

desarrollo de plug-ins con rails



jramirez@aspgems.com

conferencia rails 2007. 23 de Noviembre, Madrid

obra publicada por javier ramirez como 'Atribución-No Comercial-Licenciar Igual 2.5' de Creative Commons



» Ingeniería de Software: aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la Ingeniería al Software

(IEEE Standard Computer Dictionary)

A C O P L A M I E N T O

- » mide las dependencias que existen entre los diferentes **módulos** (mensajes, datos, estructura, común, control, externo, común y contenido)
- » un nivel de acoplamiento bajo facilita la reutilización y el mantenimiento



- » mide la atomicidad de las responsabilidades en un módulo (cohesión funcional)
- » un nivel de cohesión alto permite una mayor legibilidad y favorece la reutilización del código



» crear módulos que hagan una única cosa y que sean tan independientes como sea posible

O B J E T I V O

R E A L

- » trabajar lo menos posible
- » para disimular podemos usar conceptos como DRY o less is more ;))



- » cuando vamos a copiarpegar un fragmento de código relevante de un proyecto a otro
- » extraer, no anticipar. Más práctico, más usable, menos trabajo



- » módulos completamente separados de mi aplicación que realizan una única tarea
- » mecanismo de compartición simple
- » también para uso interno

ESTRUCTURA

ruby script/generate plugin useless

```
create vendor/plugins/useless/lib
create vendor/plugins/useless/tasks
create vendor/plugins/useless/test
create vendor/plugins/useless/README
create vendor/plugins/useless/MIT-LICENSE
create vendor/plugins/useless/Rakefile
create vendor/plugins/useless/init.rb
create vendor/plugins/useless/install.rb
create vendor/plugins/useless/uninstall.rb
create vendor/plugins/useless/lib/useless.rb
create vendor/plugins/useless/tasks/useless_tasks.rake
create vendor/plugins/useless/test/useless_test.rb
```

I N S T A L A C I O N

- » `install.rb` / `uninstall.rb`
- » instalación off-line
- » mejor usar tareas rake y que el script de instalación solamente muestre el README



- » selección de plugins y su orden
- » modificación del `load_path`
- » inicialización del plugin



» `config.plugins_path`

(["#{root_path}/vendor/plugins"])

» `config.plugin_locator`

(Plugin::FileSystemLocator < Plugin::Locator)

» `config.plugin_loader`

(Plugin::Loader)



» vendor/plugins alfabéticamente

» config.plugins =

» [] => ningún plugin

» [:fckeditor, :useless] => en orden

» [:useless, :all] => primero useless

» [:all, :useless] => último useless

» [:fckeditor, :all, :useless] => primero fckeditor, último useless. El resto alfabéticamente

L O A D

P A T H

- » `add_plugin_load_paths` de `Plugin::Loader`
- » añade directorio lib de cada plugin al `load_path` de rails
- » añade lib también a `Dependencies` para autoload de clases

INICIALIZACION

- » Plugin::Loader llama al método load para cada plugin
- » se invoca a `init.rb`
variables locales: `directory`, `name`, `config`
- » punto para extender clases, modificar paths, inicializar variables...



- » pueden dejarse directamente en lib y se comportarán como si estuvieran en app/model
- » problema con creación de migrations mediante generadores o templates
- » mejor usar tareas rake

CONTROLADORES

- » pueden dejarse directamente en lib, pero necesitamos añadir la ruta a `config.controller_paths`
- » `config.controller_paths << directory`

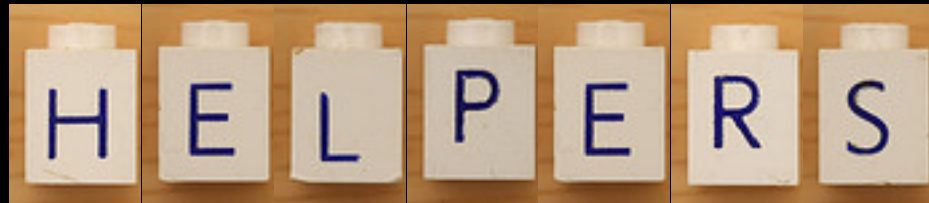


» ~~controller.template_root~~ (pre rails 2)

» controller.view_paths

» controller.prepend_view_path

» controller.append_view_path



- » pueden dejarse directamente en lib y se comportarán como si estuvieran en app/helpers
- » al no estar asociados a un controller, debemos hacer un include explícitamente para usarlos



» podemos definir módulos e incluirlos en todos los controladores, vistas (vía helper) o modelos *

- » ActionController::Base.send(:include, MyControllerModule)
- » ActionView::Base.send(:include, MyHelperModule)
- » ActiveRecord::Base.send(:include, MyARModule)

* podríamos reabrir directamente las clases Base pero queda menos elegante



» podemos crearnos directorios dentro de lib y añadirlos al load_path y a Dependencies

```
c_path = File.join(directory, 'app', 'controllers')
```

```
m_path = File.join(directory, 'app', 'models')
```

```
h_path = File.join(directory, 'app', 'helpers')
```

```
$LOAD_PATH << c_path << m_path << h_path
```

```
Dependencies.load_paths << c_path << m_path << h_path
```

```
config.controller_paths << c_path
```



- » un plugin que pueda usarse como `'acts_as_xxx'` en un modelo (o controlador) requiere un poco de metaprogramación
- » debemos crear un método de clase sobre `xxx::Base` para desde él incluir nuestro módulo en las clases hijas que lo usen

A C T S

A S

U S E L E S S

```
module ActsAsUseless
  def ActsAsUseless.included(base_class)
    base_class.extend ARClassMethods #mix-in sobre la clase AR
  end

  module ARClassMethods
    def acts_as_useless
      include ActsAsUseless::ModelMethods #mix-in sobre instancias
    end
  end

  module ModelMethods #nuestros métodos para el plugin
    def who_are_you
      "useless #{self.class.to_s}"
    end
  end
end
```

CONFIGURACION

- » normalmente se inicializan constantes en `init.rb` si es necesario
- » es más claro definirse un módulo en `directory/lib` y definirse `mattr_accessors` para las variables de configuración

```
module Useless
  mattr_accessor :useless_variable
  self.useless_variable = 'default_value'
end
```

A L I A S

M E T H O D

C H A I N

```
module ActionController
  module Routing # :nodoc:
    class RouteSet # :nodoc:
      def draw_with_content
        draw_without_content do |map|
          map.connect "", :controller => 'ct', :action=>'xx' if Content.map_home
          map.connect
            "#{Content.web_prefix}/*page_path", :controller=>'ct', :action=>'xx'
          yield map
        end
      end
      alias_method_chain :draw, :content
    end #routeset
  end #routing
end #actioncontroller
```

T A R E A S

R A K E

- » buenas sustitutas para `install.rb`
en `install` podría hacer `Rake::Task[:initialize].invoke`
- » podemos crear tareas rake para crear tablas, inicializar datos, copiar ficheros estáticos...
- » ...y para deshacer esos cambios al desinstalar



» se escriben tareas rake estándar.
Simplemente hay que dejar los ficheros
en `directory/tasks`

```
namespace :useless do
  desc 'useless task. plugin example'
  task :demo => :environment do
    puts "i told you i am a shallow useless task"
  end
  desc 'useless task without environment. plugin example'
  task :light_demo do
    puts "i don't even have the environment"
  end
end
```

INICIALIZACION RAKE

» buena idea usar `find_or_initialize` para tolerar que el usuario ejecute varias veces sin introducir duplicados

» podemos seguir la filosofía de las migrations, definiendo un método `up` y `down` y llamarlos desde rake usando métodos como

```
ActiveRecord::Base.connection.create_table :roles, :force  
=> false do |t|  
  t.column ...  
end
```

GENERADORES

- » podemos incluir generadores de código que permitan usar templates para crear ficheros desde el plugin
- » al ejecutar `script/generate` además de buscar en el path de rails, de la aplicación y de las gemas, también se busca en todos los directorios `/generators` de los plugins instalados en el proyecto

GENERADORES

- » un generador es una clase que hereda de `Rails::Generator::Base` o `Rails::Generator::NamedBase`
- » el comportamiento es idéntico, pero `NamedBase` se usa cuando el generador recibe el nombre de una clase o modelo como parámetro. `NamedBase` proporciona varias variables de conveniencia como `class_name`, `table_name`, `plural_name`, `singular_name`...

GENERADORES

- » se debe implementar un método manifest
- » en este método se pueden llevar a cabo una serie de comandos, siendo los más habituales la creación de un directorio (directory), la copia de un fichero estático (file) o la copia de un fichero dinámico (template)
- » manifest es capaz de deshacer las acciones si se llama a script/destroy



- » si definimos las tareas en el fichero Rakefile generado en la raíz del plugin, sólo será accesible ejecutando rake desde ese directorio
- » útiles para ejecutar tareas administrativas internas del plugin, como generación de documentación o testing

T E S T I N G

- » desde la raíz de la aplicación si ejecuto `rake test:plugins` lanzo el testing de todos los plugins previa carga del environment
- » desde el directorio de un plugin, si ejecuto `rake test` prueba sólo ese plugin. La tarea está definida en el Rakefile y por defecto no carga el environment

LIMITACIONES

- » ejecutar testing de plugins que afectan al modelo de datos es complejo. No sabemos a priori qué modelos tiene el usuario
- » es necesario definir ficheros de fixtures y programar la creación de modelos de prueba sobre la marcha

LIMITACIONES

Técnica aportada por Sergio Gil Pérez de la Manga, inmediatamente después de mi charla en la Conferencia Rails, con una solución simple para poder probar modelos sin necesidad de la complejidad de crearse una tabla previamente. ¡Brillante!

```
ActiveRecord::Base.class_eval do
  alias_method :save, :valid?
  def self.columns() @columns ||= []; end

  def self.column(name, sql_type = nil, default = nil, null = true)
    columns << ActiveRecord::ConnectionAdapters::Column.new(name.to_s,
      default, sql_type, null)
  end
end
```

```
class Resource < ActiveRecord::Base
  column :id, :integer
  column :name, :string
  column :amount, :integer
  column :conditions, :boolean
end
```

Luego por ejemplo en el setup de los tests o donde te haga falta puedes hacer
Resource.new...

C O N C L U S I O N E S



J R A M I R E Z @
A S P G E M S . C O M

F O R M A T I N T E R N E T
W O R D P R E S S . C O M

javier.ramirez.gomara@gmail.com
<http://formatinternet.wordpress.com>

obra publicada por javier ramirez como 'Atribución-No Comercial-Licenciar Igual 2.5' de Creative Commons

Imágenes de las letras bajo licencia CC. Autor Leo Reynolds. Descargadas de Flickr