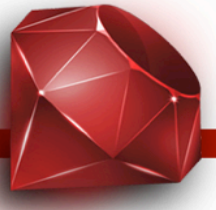
Procesando peticiones

Rodrigo Rodriguez
rorodr@gmail.com
Skype: rarodriguezr

Action Pack

- Consta de 3 módulos:
  - Action Dispatch:
    - Dirige las peticiones al controlador
  - Action Controller:
    - Convierte las solicitudes del usuario a respuestas (usando Vistas)
  - ActionView:
    - Permite dar formato a la información por mostrar al usuario

Action Dispatcher



- Se encarga de organizar la integración entre los URLs que digita el usuario y los controladores encargados de atender la solicitud.
- En Rails se incentiva el uso de REST en las rutas, por lo que Action Dispatcher provee formas simples para generar “rutas” de este tipo, las cuales se declaran en el archivo “routes.rb”

RESTful(1/2)

- REST ó Representational State Transfer: trata de facilitar la manera en que se representa la transferencia de la información.
- Se basa principalmente en 2 principios:
  - Utilizar identificadores del recurso como parte del URL, con el fin de representarlos fácilmente.
  - Transferir representaciones del estado de ese recurso entre componentes del sistema (métodos)

DELETE /photos/17

http://www.infoq.com/articles/rest-introduction

RESTful(2/2)

- **Métodos HTTP:**
  - **GET**: devolver un valor
  - **POST**: almacenar un valor
  - **PATCH**: modificar un valor
  - **DELETE**: Eliminar un elemento
- URLs válidos para PHOTOS:

GET /photos/17

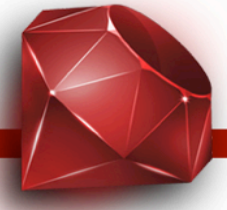
POST /photos

PATCH /photos/17

DELETE /photos/17

GET /photos

routes.rb



- Ubicado en config/routes.rb
- Archivo que permite configurar rutas para la aplicación
- Rails provee 2 formas distintas para declarar rutas:
  - *"Resourceful" o simplificada*: Permite definir rutas basadas en recursos (ej. Modelos) y relacionarlos con métodos estándar en los controladores
  - *Comprensiva*: Permite definir URLs específicos para realizar acciones, establecer condiciones, etc

Rutas: Resourceful (1/4)

- Forma recomendada para declarar rutas en las aplicaciones
  - `resources :contacts`

Verbo HTTP	PATH	Controller#action
GET	/contacts	photos#index
GET	/contacts/new	photos#new
POST	/contacts	photos#create
GET	/contacts/:id	photos#show
GET	/contacts/:id/edit	photos#edit
PATCH/PUT	/contacts/:id	photos#update
DELETE	/contacts/:id	photos#destroy

Rutas: Resourceful (2/4)

- Crear rutas “resourceful” genera además algunos helpers accesibles en los controladores y vistas para generar los URL correctos:
  - `contacts_path => /contacts`
  - `new_contact_path => /contacts/new`
  - `edit_contact_path(:id) => /contacts/:id/edit`
  - `contact_path(:id) => /contacts/:id`
- NOTA: existe una versión “_url” para cada uno

Rutas: Resourceful (3/4)

- Es posible crear rutas tipo “resource” (en singular), las cuales se diferencian en que no crean el URL para listar todos los elementos
- A veces ocupamos hacer “Namespaces” en nuestros controller (y rutas), para ello ocupamos hacer que el controlador tenga el namespace (ej.: Admin) y en las rutas poner:
  - `namespace :admin do`
  - `  resources :contacts, :stores`
  - `end`

Rutas: Resourceful (4/4)

- Algunas veces se desea anidar recursos (por ejemplo, que un contacto pertenece por fuerza a una tienda), para ello se usa:

```
resources :stores do
    resources :contacts
end
```

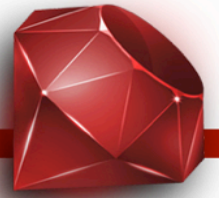
- Esto genera un URL semejante a: `/stores/:store_id/contacts`

Para más información de rutas visitar:
http://guides.rubyonrails.org/routing.html

Rutas: Comprensiva (1/4)

- En su forma más simple, permite generar rutas basados en parámetros:
  - `get ':controller(/:action(/:id))'`
    - Los paréntesis representan secciones opcionales
    - NOTA: Por seguridad no es bueno agregar esta ruta pues se exponen los métodos públicos del controlador.
  - Algunos URLs válidos (y sus métodos) son:
    - '/contacts' => ContactsController#index
    - '/contacts/active' => ContactsController#active
    - '/contacts/active/1' => ContactsController#active (con un parámetro llamado ':id' con valor 1)

Rutas: Comprensiva (2/4)



- A las rutas anteriores se les pueden agregar secciones dinámicas y estáticas:
  - `get ':controller/:action/:id/:user_id'`
    - Envía un parámetro adicional (:user_id) al método
    - Un URL válido sería: /contacts/create/1/21
  - `get ':controller(/:action(/:id))', controller: /admin\/[^\/]+/`
    - Verifica que el controlador tenga la restricción de iniciar con "admin/"
  - get ':controller/:action/:id/with_user/:user_id
    - Obliga al usuario a escribir el texto 'with_user' como parte del URL

Rutas: Comprensiva (3/4)

- También se puede generar rutas con URLs específicos y relacionarlas con los métodos
  - `get ‘mis_productos’, to: ‘products#mine’`
    - Si un usuario utiliza: “/mis_productos?name=carro”, en método ‘mine’ del controlador products va a recibir un parámetro llamado ‘name’ con valor ‘carro’.
  - get 'photos/:id', to: 'photos#show', defaults: { format: 'jpg' }
    - Obliga al usuario a agregar el formato en el URL, por ejemplo “/photos/1.jpg”

Rutas: Comprensiva (4/4)

- Se pueden establecer nombres a una ruta:
  - `get 'exit', to: 'sessions#destroy', as: :logout`
- Se pueden establecer restricciones a las rutas
  - `match 'photos', to: 'photos#show', via: [:get, :post]`
  - `get 'photos/:id', to: 'photos#show', constraints: { id: /[A-Z]\d{5}/ }`
- Se puede redirigir acciones directamente desde la ruta, usando:
  - get '/stories', to: redirect('/articles')

Action Controller

- Parte primordial del desarrollo del sitio, pues hace la relación entre la vista y el modelo
- Un controlador empieza su función una vez que el enrutador solicita la ejecución de una acción en un controlador específico y deja su responsabilidad una vez que se presenta al usuario la información con el formato esperado.

Controladores

- En Rails un controlador es una clase que hereda de ApplicationController y tiene sus métodos propios, los que pueden ser públicos, privados y protegidos

```
class StoresController < ApplicationController
  def new
  end
  private
  def sample_calculation; end
end
```

Controlador: Proceso

- Considerando el controlador anterior, el proceso completo de interacción sería:
  - Usuario digita '/stores/new' en su navegador
  - Routes crea instancia de StoresController
  - Método 'new' atiende la petición y ejecuta la acción (en este caso crear un nuevo Store)
  - Se presenta al usuario el la vista (html) '/stores/new.html.erb'

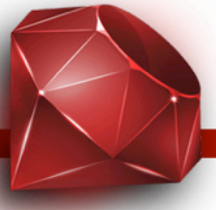
Controlador: Parámetros

- El usuario puede enviar parámetros al servidor de diversas maneras:
  - :String parameters': Son aquellos parámetros que forman parte del URL en el archivo routes (/stores/:id). A manera de ejemplo: '/stores/1'
  - 'Query parameters': Son los parámetros que están luego de '?' en los URL o en en los casos de POST, como un parámetro no visible en el URL.
- En el controlador, se reciben como un hash, accesible por medio del método "params"

Controlador: Parámetros

```
class StoresController < ApplicationController
    def create
     @client = Client.new(params[:client])
     if @client.save
       redirect_to @client
     else
       render "new"
     end
   end
end
```

Parámetros: Hashes y arreglos



- Por lo general no se generan únicamente params unidimensionales, sino que hay ocasiones que requieren valores anidados y arreglos, por ejemplo:
  - GET /stores?ids[]=1&ids[]=2&ids[]=3
    - params[:ids] => [‘1’, ‘2’, ‘3’]
  - POST /stores? store[name]=superTienda&store[url]=127.0.0.1
    - params => { store: {name: “superTienda”, url: “127.0.0.1”}}

Parámetros: JSON

- Si en el controlador se recibe un “Content-Type”: "application/json”, al recibir por parámetro un JSON, automáticamente lo convierte a un HASH.
  - POST /stores
    - (payload) { "company": { "name": "acme", "address": "123 Carrot Street" } }
    - params => { company: { name: ‘acme’, address: ‘123 Carrot Street’ } }

Strong Parameters

- Validación de seguridad que bloquea la asignación masiva de variables a un modelo a no ser que hayan sido autorizados.
- Obliga a declarar los atributos válidos usando la clausula “permit”
  - params.permit(:id)
    - Autoriza el uso de ID
  - params.require(:log_entry).permit!
    - Autoriza todos los sub-hash

Strong Parameters

```
class PeopleController < ActionController::Base
  def create
    Person.create(params[:person])
  end
  def update
    person = current_account.people.find(params[:id])
    person.update!(person_params)
    redirect_to person
  end
  private
    def person_params
       params.require(:person).permit(:name, :age)
    end
end
```

Manejo de Sesiones

- En un sitio Web cada solicitud es independiente, en caso de que se quiera "persistir" información entre cada solicitud, se utilizan Cookies o sino Sesiones.
- En el caso de las sesiones, en Rails se pueden configurar diversos almacenamientos:
  - CookieStore, CacheStore, ActiveRecordStore, MemCacheStore
- Semejante a los params, Rails trata la sesión como un Hash, usando el método: "session"

Flash

- Es un tipo de sesión especial, pues únicamente permanece activo hasta la siguiente petición del mismo cliente (misma sesión).
- Es MUY común el uso de Flash para desplegar mensajes luego de almacenar un registro.

```
def destroy
    session[:current_user_id] = nil
    flash[:notice] = "You have successfully
logged out."
    redirect_to root_url
  end
```

Manejo de Cookies

- Información almacenada del lado del cliente
- Puede ser cifrada o sin cifrar
- Se les puede establecer una fecha de expiración o eliminarse desde el lado del servidor.
- Permite compartir información entre el Javascript y el Servidor (NO cifrado)
- Semejante a la sesión, se almacena en un HASH por medio de: ‘cookies’

Mostrando JSON o XML

```
class UsersController < ApplicationController
  def index
    @users = User.all
    respond_to do |format|
      format.html # index.html.erb
      format.xml  { render xml: @users}
      format.json { render json: @users}
    end
  end
end
```

Filtros

- Son métodos que se ejecutan antes, después o durante los métodos del controlador
- Si se desean filtros globales, se pueden declarar en el ApplicationController
- Si se desean filtros únicos para un controlador, se declaran en el controlador específico.
- Es posible declarar en los controladores que ignoren un filtro específico utilizando: skip_before_action

before_action

```
class ApplicationController < ActionController::Base
  before_action :require_login
  private
  def require_login
    unless logged_in?
      flash[:error] = "You must be logged in to
access this section"
      redirect_to new_login_url
    end
  end
end
```

around_action

```
class ChangesController < ApplicationController
  around_action :wrap_in_transaction, only: :show
  private
  def wrap_in_transaction
    ActiveRecord::Base.transaction do
      begin
        yield
      ensure
        raise ActiveRecord::Rollback
      end
    end
  end
end
```

Action Views

- Como se vio antes, cada acción tiene una vista asociada (o al menos debería de tener)
- Las vistas pueden generarse usando distintos sistemas de “maquetado”, siendo el más popular ERB.
- Por lo general las vistas incluyen métodos comunes denominados helpers

Recursos adicionales

- ***Libro:*** *Agile Web Development with Rails 4*
- http://guides.rubyonrails.org/action_controller_overview.html
- http://guides.rubyonrails.org/routing.html