Catalyst::Controller::REST

Catalyst::Controller::REST implementa un mecanismo para crear servicios RESTful en Catalyst. Extiende el mecanismo convencional de despacho de Catalyst para soportar una serie de subrutinas que son invocadas en base al verbo HTTP de la solicitud.

Catalyst::Controller::REST maneja de manera transparente todo el proceso de serialización y deserialización por nosotros.

Synopsis

```
1  package TestServer::Controller::REST;
2  use Moose;
3  use namespace::autoclean;
4
5  BEGIN {extends 'Catalyst::Controller::REST'; }
6
7  sub prueba : Local : ActionClass('REST') {}
8
9  sub prueba_GET {
10     my ($self, $c) = @_;
11     # Retornar 200 OK, con los datos en entity serializados en el body
12     $self->status_ok(
13         $c,
14         entity => {
15             nombre   => 'elnombre',
16             apellido => 'elapellido',
17         }
18     );
19 }
```

Una controladora REST en pocos pasos I

1. Crear una controladora de ejemplo `REST`

   ```
   $ script/testserver_create.pl controller REST
   ```

2. Editamos el archivo `lib/TestServer/Controller/REST.pm`
3. La controladora debe extender de *Catalyst::Controller::REST*

```
1  BEGIN {extends 'Catalyst::Controller::REST'; }
```

4. Creamos una acción base utilizando la clase de acción `REST`

```
1  sub prueba : Local : ActionClass('REST') {}
```

5. Ahora debemos crear una subrutina por cada verbo HTTP que quisieramos utilizar.

Una controladora REST en pocos pasos II

```
1  sub prueba_GET {
2      my ($self, $c) = @_;
3      # Retornar 200 OK, con los datos en entity serializados en el body
4      $self->status_ok(
5          $c,
6          entity => {
7              nombre   => 'elnombre',
8              apellido => 'elapellido',
9          }
10     );
11 }
```

6. Iniciamos el servidor de desarrollo de *Catalyst*.

   ```
   $ script/testserver_server.pl -r
   ```

7. Probamos el verbo GET con `curl`

   ```
   $ curl -X GET -i -H "Content-Type: text/html" localhost:3000/rest/prueba
   ```

Resumen: Todas las peticiones GET hacia la ruta `/rest/prueba` se procesan con la subrutina `prueba_GET` y así sucesivamente para cada uno de los verbos HTTP.

Métodos no implementados

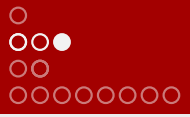
Cualquier petición que utilice un método HTTP no implementado sera respondida con el mensaje *405 Method Not Allowed* y en la cabecera se especificarán los métodos soportados.

Ejemplo:

```
$ curl -X PUT -i -H "Content-Type: text/html" localhost:3000/rest/prueba

HTTP/1.0 405 Method Not Allowed
Connection: close
Date: Tue, 27 Sep 2011 16:00:41 GMT
Allow: GET
Allow: POST
Content-Length: 64
Content-Type: text/plain
Status: 405
X-Catalyst: 5.80032

Method PUT not implemented for http://localhost:3000/rest/prueba
```



OPTIONS

Si no se especifica un `handler`, la controladora va a responder a cualquier petición `OPTIONS` con un mensaje `200 OK`, agregando los verbos permitidos en la cabecera de la respuesta de manera automática.

Serialización

`Catalyst::Controller::REST` va automáticamente a serializar nuestras respuestas, y a deserializar cualquier petición `POST`, `PUT`, `OPTIONS`. Evalua cual serializador debe utilizar, mapeando el tipo de contenido (`content-type`) a un módulo de serialización.

El tipo de contenido (`content-type`) se selecciona en base a:

- **Content-Type Header**, si la petición viene con un atributo `Content-Type` definido, se utilizara este valor.

  ```
  $ curl -X GET -i -H "Content-Type: application/json" localhost:3000/rest/prueba
  ```

- **content-type query parameter**, Si la petición es GET, puedes suministrar el parámetro content-type junto con la lista de parámetros.

  ```
  $ curl -X GET -i  localhost:3000/rest/prueba -d "content-type=text/xml"
  ```

- **Accept Header**, finalmente si el cliente provee un *Accept Header*, se evalua y se utiliza la mejor opción disponible para serializar.

  ```
  $ curl -X GET -i -H "Accept: application/json" localhost:3000/rest/prueba
  ```

Serializadores Disponibles I

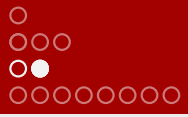
Un mecanismo de serialización dado sólo esta disponible si tiene instalado los módulos subyacentes. Por ejemplo, no puedes utilizar `XML::Simple` si el módulo no está disponible.

Adicionalmente, cada serializador tiene sus peculiaridades en cuanto a que estructuras de datos serán manejadas correctamente.
En la siguiente lista podrá ver los mecanismos de serialización más comunes disponibles.

- `text/x-yaml => YAML::Syck`
  Devuelve un YAML generado por el módulo YAML::Syck
- `text/html => YAML::HTML`
  Utiliza los módulos YAML::Syck y URI::Find para generar YAML con todas las URLs en forma de `hyperlinks`.

Serializadores Disponibles II

- `application/json => JSON`
  Utiliza el módulo JSON para generar la salida en formato JSON. Se recomienda fuertemente tener el módulo `JSON::XS` instalado. El tipo de contenido `text/x-json` es soportado pero esta *deprecated* y al utilizarlo va a recibir mensajes de advertencia en los logs.
- `x-data-dumper => Data::Serializer`
  Utiliza el módulo `Data::Serializer` para generar una salida en el formato `Data::Dumper`
- `application/x-storable => Data::Serializer`
  Utiliza el módulo `Data::Serializer` para generar una salida en el formato Storable de Perl.
- `text/x-php-serialization => Data::Serializer`
  Utiliza el módulo `Data::Serializer` para generar una salida en el formato `PHP::Serialization`



Serializadores Disponibles III

- `text/xml => XML::Simple`
  Utiliza `XML::Simple` para generar la salida en formato XML. Este método no es recomendado para trabajo pesado con XML.

Status Helpers

Dado que gran parte de REST es HTTP, fueron creados los *helpers* de estado. Utilizar estos *helpers* garantiza que la respuesta esta correctamente formada conforme a los códigos, cabeceras y entidades HTTP 1.1.

Estos *helpers* cumplen con la especificación HTTP 1.1. Están implementados como subrutinas convencionales, por lo cual es necesario pasar como primer argumento la variable contexto de Catalyst ($c).

status_ok

Devuelve como respuesta la cadena *200 OK*. Toma una entidad (`entity`) y la serializa.

Ejemplo

```
1  $self->status_ok( $c, entity => { radiohead => "Is a good band!" } );
```

status_created

Devuelve como respuesta la cadena *201 CREATED*. Toma una entidad (`entity`) y la serializa, adicionalmente se debe definir un atributo `location` que contiene la ruta para alcanzar al recurso reicentemente creado.

Ejemplo

```
1  $self->status_created(
2     $c,
3     location => $c->req->uri->as_string,
4     entity => {
5         radiohead => "Is a good band!",
6     }
7   );
```

status_no_content

Devuelve como respuesta la cadena *204 NO CONTENT*. Se utiliza en los casos en que la petición fue correctamente procesada, pero no hace falta devolver información al agente de usuario.

Una respuesta 204 no debe incluir cuerpo de mensaje, y siempre termina con la primera línea vacía despues de la cabecera.

status_multiple_choices

Devuelve como respuesta la cadena *300 MULTIPLE CHOICES*. Toma una entidad (`entity`) para serializarla, la cual debería proporcionar una lista de posibles rutas para alcanzar los recursos. También puede contener un parámetro opcional `location` para definir una ruta de preferencia.

Se utiliza en los casos en que el recurso solicitado se corresponde con algundo de entre una lista de recursos alternativos. Ha de ser el propio agente de usuario (navegador, normalmente) quien elija entre los diferentes recursos.

status_bad_request

Devuelve como respuesta la cadena *400 BAD REQUEST*. Recibe como argumento un hash `messaje` que será convertida en el valor de `error` en la respuesta serializada.

Ejemplo:

```
1  $self->status_bad_request(
2      $c,
3      message => "No puedo darle lo que me pide",
4    );
```

status_not_found

Devuelve como respuesta la cadena *404 NOT FOUND*. Toma un hash `messaje` que será convertido en el valor de `error` en la respuesta serializada.

Ejemplo:

```
1  $self->status_not_found(
2      $c,
3      message => "No puede encontrar lo que estas buscando",
4    );
```

status_gone

Devuelve una cadena `410 GONE` como respuesta. Toma un hash `messaje` que será convertido en el valor de `error` en la respuesta serializada.

`410 GONE`: El recurso ya no está disponible, y no se espera que lo vuelva a estar en el futuro.

Ejemplo:

```
1  $self->status_gone(
2      $c,
3      message => "El documento fue eliminado.",
4    );
```