping put on rails

> Active Record connects classes to relational database tables to establish an almost zero-configuration persistence layer for applications. The library provides a base class that, when subclassed, sets up a mapping between the new

class and an existing table in the database. In the context of an application, these classes are commonly referred to as *models*. Models can also be connected to other models; this is done by defining *associations*.
Active Record relies heavily on naming in that it uses class and association names to establish mappings between respective database tables and foreign key columns. Although these mappings can be defined explicitly, it's recommended to follow naming conventions, especially when getting started with the library.

A short rundown of some of the major features:

- ★ Automated mapping between classes and tables, attributes and columns.
- ★ Associations between objects defined by simple class methods.
- ★ Aggregations of value objects.
- ★ Validation rules that can differ for new or existing objects.
- ★ Callbacks available for the entire life cycle (instantiation, saving, destroying, validating, etc.).
- ★ Observers that react to changes in a model.
- ★ Inheritance hierarchies.
- ★ Transactions.
- ★ Reflections on columns, associations, and aggregations.
- ★ Database abstraction through simple adapters.
- ★ Logging support for Log4r and Logger.
- ★ Database agnostic schema management with Migrations.

Arel is a SQL AST manager for Ruby

Arel uses ActiveRecord's connection adapters to connect to the various engines, connection pooling, perform quoting, and do type conversion.

- Simplifies the generation of complex SQL queries
- Adapts to various RDBMS systems

Tilt

Tilt is a thin interface over a bunch of different Ruby template engines in an attempt to make their usage as generic possible. This is useful for web frameworks, static site generators, and other systems that support multiple template engines but don't want to code for each of them individually.

The following features are supported for all template engines (assuming the feature is relevant to the engine):

- Custom template evaluation scopes / bindings
- Ability to pass locals to template evaluation
- Support for passing a block to template evaluation for "yield"
- Backtraces with correct filenames and line numbers
- Template file caching and reloading
- Fast, method-based template source compilation

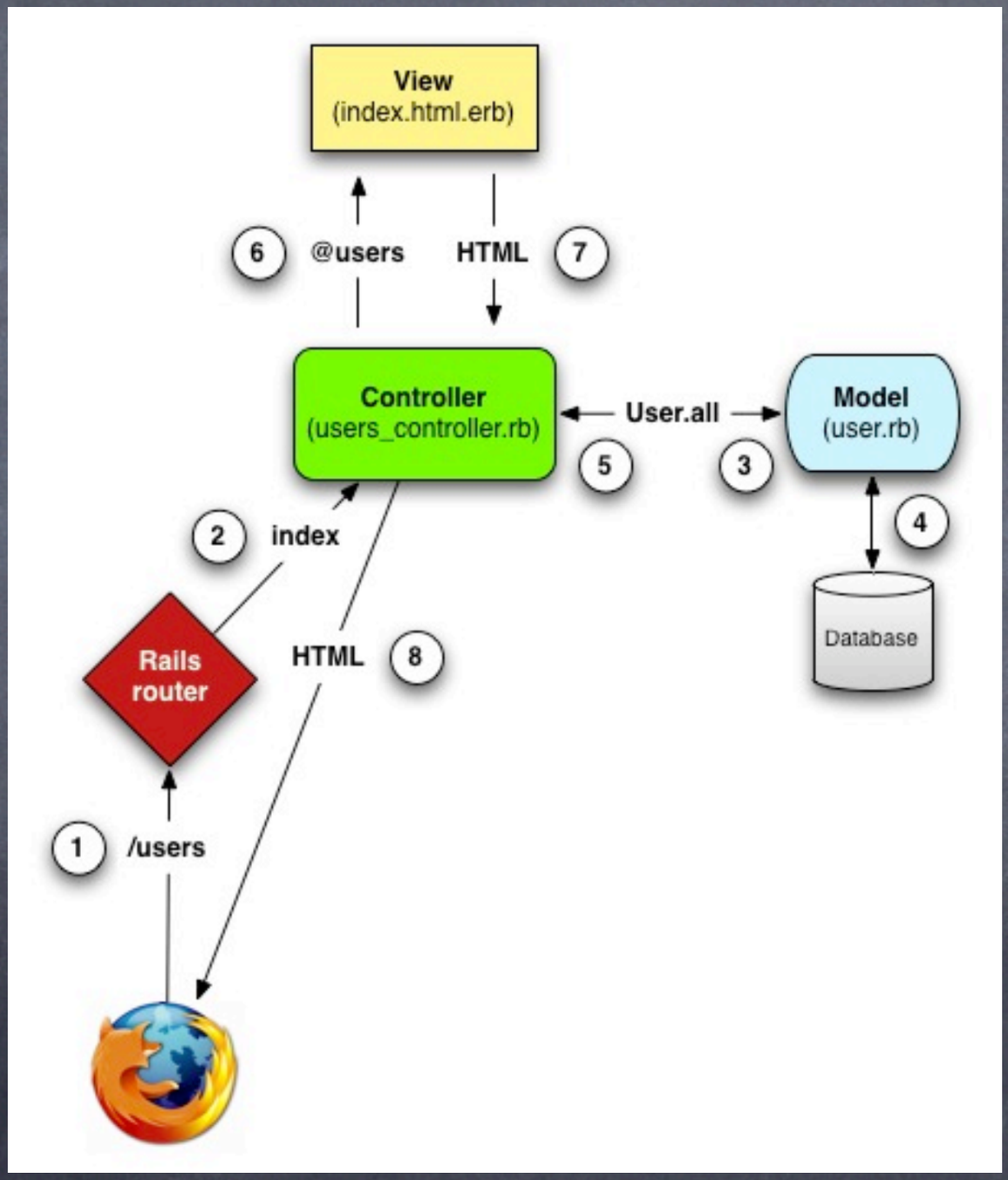
Tilt template engines

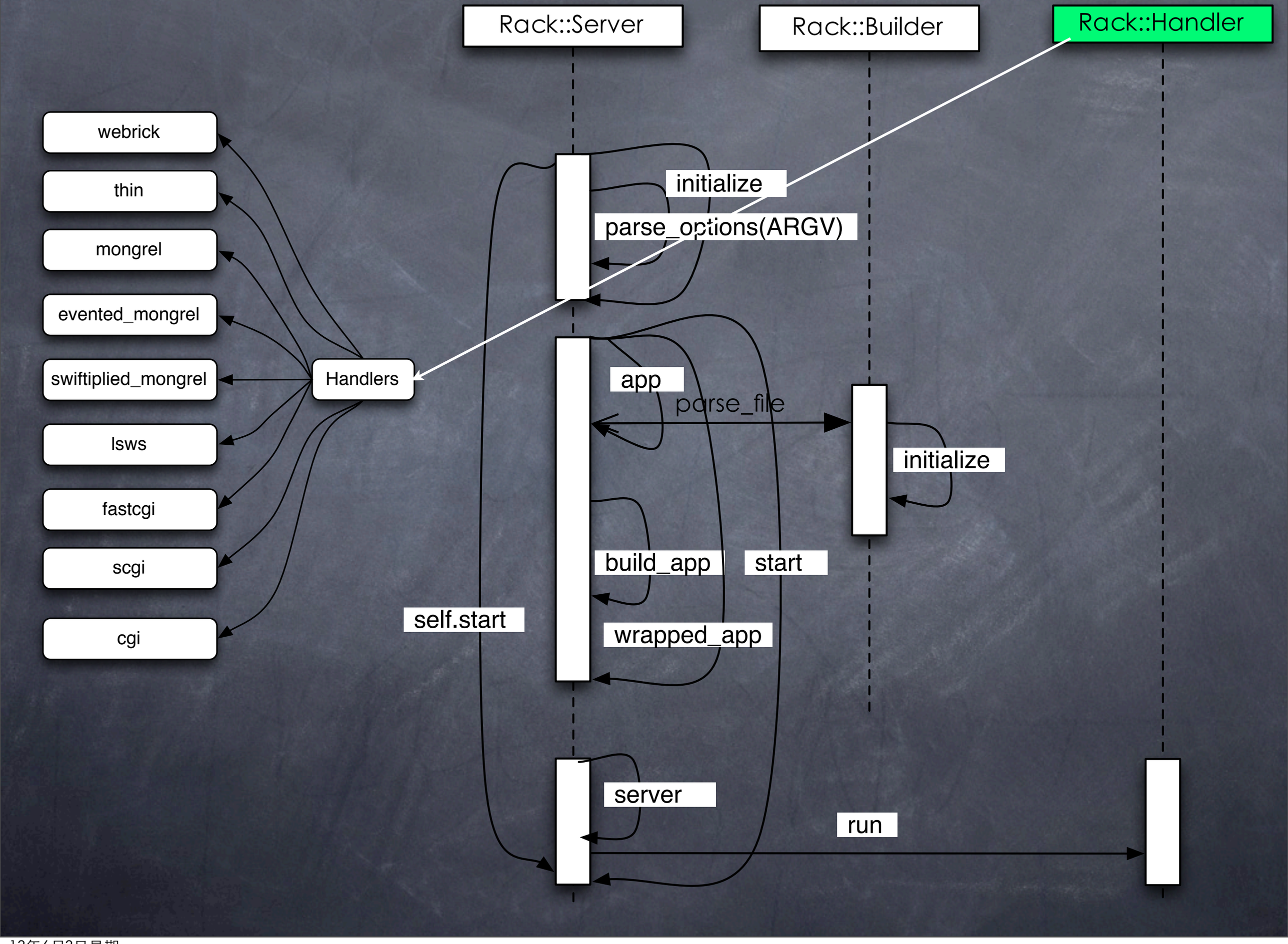
ENGINE	FILE EXTENSIONS	REQUIRED LIBRARIES
ERB	.erb, .rhtml	none (included ruby stdlib)
Interpolated String	.str	none (included ruby core)
Erubis	.erb, .rhtml, .erubis	erubis
Haml	.haml	haml
Sass	.sass	haml (< 3.1) or sass (>= 3.1)
Scss	.scss	haml (< 3.1) or sass (>= 3.1)
Less CSS	.less	less
Builder	.builder	builder
Liquid	.liquid	liquid
RDiscount	.markdown, .mkd, .md	rdiscount
Redcarpet	.markdown, .mkd, .md	redcarpet
BlueCloth	.markdown, .mkd, .md	bluecloth
Kramdown	.markdown, .mkd, .md	kramdown
Maruku	.markdown, .mkd, .md	maruku
RedCloth	.textile	redcloth
RDoc	.rdoc	rdoc
Radius	.radius	radius
Markaby	.mab	markaby
Nokogiri	.nokogiri	nokogiri
CoffeeScript	.coffee	coffee-script (+ javascript)
Creole (Wiki markup)	.wiki, .creole	creole
WikiCloth (Wiki	.wiki, .mediawiki, .mw	wikicloth
Yajl	.yajl	yajl-ruby

Other template engines

ENGINE	FILE EXTENSIONS	REQUIRED LIBRARIES
Slim	.slim	slim (>= 0.7)
Embedded JavaScript		sprockets
Embedded CoffeeScript		sprockets
JST		sprockets

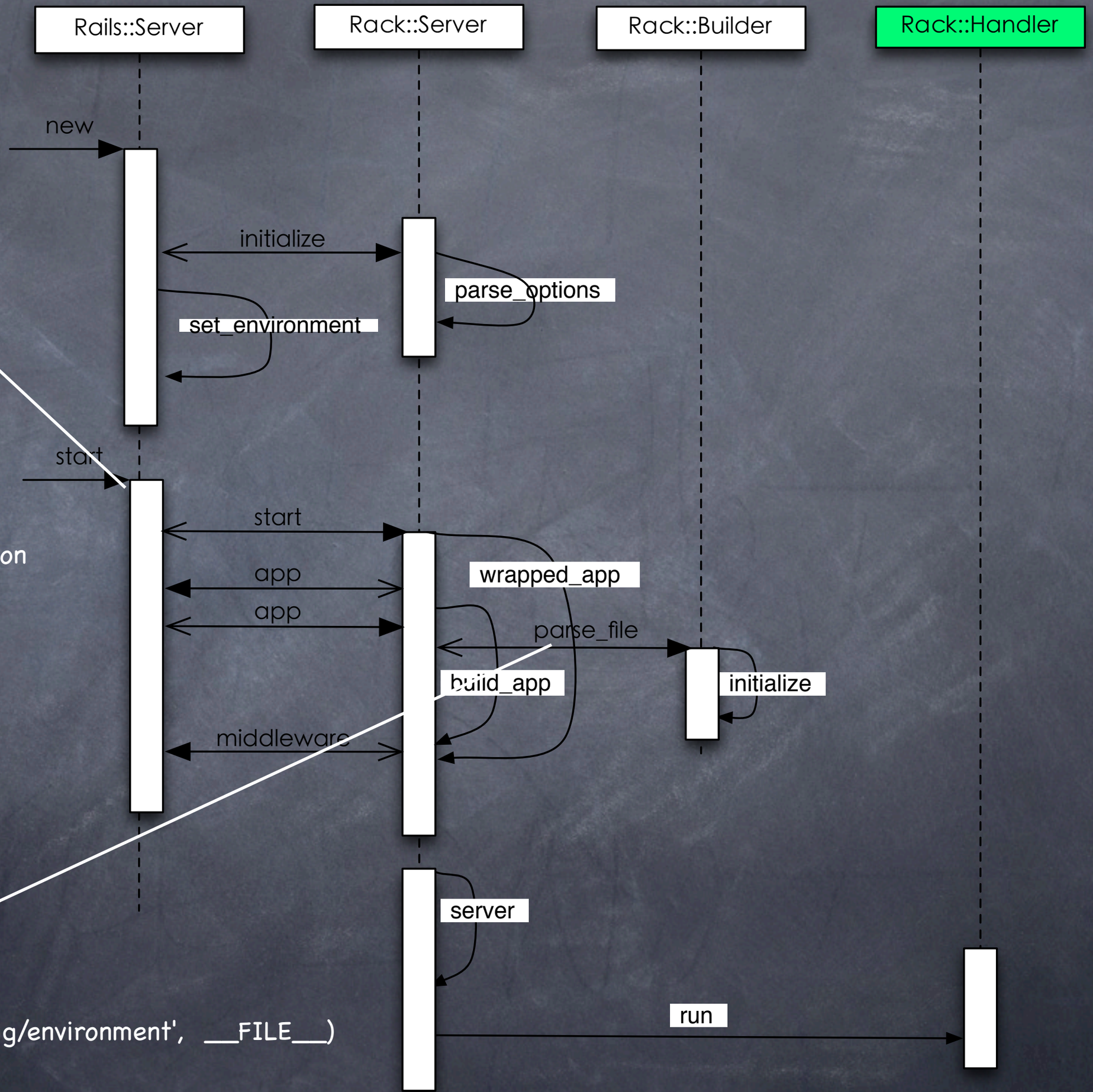
Rails MVC Model




Rack::Server
Rack::Builder
Rack::Handler
webrick
thin
mongrel
evented_mongrel
swiftiplied_mongrel
lsws
fastcgi
scgi
cgi
Handlers
initialize
parse_options(ARGV)
app
parse_file
initialize
build_app
start
self.start
wrapped_app
server
run

```
require 'rails/commands/server'
Rails::Server.new.tap { |server|
  require APP_PATH
  Dir.chdir(Rails.application.root)
  server.start
}
```

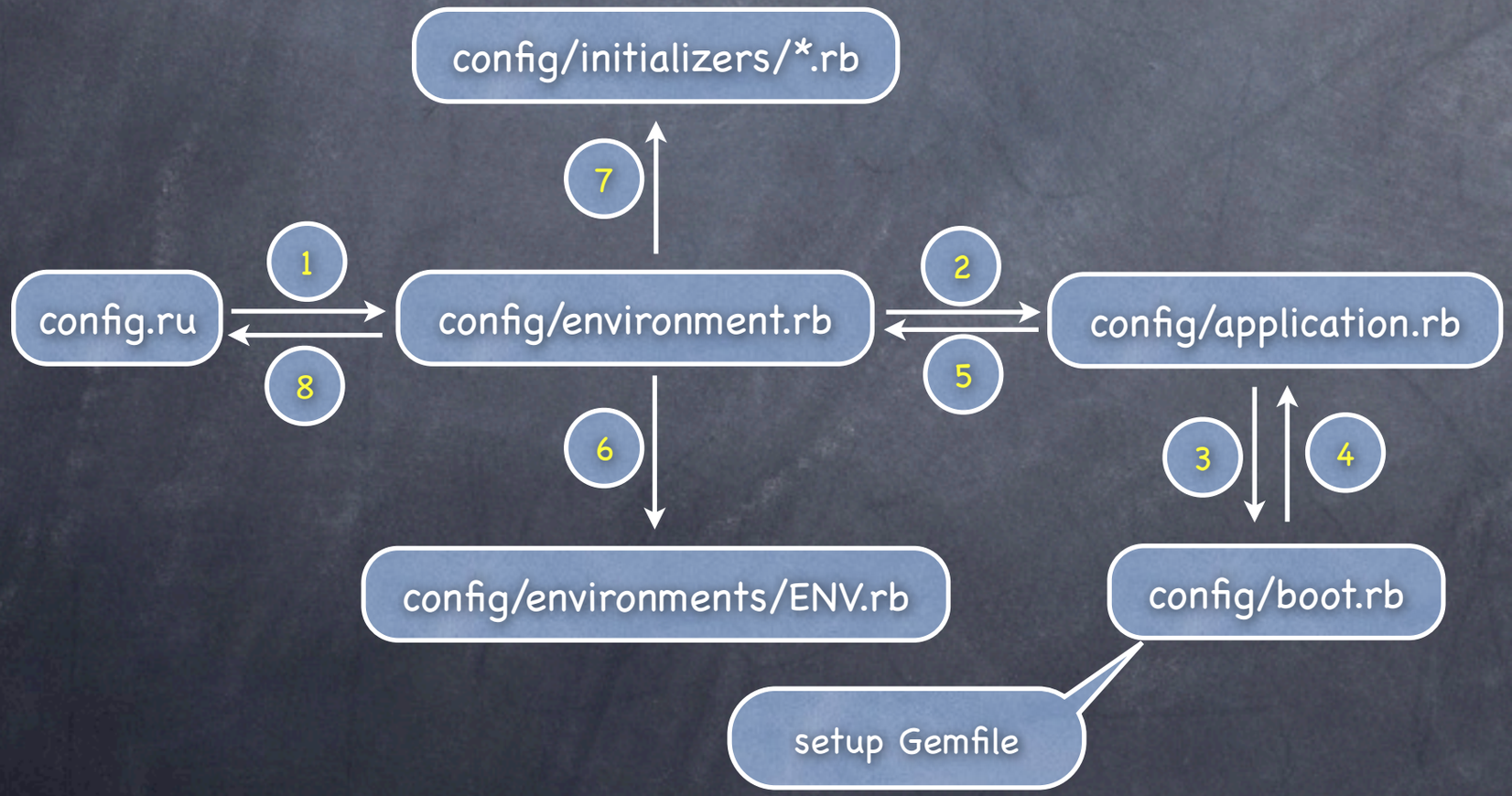
APP_PATH=${config.root}/application



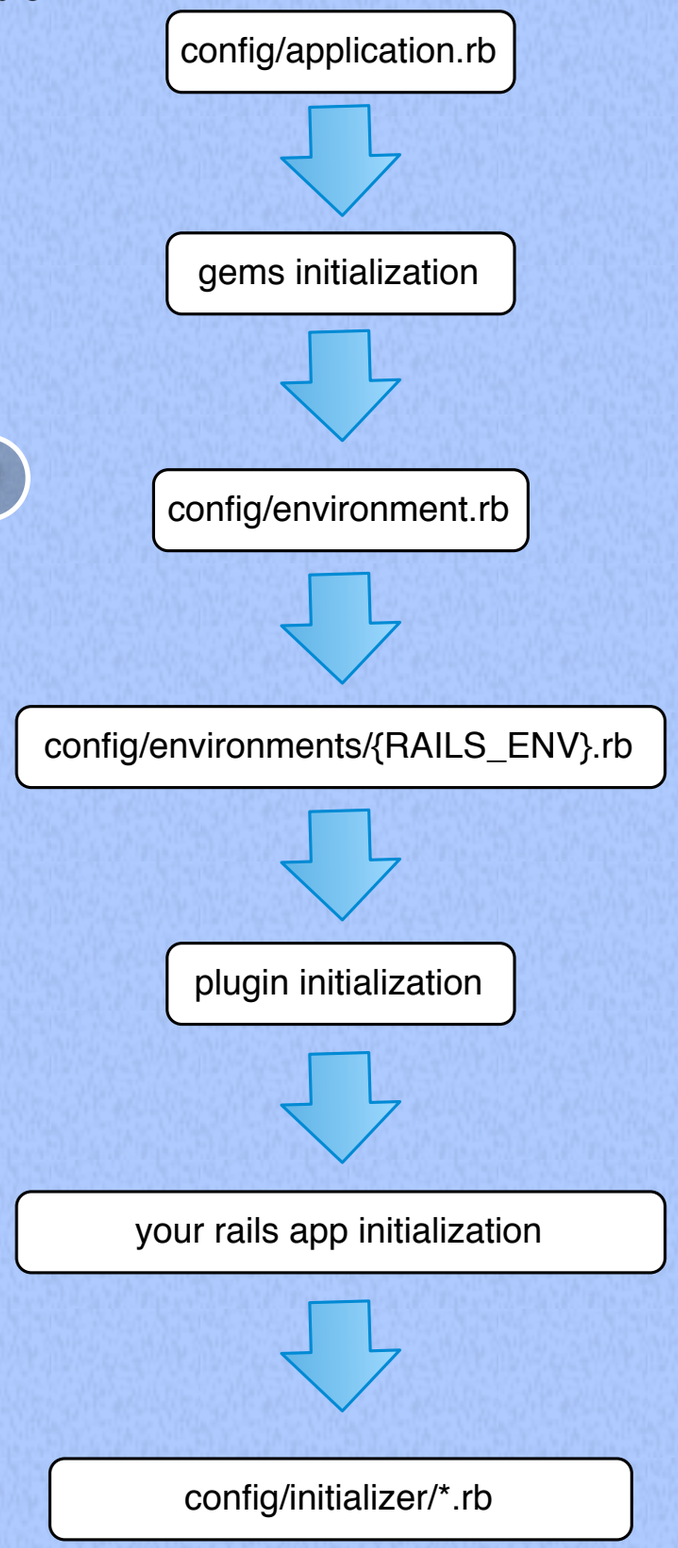

```
#config.ru
require ::File.expand_path('../config/environment',  __FILE__)
run ITutorial::Application
```

Rails Application Initialization

01) require "config/boot.rb" to setup load paths
02) require railties and engines
03) Define Rails.application as "class MyApp::Application < Rails::Application"
04) Run config.before_configuration callbacks
05) Load config/environments/ENV.rb
06) Run config.before_initialize callbacks
07) Run Railtie#initializer defined by railties, engines and application.
08) One by one, each engine sets up its load paths, routes and runs its config/initializers/* files.
09) Custom Railtie#initializers added by railties, engines and applications are executed
10) Build the middleware stack and run to_prepare callbacks
11) Run config.before_eager_load and eager_load if cache classes is true
12) Run config.after_initialize callbacks



Rails 3



Rails Server Start

```
#rails s
require 'rails/commands/server'
Rails::Server.new.tap { |server|
  require APP_PATH
  Dir.chdir(Rails.application.root)
  server.start
}
```

```
def start &blk
  if options[:warn]
    $-w = true
  end
  if includes = options[:include]
    $LOAD_PATH.unshift(*includes)
  end
  if library = options[:require]
    require library
  end
  check_pid! if options[:pid]
  wrapped_app
  daemonize_app if options[:daemonize]
  write_pid if options[:pid]
  trap(:INT) do
    if server.respond_to?(:shutdown)
      server.shutdown
    else
      exit
    end
  end
  server.run wrapped_app, options, &blk
end
```

```
#Rails::Server#start
def start
  url = "#{options[:SSLEnable] ? 'https' : 'http'}://#{options[:Host]}:#{options[:Port]}"
  puts "=> Booting #{ActiveSupport::Inflector.demodulize(server)}"
  puts "=> Rails #{Rails.version} application starting in #{Rails.env} on #{url}"
  puts "=> Call with -d to detach" unless options[:daemonize]
  trap(:INT) { exit }
  puts "=> Ctrl-C to shutdown server" unless options[:daemonize]
  %w(cache pids sessions sockets).each do |dir_to_make|
    FileUtils.mkdir_p(Rails.root.join('tmp', dir_to_make))
  end
  super
ensure
  puts 'Exiting' unless @options && options[:daemonize]
end
```

```ruby
#Rack::Server#wrapped_app
def wrapped_app
  @wrapped_app ||= build_app app
end
```

```ruby
#Rails::Server#app
def app
  @app ||= super.respond_to?(:to_app) ? super.to_app : super
end
```

```ruby
#Rack::Server#middleware
def middleware
  self.class.middleware
end
```

```ruby
#Rack::Server#build_app
def build_app(app)
  middleware[options[:environment]].reverse_each do |middleware|
    middleware = middleware.call(self) if middleware.respond_to?(:call)
    next unless middleware
    klass = middleware.shift
    app = klass.new(app, *middleware)
  end
  app
end
```

```ruby
#Rack::Server.middleware
def self.middleware
  @middleware ||= begin
    m = Hash.new {|h,k| h[k] = []}
    m["deployment"].concat [[Rack::ContentLength],
     [Rack::Chunked], logging_middleware]
    m["development"].concat m["deployment"]
      + [[Rack::ShowExceptions], [Rack::Lint]]
    m
  end
end
```

```ruby
#Rack::Server#app
def app
  @app ||= begin
    if !::File.exist? options[:config]
      abort "configuration #{options[:config]} not found"
    end
    app, options = Rack::Builder.parse_file(self.options[:config], opt_parser)
    self.options.merge! options
    app
  end
end
```

```ruby
#Rails::Server#middleware
def middleware
  middlewares = []
  middlewares << [Rails::Rack::LogTailer, log_path] unless options[:daemonize]
  middlewares << [Rails::Rack::Debugger]  if options[:debugger]
  middlewares << [::Rack::ContentLength]
  Hash.new(middlewares)
end
```

Rails Application Initialization

```
#config.ru
require ::File.expand_path('../config/environment',  __FILE__)
run ITutorial::Application
```

9

1

```
#config/environment.rb
require File.expand_path('../application', __FILE__)
ITutorial::Application.initialize!
```

6

2

```
#config/application.rb
require File.expand_path('../boot', __FILE__)
require 'rails/all'
if defined?(Bundler)
  Bundler.require(*Rails.groups(:assets => %w(development test)))
end
module ITutorial
  class Application < Rails::Application
    #Your Config Here
  end
end
```

4

5

7

```
def initialize!(group=:default) #:nodoc:
  raise "Application has been already initialized." if @initialized
  run_initializers(group, self)
  @initialized = true
  self
end
```

3

```
#config/boot.rb
require 'rubygems'
ENV['BUNDLE_GEMFILE'] ||= File.expand_path('../../Gemfile', __FILE__)
require 'bundler/setup' if File.exists?(ENV['BUNDLE_GEMFILE'])
```

8

```
def run_initializers(group=:default, *args)
  return if instance_variable_defined?(:@ran)
  initializers.tsort.each do |initializer|
    initializer.run(*args) if initializer.belongs_to?(group)
  end
  @ran = true
end
```

load config/environments/ENV.rb

load config/initializers/*.rb

Rails::Initializable

```
module Initializable
  def self.included(base)
    base.extend ClassMethods
  end
  def run_initializers(group=:default, *args)
    return if instance_variable_defined?(:@ran)
    initializers.tsort.each do |initializer|
      initializer.run(*args) if initializer.belongs_to?(group)
    end
    @ran = true
  end
  def initializers
    @initializers ||= self.class.initializers_for(self)
  end
  module ClassMethods
    def initializers
      @initializers ||= Collection.new
    end
    def initializers_chain
      initializers = Collection.new
      ancestors.reverse_each do |klass|
        next unless klass.respond_to?(:initializers)
        initializers = initializers + klass.initializers
      end
      initializers
    end
    def initializers_for(binding)
      Collection.new(initializers_chain.map { |i| i.bind(binding) })
    end
    def initializer(name, opts = {}, &blk)
      raise ArgumentError, "A block must be passed when defining an initializer" unless blk
      opts[:after] ||= initializers.last.name unless initializers.empty? || initializers.find { |i| i.name == opts[:before] }
      initializers << Initializer.new(name, nil, opts, &blk)
    end
  end
end
```

Rails Application Initialization

```
#Rails::Application
def initializers #:nodoc:
  Bootstrap.initializers_for(self) 1
  super + 2
  Finisher.initializers_f 5 elf)
end
```

```
#Rails::Engine
def initializers 3
  initializers = []
  ordered_railties.each do |r|
    if r == self
      initializers += super
    else
      initializers += r.initializers
    end
  end
  initializers
end
```

```
#Rails::Railtie
def initializers 4
  @initializers ||= self.class.initializers_for(self)
end
```

Rails Application Initialization

```
initializer :load_environment_hook, 7 oup => :all do end
```

```
initializer :set_load_path, 1 efore => :bootstrap_hook do
  _all_load_paths.reverse_each do |path|
    $LOAD_PATH.unshift(path) if File.directory?(path)
  end
  $LOAD_PATH.uniq!
end
```

```
initializer :load_active_support, 8 up => :all do
  require "active_support/all" unless config.active_support.bare
end
```

```
initializer :preload_frameworks, 9 oup => :all do
  ActiveSupport::Autoload.eager_autoload! if config.preload_frameworks
end
```

```
initializer :set_autoload_paths, 2 efore => :bootstrap_hook do |app|
  ActiveSupport::Dependencies.autoload_paths.unshift(*_all_autoload_paths)
  ActiveSupport::Dependencies.autoload_once_paths.unshift(*_all_autoload_once_paths)
  config.autoload_paths.freeze
  config.eager_load_paths.freeze
  config.autoload_once_paths.freeze
end
```

```
initializer :add_view_paths do 5
  views = paths["app/views"].existent
  unless views.empty?
    ActiveSupport.on_load(:action_controller){ prepend_view_path(views) }
    ActiveSupport.on_load(:action_mailer){ prepend_view_path(views) }
  end
end
```

auto add all the engines views

```
initializer :add_routing_paths, 3 app|
  paths = self.paths["config/routes"].existent
  if routes? || paths.any?
    app.routes_reloader.paths.unshift(*paths)
    app.routes_reloader.route_sets << routes
  end
end
```

```
initializer :add_locales, 4
  config.i18n.railties_load_path.concat(paths["config/locales"].existent)
end
```

load file in config/environments/ENV.rb

```
initializer :load_environment_config, 6 fore => :load_environment_hook, :group => :all do
  environment = paths["config/environments"].existent.first
  require environment if environment
end
```

```
initializer :initialize_logger, 10 p => :all do
```

```
initializer :initialize_cache, 11 oup => :all do
  unless defined?(RAILS_CACHE)
    silence_warnings { Object.const_set "RAILS_CACHE", ActiveSupport::Cache.lookup_store(config.cache_store) }
    config.middleware.insert_before("Rack::Runtime", RAILS_CACHE.middleware) if RAILS_CACHE.respond_to?(:middleware)
  end
end
```

Rails Application Initialization

```
initializer :initialize_dependency_mechanism, 12 oup => :all do
  ActiveSupport::Dependencies.mechanism = config.cache_classes ? :require : :load
end
```

```
initializer :bootstrap_hook, 13 up => :all do |appl
  ActiveSupport.run_load_hooks(:before_initialize, app)
end
```

```
#active_support/i18n_railtie.rb:
initializer "i18n.callbacks" 14 appl
  app.reloaders << I18n::Railtie.reloader
  ActionDispatch::Reloader.to_prepare do
    I18n::Railtie.reloader.execute_if_updated
  end
end
```

```
#active_support/railtie.rb:
initializer "active_support.initialize_whiny_nils" 15 appl
  require 'active_support/whiny_nil' if app.config.whiny_nils
end
initializer "active_support.deprecation_behavior" 16 |appl
#.....
end
initializer "active_support.initialize_time_zone" 17 appl
#.....
end
```

```
#action_dispatch/railtie.rb:
initializer "action_dispatch.configure" 18 appl
  ActionDispatch::Http::URL.tld_length = app.config.action_dispatch.tld_length
  ActionDispatch::Request.ignore_accept_header = app.config.action_dispatch.ignore_accept_header
  ActionDispatch::Response.default_charset = app.config.action_dispatch.default_charset || app.config.encoding
  ActionDispatch::ExceptionWrapper.rescue_responses.merge!(config.action_dispatch.rescue_responses)
  ActionDispatch::ExceptionWrapper.rescue_templates.merge!(config.action_dispatch.rescue_templates)
  config.action_dispatch.always_write_cookie = Rails.env.development? if config.action_dispatch.always_write_cookie.nil?
  ActionDispatch::Cookies::CookieJar.always_write_cookie = config.action_dispatch.always_write_cookie
end
```

Rails Application Initialization

```
#action_view/railtie.rb
initializer "action_view.embed_authenticity_token_in_remote_forms" 19 ,app|
  ActiveSupport.on_load(:action_view) do
    ActionView::Helpers::FormTagHelper.embed_authenticity_token_in_remote_forms = app.config.action_view.delete(:embed_authenticity_token_in_remote_forms)
  end
end
initializer "action_view.cache_asset_ids" 20 app|
  unless app.config.cache_classes
    ActiveSupport.on_load(:action_view) { ActionView::Helpers::AssetTagHelper::AssetPaths.cache_asset_ids = false }
  end
end
initializer "action_view.javascript_expansions" 21 |app|
  ActiveSupport.on_load(:action_view) do
    ActionView::Helpers::AssetTagHelper.register_javascript_expansion(app.config.action_view.delete(:javascript_expansions))
    ActionView::Helpers::AssetTagHelper.register_stylesheet_expansion(app.config.action_view.delete(:stylesheet_expansions))
  end
end
initializer "action_view.set_configs" 22 app|
  ActiveSupport.on_load(:action_view) do
    app.config.action_view.each { |k,v| send "#{k}=", v }
  end
end
initializer "action_view.caching" 23 app|
  ActiveSupport.on_load(:action_view) do
    ActionView::Resolver.caching = app.config.cache_classes if app.config.action_view.cache_template_loading.nil?
  end
end
```

Rails Application Initialization

```
#action_controller/railtie.rb
initializer "action_controller.logger" 24
  ActiveSupport.on_load(:action_controller) { self.logger ||= Rails.logger }
end
initializer "action_controller.initialize_framework_caches" 25
  ActiveSupport.on_load(:action_controller) { self.cache_store ||= RAILS_CACHE }
end
initializer "action_controller.assets_config", :group => :all 26 app|
  app.config.action_controller.assets_dir ||= app.config.paths["public"].first
end
initializer "action_controller.set_configs" 27 app|
  paths    = app.config.paths
  options = app.config.action_controller
  options.javascripts_dir       ||= paths["public/javascripts"].first
  options.stylesheets_dir       ||= paths["public/stylesheets"].first
  options.page_cache_directory ||= paths["public"].first
  options.asset_path            ||= app.config.asset_path
  options.asset_host            ||= app.config.asset_host
  options.relative_url_root     ||= app.config.relative_url_root
  ActiveSupport.on_load(:action_controller) do
    include app.routes.mounted_helpers
    extend ::AbstractController::Railties::RoutesHelpers.with(app.routes)
    extend ::ActionController::Railties::Paths.with(app)
    options.each { |k,v| send("#{k}=", v) }
  end
end
initializer "action_controller.compile_config_methods" 28
  ActiveSupport.on_load(:action_controller) do
    config.compile_methods! if config.respond_to?(:compile_methods!)
  end
end
```

Rails Application Initialization

```
#active_record/railtie.rb
initializer "active_record.initialize_timezone" do
  ActiveSupport.on_load(:active_record) do
    self.time_zone_aware_attributes = true
    self.default_timezone = :utc
  end
end
initializer "active_record.logger" do
  ActiveSupport.on_load(:active_record) { self.logger ||= ::Rails.logger }
end
initializer "active_record.identity_map" do |app|
  config.app_middleware.insert_after "::ActionDispatch::Callbacks",
    "ActiveRecord::IdentityMap::Middleware" if config.active_record.delete(:identity_map)
end
initializer "active_record.set_configs" do |app|
  ActiveSupport.on_load(:active_record) do
    attr_accessible(nil) if app.config.active_record.delete(:whitelist_attributes)
    app.config.active_record.each { |k,v| send "#{k}=", v }
  end
end
initializer "active_record.initialize_database" do |app|
  ActiveSupport.on_load(:active_record) do
    db_connection_type = "DATABASE_URL"
    unless ENV['DATABASE_URL']
      db_connection_type  = "database.yml"
      self.configurations = app.config.database_configuration
    end
    Rails.logger.info "Connecting to database specified by #{db_connection_type}"
    establish_connection
  end
end
```

29 30 31 32 33

Rails Application Initialization

```
#active_record/railtie.rb
initializer "active_record.log_runtime" do |app|
  require "active_record/railties/controller_runtime"
  ActiveSupport.on_load(:action_controller) do
    include ActiveRecord::Railties::ControllerRuntime
  end
end
initializer "active_record.set_reloader_hooks" do |app|
  hook = lambda do
    ActiveRecord::Base.clear_reloadable_connections!
    ActiveRecord::Base.clear_cache!
  end
  if app.config.reload_classes_only_on_change
    ActiveSupport.on_load(:active_record) do
      ActionDispatch::Reloader.to_prepare(&hook)
    end
  else
    ActiveSupport.on_load(:active_record) do
      ActionDispatch::Reloader.to_cleanup(&hook)
    end
  end
end
initializer "active_record.add_watchable_files" do |app|
  config.watchable_files.concat ["#{app.root}/db/schema.rb", "#{app.root}/db/structure.sql"]
end
```

```
#action_mailer/railtie.rb
initializer "action_mailer.logger" do
  ActiveSupport.on_load(:action_mailer) { self.logger ||= Rails.logger }
end
initializer "action_mailer.set_configs" do |app|
  paths    = app.config.paths
  options = app.config.action_mailer
  options.assets_dir       ||= paths["public"].first
  options.javascripts_dir ||= paths["public/javascripts"].first
  options.stylesheets_dir ||= paths["public/stylesheets"].first
  options.asset_path          ||= app.config.asset_path
  options.asset_host           ||= app.config.asset_host
  options.relative_url_root  ||= app.config.relative_url_root
  ActiveSupport.on_load(:action_mailer) do
    include AbstractController::UrlFor
    extend ::AbstractController::Railties::RoutesHelpers.with(app.routes)
    include app.routes.mounted_helpers
    register_interceptors(options.delete(:interceptors))
    register_observers(options.delete(:observers))
    options.each { |k,v| send("#{k}=", v) }
  end
end
```

```
#action_mailer/railtie.rb
initializer "action_mailer.compile_config_methods" do
  ActiveSupport.on_load(:action_mailer) do
    config.compile_methods! if config.respond_to?(:compile_methods!)
  end
end
```

```ruby
#active_resource/railtie.rb
initializer "active_resource.set_configs" do |app|
  app.config.active_resource.each do |k,v|
    ActiveResource::Base.send "#{k}=", v
  end
end
#sprockets/railtie.rb
initializer "sprockets.environment", :group => :all do |app|
  config = app.config
  next unless config.assets.enabled
  require 'sprockets'
  app.assets = Sprockets::Environment.new(app.root.to_s) do |env|
    env.version = ::Rails.env + "-#{config.assets.version}"
    env.logger  = config.assets.logger || ::Rails.logger if config.assets.logger != false
    if config.assets.cache_store != false
      env.cache = ActiveSupport::Cache.lookup_store(config.assets.cache_store) || ::Rails.cache
    end
  end
  if config.assets.manifest
    path = File.join(config.assets.manifest, "manifest.yml")
  else
    path = File.join(Rails.public_path, config.assets.prefix, "manifest.yml")
  end
  config.assets.digests = YAML.load_file(path) if File.exist?(path)
  ActiveSupport.on_load(:action_view) do
    include ::Sprockets::Helpers::RailsHelper
    app.assets.context_class.instance_eval do
      include ::Sprockets::Helpers::IsolatedHelper
      include ::Sprockets::Helpers::RailsHelper
    end
  end
end
```

```
#sass/rails/railtie.rb
initializer :setup_sass, :group => :all do |app|
  # Only emit one kind of syntax because though we have registered two kinds of generators
  syntax     = app.config.sass.preferred_syntax.to_sym
  alt_syntax = syntax == :sass ? "scss" : "sass"
  app.config.generators.hide_namespace alt_syntax
  # Override stylesheet engine to the preferred syntax
  config.app_generators.stylesheet_engine syntax
  # Set the sass cache location
  config.sass.cache_location   = File.join(Rails.root, "tmp/cache/sass")
  # Establish configuration defaults that are evironmental in nature
  if config.sass.full_exception.nil?
    # Display a stack trace in the css output when in development-like environments.
    config.sass.full_exception = app.config.consider_all_requests_local
  end
  if app.assets
    app.assets.context_class.extend(SassContext)
    app.assets.context_class.sass_config = app.config.sass
  end
  Sass.logger = app.config.sass.logger
end
initializer :setup_compression, :group => :all do |app|
  if app.config.assets.compress
    # Use compressed style if none specified
    app.config.sass.style ||= :compressed
    app.config.assets.css_compressor ||= CssCompressor.new(:style => app.config.sass.style)
  else
    # Use expanded output instead of the sass default of :nested unless specified
    app.config.sass.style ||= :expanded
  end
end
```

```
initializer :append_assets_path, :group => :all do |app|
  app.config.assets.paths.unshift(*paths["vendor/assets"].existent_directories)
  app.config.assets.paths.unshift(*paths["lib/assets"].existent_directories)
  app.config.assets.paths.unshift(*paths["app/assets"].existent_directories)
end
initializer :prepend_helpers_path do |app|
  if !isolated? || (app == self)
    app.config.helpers_paths.unshift(*paths["app/helpers"].existent)
  end
end
initializer :load_config_initializers do
  config.paths["config/initializers"].existent.sort.each do |initializer|
    load(initializer)
  end
end
initializer :engines_blank_point do
# We need this initializer so all extra initializers added in engines are
# consistently executed after all the initializers above across all engines.
end
```

auto add all the engines assets

auto add all the engines helpers

load all files in config/initializers

```ruby
#rails/application/finisher.rb
initializer :add_generator_templates do -10
  config.generators.templates.unshift(*paths["lib/templates"].existent)
end
initializer :ensure_autoload_once_paths_as_subset do -9
  extra = ActiveSupport::Dependencies.autoload_once_paths - ActiveSupport::Dependencies.autoload_paths
  unless extra.empty?
    abort <<-end_error
      autoload_once_paths must be a subset of the autoload_paths.
      Extra items in autoload_once_paths: #{extra * ','}
    end_error
  end
end
initializer :add_builtin_route do |app| -8
  if Rails.env.development?
    app.routes.append do
      match '/rails/info/properties' => "rails/info#properties"
    end
  end
end
initializer :build_middleware_stack do -7
  build_middleware_stack
end
initializer :define_main_app_helper do |app| -6
  app.routes.define_mounted_helper(:main_app)
end
initializer :add_to_prepare_blocks do -5
  config.to_prepare_blocks.each do |block|
    ActionDispatch::Reloader.to_prepare(&block)
  end
end
```

```ruby
#rails/application/finisher.rb
initializer :run_prepare_callbacks do -4
  ActionDispatch::Reloader.prepare!
end
initializer :eager_load! do
  if config.cache_classes && !(defined?($rails_rake_task) && $rails_rake_task)
    ActiveSupport.run_load_hooks(:before_eager_load, self)
    eager_load!
  end
end
initializer :finisher_hook do -3
  ActiveSupport.run_load_hooks(:after_initialize, self)
end
initializer :set_routes_reloader_hook do
  reloader = routes_reloader
  reloader.execute_if_updated
  self.reloaders << reloader
  ActionDispatch::Reloader.to_prepare { reloader.execute_if_updated }
end
initializer :set_clear_dependencies_hook, :group => :all do -2
  callback = lambda do
    ActiveSupport::DescendantsTracker.clear
    ActiveSupport::Dependencies.clear
  end
  if config.reload_classes_only_on_change
    reloader = config.file_watcher.new(*watchable_args, &callback)
    self.reloaders << reloader
    ActionDispatch::Reloader.to_prepare(:prepend => true){ reloader.execute }
  else
    ActionDispatch::Reloader.to_cleanup(&callback)
  end
end
```

```ruby
initializer :disable_dependency_loading do -1
  if config.cache_classes && !config.dependency_loading
    ActiveSupport::Dependencies.unhook!
  end
end
```

```ruby
def eager_load!
 railties.all(&:eager_load!)
 config.eager_load_paths.each do |load_path|
   matcher = /\A#{Regexp.escape(load_path)}\/(.*)\.rb\Z/
   Dir.glob("#{load_path}/**/*.rb").sort.each do |file|
     require_dependency file.sub(matcher, '\1')
   end
 end
end
```

Plugin

Railtie

Engine

Application

```
def inherited(base)
  unless base.abstract_railtie?
    base.send(:include, Railtie::Configurable)
    subclasses << base
  end
end
```

```
def self.inherited(base)
  raise "You cannot inherit from Rails::Plugin"
end
```

```
def inherited(base)
  unless base.abstract_railtie?
    base.called_from = begin
    # Remove the line number from backtraces making sure we don't leave anything behind
    call_stack = caller.map { |p| p.sub(/:\d+.*/, '') }
    File.dirname(call_stack.detect { |p| p !~ %r[railties[\w.-]*/lib/rails|rack[\w.-]*/lib/rack] })
    end
  end
  super
end
```

```
def inherited(base)
  raise "You cannot have more than one Rails::Application" if Rails.application
  super
  Rails.application = base.instance
  Rails.application.add_lib_to_load_path!
  ActiveSupport.run_load_hooks(:before_configuration, base.instance)
end
```

@initializers => []
@subclasses => [
I18n::Railtie,
ActiveSupport::Railtie,
ActionDispatch::Railtie,
ActionView::Railtie,
ActionController::Railtie,
ActiveRecord::Railtie,
ActionMailer::Railtie,
ActiveResource::Railtie,
Rails::TestUnitRailtie,
Sprockets::Railtie,
Kaminari::Railtie,
Sass::Rails::Railtie,
Jquery::Rails::Railtie,
RSpec::Rails::Railtie,
FactoryGirl::Railtie]

Railtie

@initializers => [
:handle_lib_autoload,
:load_init_rb,
:sanity_check_railties_collision]

Plugin

@initializers => []
@subclasses => [ITutorial::Application]

Application

@initializers => [
:set_load_path,
:set_autoload_paths,
:add_routing_paths,
:add_locales,
:add_view_paths,
:load_environment_config,
:append_assets_path,
:prepend_helpers_path,
:load_config_initializers,
:engines_blank_point]
@rake_tasks => [{railtie_name}/:install/:migrations]
@subclasses => [Devise::Engine, Kaminari::Engine, Coffee::Rails::Engine, Jquery::Rails::Engine]

Engine

Railtie

```
def rake_tasks(&blk)
  @rake_tasks ||= []
  @rake_tasks << blk if blk
  @rake_tasks
end
```

```
def console(&blk)
  @load_console ||= []
  @load_console << blk if blk
  @load_console
end
```

```
def generators(&blk)
  @generators ||= []
  @generators << blk if blk
  @generators
end
```

```
def load_tasks(app=self)
  extend Rake::DSL if defined? Rake::DSL
  self.class.rake_tasks.each { |block| self.instance_exec(app, &block) }
  # load also tasks from all superclasses
  klass = self.class.superclass
  while klass.respond_to?(:rake_tasks)
    klass.rake_tasks.each { |t| self.instance_exec(app, &t) }
    klass = klass.superclass
  end
end
```

```
def load_console(app=self)
  self.class.console.each { |block| block.call(app) }
end
```

```
def load_generators(app=self)
  self.class.generators.each { |block| block.call(app) }
end
```

```
#include Initializable
def initializer(name, opts = {}, &blk)
  raise ArgumentError, "A block must be passed when defining an initializer" unless blk
  opts[:after] ||= initializers.last.name unless initializers.empty? || initializers.find { |i| i.name == opts[:before] }
  initializers << Initializer.new(name, nil, opts, &blk)
end
```

```
def config
  @config ||= Railtie::Configuration.new
end
```

Engine

```
def load_tasks(app=self)
  railties.all { |r| r.load_tasks(app) }
  super
  paths["lib/tasks"].existent.sort.each { |ext| load(ext) }
end
def load_generators(app=self)
  initialize_generators
  railties.all { |r| r.load_generators(app) }
  Rails::Generators.configure!(app.config.generators)
  super
  self
end
def initialize_generators
  require "rails/generators"
end
def load_console(app=self)
  railties.all { |r| r.load_console(app) }
  super
end
```

```
def eager_load!
  railties.all(&:eager_load!)
  config.eager_load_paths.each do |load_path|
    matcher = /\A#{Regexp.escape(load_path)}\/(.*)\.rb\Z/
    Dir.glob("#{load_path}/**/*.rb").sort.each do |file|
      require_dependency file.sub(matcher, '\1')
    end
  end
end
```

```
def find_root_with_flag(flag, default=nil)
  root_path = self.class.called_from
  while root_path && File.directory?(root_path) && !File.exist?
("#{root_path}/#{flag}")
    parent = File.dirname(root_path)
    root_path = parent != root_path && parent
  end
  root = File.exist?("#{root_path}/#{flag}") ? root_path : default
  raise "Could not find root path for #{self}" unless root
  RbConfig::CONFIG['host_os'] =~ /mswin|mingw/ ?
    Pathname.new(root).expand_path : Pathname.new(root).realpath
end
```

```
def default_middleware_stack
  ActionDispatch::MiddlewareStack.new
end
```

```
def app
  @app ||= begin
    config.middleware = config.middleware.merge_into(default_middleware_stack)
    config.middleware.build(endpoint)
  end
end
```

```
def env_config
  @env_config ||= {
    'action_dispatch.routes' => routes
  }
end
```

```
def routes
  @routes ||= ActionDispatch::Routing::RouteSet.new
  @routes.append(&Proc.new) if block_given?
  @routes
end
```

```
def endpoint
  self.class.endpoint || routes
end
```

```
    app.call(env.merge!(env_config))
  end
```

```
def config
  @config ||= Engine::Configuration.new(find_root_with_flag("lib"))
end
```

Application

```
def initialize_tasks #:nodoc:
  self.class.rake_tasks do
    require "rails/tasks"
    task :environment do
      $rails_rake_task = true
      require_environment!
    end
  end
end

def load_tasks(app=self)
  initialize_tasks
  super
  self
 end

 def initialize_console #:nodoc:
   require "pp"
   require "rails/console/app"
   require "rails/console/helpers"
 end

def load_console(app=self)
  initialize_console
  super
  self
end

def add_lib_to_load_path! #:nodoc:
  path = config.root.join('lib').to_s
  $LOAD_PATH.unshift(path) if File.exists?(path)
 end
```

```
def config #:nodoc:
  @config ||= Application::Configuration.new(find_root_with_flag("config.ru", Dir.pwd))
end
```

```
def call(env)
  env["ORIGINAL_FULLPATH"] = build_original_fullpath(env)
  super(env)
end
```

```
def to_app
  self
end
```

```
def env_config
  @env_config ||= super.merge({
    "action_dispatch.parameter_filter" => config.filter_parameters,
    "action_dispatch.secret_token" => config.secret_token,
    "action_dispatch.show_exceptions" => config.action_dispatch.show_exceptions,
    "action_dispatch.show_detailed_exceptions" => config.consider_all_requests_local,
    "action_dispatch.logger" => Rails.logger,
    "action_dispatch.backtrace_cleaner" => Rails.backtrace_cleaner
  })
end
```

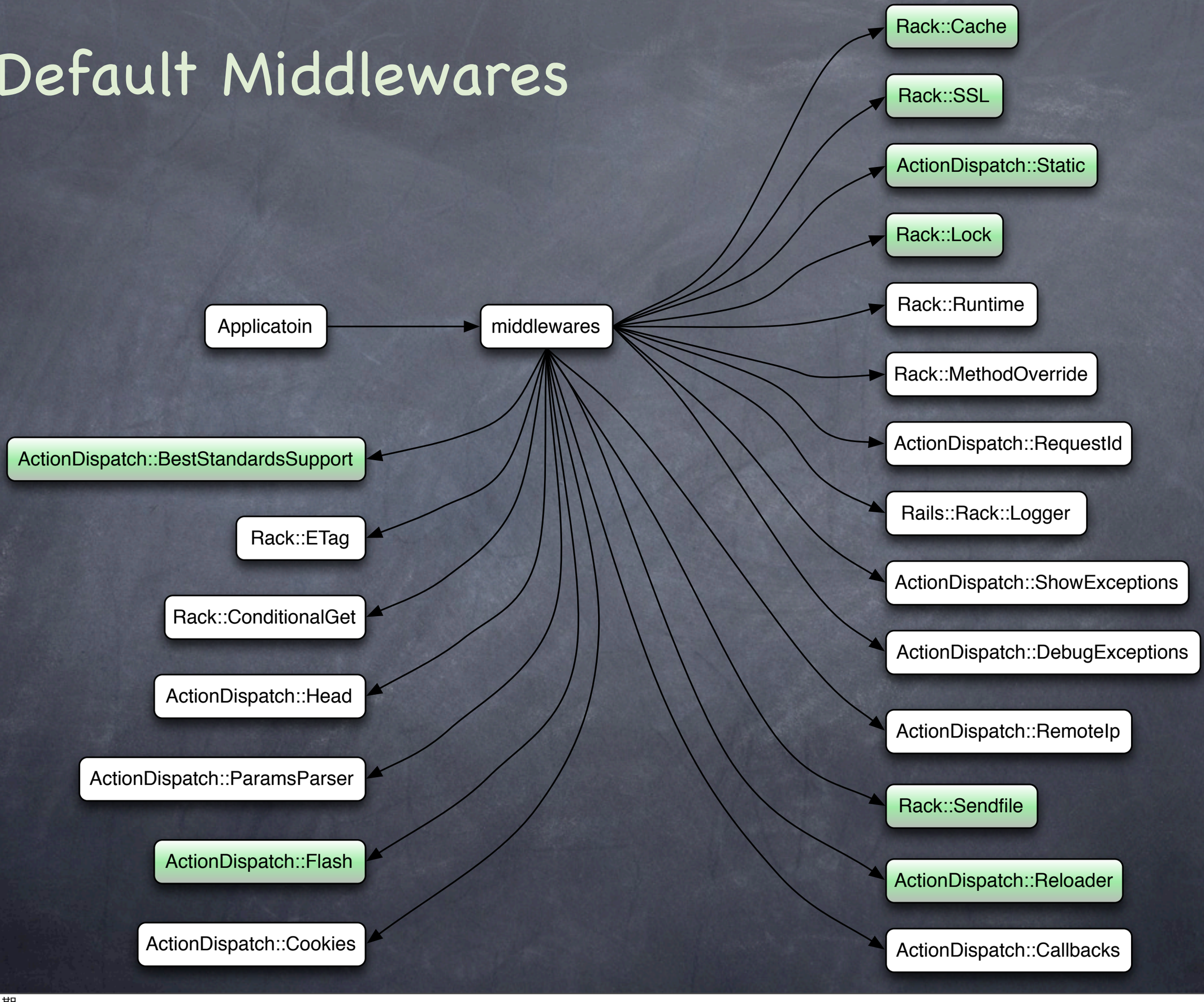
```
def default_middleware_stack
  require 'action_controller/railtie'
  ActionDispatch::MiddlewareStack.new.tap do |middleware|
    #more...
  end
end
```

Rails::Railtie::Configurable

```
module Configurable
  extend ActiveSupport::Concern
  module ClassMethods
    delegate :config, :to => :instance

    def inherited(base)
      raise "You cannot inherit from a #{self.superclass.name} child"
    end

    def instance
      @instance ||= new
    end

    def respond_to?(*args)
        super || instance.respond_to?(*args)
    end

    def configure(&block)
      class_eval(&block)
    end

    protected
    def method_missing(*args, &block)
      instance.send(*args, &block)
    end
  end
end
```

```
#file place in {config.root}/environments/ENV.rb
ITutorial::Application.configure do
  config.cache_classes = false
  config.whiny_nils = true
  config.consider_all_requests_local       = true
  config.action_controller.perform_caching = false
  config.action_mailer.raise_delivery_errors = false
  config.active_support.deprecation = :log
  config.action_dispatch.best_standards_support = :builtin
  config.active_record.mass_assignment_sanitizer = :strict
  config.active_record.auto_explain_threshold_in_seconds = 0.5
  config.assets.compress = false
  config.assets.debug = true
end
```

Default Middlewares



Routes

```
def routes
  @routes ||= ActionDispatch::Routing::RouteSet.new
  @routes.append(&Proc.new) if block_given?
  @routes
end
```

```
def call(env)
  finalize
  @router.call(env)
end
```